[Mechanical Translation and Computational Linguistics, vol.8, nos.3 and 4, June and October 1965]

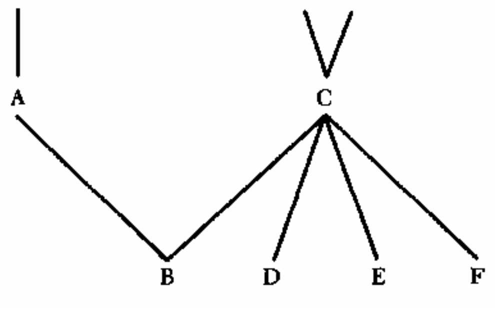
Applications of the Theory of Clumps*

by R. M. Needham, Cambridge Language Research Unit, Cambridge, England

The paper describes how the need for automatic aids to classification arose in a manual experiment in information retrieval. It goes on to discuss the problems of automatic classification in general, and to consider various methods that have been proposed. The definition of a particular kind of class, or "clump," is then put forward. Some programming techniques are indicated, and the paper concludes with a discussion of the difficulties of adequately evaluating the results of any automatic classification procedure.

The C.L.R.U. Information Retrieval Experiment

Since the work on classification and grouping now being carried out at the C.L.R.U. arose out of the Unit's original information retrieval experiment, I shall describe this experiment briefly. The Unit's approach represented an attempt to combine descriptors and uniterms. Documents in the Unit's research library of offprints were indexed by their most important terms or keywords, and these were then arranged in a multiple lattice hierarchy. The inclusion relation in this system was interpreted, very informally, as follows: term A includes term B if, when you ask for a document containing A, you do not mind getting one containing B. A particular term could be subsumed under as many others as seemed appropriate, so that the system contained meets as well as joins, that is, was a lattice as opposed to a tree, for example as follows:



The system was realized using punched cards. There was a card per term, with the accession numbers of the documents containing the term punched on it; at the right hand side of the card were the numbers of the terms that included the term in question. The document numbers were also punched on all the cards for the terms including the terms derived from the document, and for the terms including these terms and so on.

In retrieval, the cards for the terms in the request were superimposed, so that any document containing all of them would be identified. If there was no immediate output, a "scale of relevance" procedure could be used, in which successive terms above a given term are brought down, and with them, all the terms that they include. In replacing D by C, for example, we are saying that documents containing B, E and F as well as C are relevant to our request (we pick up this information because the numbers for the documents containing B, E, and F are punched on the card for C, as well as those for documents containing C itself). Where a request contained a number of terms, there was a step-by-step rule for bringing down the sets of higher-level terms, though the whole operation of the retrieval system could be modified to suit the user's requirements if appropriate.

The system seemed to work reasonably well when tested, but suffered from one major disadvantage: the labor of constructing and enlarging the lattice is enormous, and as terms and not descriptors are used, and as the number of terms generated by the document sample did not tail off noticeably as the sample increased, this was a continual problem. The only answer, given that we did not want to change the system, was to try to mechanize the process of setting up the lattice. One approach might be to give all the pairs of terms,

for example $\overset{A}{\underset{B}{|}}, \overset{C}{\underset{B}{|}}$ and then sort them mechanically to produce the whole structure. The difficulty here, however, is that the person setting up the pairs does not really know what he is doing: we have found by experience that the lattice cannot be constructed properly unless groups of related terms are all considered together. Moreover, even if we could set up the lattice in this way, it w ould be only a partial solution to our

* This document is based on lectures given at the Linguistic Research Center of the University of Texas, and elsewhere in the United States, in the spring of 1963. It is intended as a general reference work on the Theory of Clumps, to supersede earlier publications. The research described was supported by the Office of Science Information Service of the National Science Foundation, Washington, D.C.

problem. What we really want to attack is the problem of mechanizing the question “Does A come above B?” When we put our problem in this form, however, it merely brings out its full horror; how on earth do we set about writing a program to answer a question like this?

As there does not seem to be any obvious way of setting up pairs of terms mechanically, we shall have to tackle the problem of lattice construction another way. What we can do is look at what the system does when we have got it, and see whether we can get a lead from this. If we replace B by C in the example above, we get D, E and F as well; we have an inclusive disjunction “C or B or D or E or F.” These terms are equally acceptable. We can say, to put it another way, that we have a class of terms that are mutually intersubstitutible. It may be, that if we treat a set of lattice-related terms as a set of intersubstitutible terms, we can set up a machine model of their relationship. Intersubstitutibility is at least a potentially mechanizable notion, and a system resulting from it a mechanizable structure. What we have to try to do, therefore, is to obtain groups of intersubstitutible terms and see whether these will give the same result as the hand-made structure.

The first thing we have to do is define 'intersubstitutibility.' In retrieval, two terms are totally intersubstitutible if they invariably co-occur in documents. They then each specify the same document set, and it does not matter which is used in a request. The point is that the meaning of the two terms is irrelevant, and there need not be any detectable semantic relation between them. That is to say, we need not take the meaning of the terms explicitly into account, and there need be no stronger semantic relation between them than that of their occurring in the same document. What we have to do, therefore, is measure the co-occurrence of terms with respect to documents. Our hypothesis is that measuring the tendency to co-occur will also measure the extent of intersubstitutibility. This is the first stage; when we have accumulated co-occurrence coefficients for our terms or keywords, we look for clusters of terms with a strong mutual tendency to co-occur, which we can use in the same way as our original lattice structure, as a parallel to the kind of group illustrated in our example by “C or B or D or E or F.”

The attempt to improve the original information retrieval system thus turned into a classification problem of a familiar kind: we have a set of objects, the documents, a set of properties, the terms, and we want to find groups of properties that we can use to classify the objects. Subsequent work on classification theory and procedures has been primarily concerned with application to information retrieval, but we thought that we could usefully treat the question as a more general one, and that attempts to deal with classification problems in other fields might throw some light on the retrieval case. The next part of this report will therefore be concerned with classification in general.

Classification Problems and Theories; the Theory of Clumps

In classification, we may be concerned with any one of three different problems. We may have

1) to assign given objects to given classes;
2) to discover, with given classes and objects, what the characteristics of these classes are;
3) to set up, given a number of objects and some information about them, appropriate classes, clusters or groups.

1) and 2) are, to some extent, statistical problems, but 3) is not. 3) is the most fundamental, as it is the basis for 2), which is in turn the basis for 1). We cannot assign objects to classes unless we can compare the objects' properties with the defining properties of the classes; we cannot do this unless we can list these defining properties; and we cannot do this unless we have established the classes. The research described below has been concerned with the third problem: this has become increasingly important, as, with the computers currently available, we can tackle quite large quantities of data and make use of fairly comprehensive programs.

Classification can be looked at in two complementary ways. Firstly, as an information-losing process: we can forget about the detailed properties of objects, and just state their class membership. Members of the same class, that is, though different, may not be distinguished. Secondly, as an information-retaining process: a statement about the class-membership of an object has implications. If we say, that is, that two objects are members of the same class, this statement about the relation between them tells us more about each of them than if we considered them independently. In a good classification, a lot follows from a statement of class membership, so that in a particular application the predictive power of any classification that we propose is a good test of its suitability. In constructing a classification theory, therefore, we have to achieve a balance between loss and gain, and if we are setting up a computational procedure, we must obviously throw away the information we do not want as quickly as possible. If we have a set of O_n objects with P_m properties, and P_m greatly exceeds O_n, we want if we can to throw as much of the detailed property information away as is possible without losing useful distinctions. This cannot, of course, necessarily be done simply by omission of properties.

We may now consider the classification process in more detail. Our initial data consists of a list of objects each having one or more properties.* We can conveniently arrange this information in an array, as follows:

* We have not yet encountered cases where non-binary properties seemed necessary. They could easily be taken account of.

$$\begin{array}{cc|cccccc}
 & & \multicolumn{6}{l}{\text{properties}} \\
 & & P_1 & P_2 & \ldots & & & P_m \\
\hline
\text{o} & O_1 & 1 & 1 & 0 & 0 & 1 & 0 \\
\text{b} & & & & & & & \\
\text{j} & O_2 & 0 & 1 & 1 & 0 & 1 & 0 \\
\text{e} & & & & & & & \\
\text{c} & & & & & & & \\
\text{t} & & & & & & & \\
\text{s} & & & & & & & \\
 & O_n & 1 & 0 & 1 & 0 & 0 & 0
\end{array}$$

where O_1 has P_1, P_2, P_5 and so on, O_2 has P_2, P_3, P_5 and so on. We have to have this much information, though we do not need more—we need not know what the objects or properties actually are,—and we have, at least to start with, to treat the data as sacred.

We can try to derive classes from this information in two ways:

1) directly, using the occurrences of objects or properties;
2) indirectly, using the occurrences of objects or properties to obtain resemblance coefficients, which are then used to give classes.

We have been concerned only with the second, and under this heading mostly with computing the resemblance between objects on the basis of their properties. If we do this for every pair of objects we get a (symmetric) resemblance or similarity matrix with the similarity between Oi and Oj in the ijth cell as follows:

$$\begin{array}{c|ccc}
 & O_1 & O_2 & O_3 \ldots \\
\hline
O_1 & & S_{12} & S_{13} \\
O_2 & S_{21} & & S_{23} \\
O_3 & S_{31} & S_{32} &
\end{array}$$

To set up this matrix, we have to define our similarity or resemblance coefficient, and the first problem is which coefficient to choose. It was originally believed that if the clusters are there to be found, they will be found whatever coefficient one uses, so long as two objects with nothing in common give 0, and two with everything in common give 1. Early experiments seemed to support this. We have found, however, in experiments on different material, that this is probably not true: we have to relate the coefficient to the statistical properties of the data. We have therefore to take into account

i) how many positively-shown properties there are (that is, how many properties each object has on the average),

ii) how many properties there are altogether,

iii) how many objects each property has.

Thus we may take account of i) and ii) by computing the coefficient for each pair of objects on the basis of the observed number of common properties, and then weighting it by the unlikelihood* of the pair having at least that number of properties in common on a random basis.

In any particular problem there is, however, a choice of coefficient, though for experimental purposes, as it saves computing effort, there is a great deal to be said for starting with a simple one. Both for this reason, and also because we did not know how things were going to work out, we defined the resemblance, R, of a pair of objects, O_1 and O_2, as follows:

$$R_{O_1, O_2} = \frac{\text{Number of properties in common}}{\text{Total number of different properties}}$$

This was taken from Tanimoto[1]; it is, however, a fairly obvious coefficient to try, as it comes simply from the Boolean set intersection and set union. For any pair of rows in the data array we take:

$$\frac{\text{Number of 1's in the intersection}}{\text{Number of 1's in the union}}$$

This coefficient is all right if each object has only a few properties, but there are a large number of properties altogether, so that agreement in properties is informative. We would clearly have to make a change (as we found) if every object has a large number of properties, as the random intersection will be quite large. In this case we have to weight agreement in 1's by their unlikelihood. There is a general belief (especially in biological circles) that properties should be equally weighted, that is, that each 1 is equally significant. We claim, on the contrary, that equal weighting should be interpreted as equality of information conveyed by each property, and this means that a given occurrence gives more or less information according to the number of occurrences of the property concerned. Agreement in a frequently-occurring property is thus much less significant than agreement in an infrequently-occurring one. If N_1 is the number of occurrences of P_1, N_2 the number of occurrences of P_2, N_3 of P_3 and so on, and we have O_1 and O_2 in our example, possessing P_1, P_2 and P_5, and P_2, P_3 and P_5 respectively, we get

$$R_{O_1, O_2} = \frac{\dfrac{1}{N_2} + \dfrac{1}{N_5}}{\dfrac{1}{N_1} + \dfrac{1}{N_2} + \dfrac{1}{N_3} + \dfrac{1}{N_5}}$$

This coefficient is thus essentially a de-weighting. Though more complicated than the other, it can still be computed fairly easily.

When we have set up our resemblance or similarity matrix, we have the information we require for carry-

* The unlikelihood is theoretically an incomplete B-function, but a normal approximation is quite adequate.

ing out our classification. We now have to think of a definition for, or criterion of, a cluster. We want to say "A subset S is a cluster if . . ." and then give the conditions that must be fulfilled. There are, however, (as we want to do actual classification, and not merely think about it) some requirements that any definition we choose must satisfy:

i) we must be able to find clusters in theory,

ii) we must be able to find them in practice (as opposed to being able to find them in theory).

These points look obvious, but are easily forgotten in constructing definitions, when mathematical elegance is a more tempting objective. What we want, that is, is

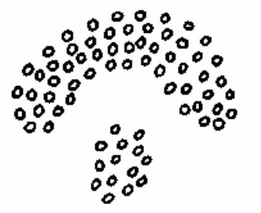
1) a definition with no offensive mathematical properties, and
2) a definition that leads to an algorithm for finding the clusters (on a computer).

We still have a choice of definition, and we now have to consider what a given definition commits us to. Most definitions depend on an underlying model of some kind, and so we have to see what assumptions we are making as the basis for our definition. Do we, for example, want a strong geometrical model? We can indeed make a fairly useful division into definitions that are concerned with the shape of a cluster (is it a football, for instance?), and those that are concerned with its boundary properties (are the sheep and the goats to be completely separated?). Boundary definitions are weaker than those based on shape, and may be preferable for this reason. There are other points to be taken into account too, for instance whether it is desirable that one should allow overlap, or that one should know if all the clumps have been found.

Bearing these points in mind, we may now consider a number of definitions. We can perhaps best show how they work out if we think of a row of the data array as a vector positioning the object concerned in an n-dimensional space.

CLIQUE

(Classes on this definition are sometimes referred to simply as "clusters"; in the present context, however, this would be ambiguous. These clusters were first used in a sociological application, where they were called "cliques," and I shall continue to use the term, to avoid ambiguity, though no sociological implications are intended.) According to our definition, S is a clique if every pair of members of S has a resemblance equal to or greater than a suitably chosen threshold θ, and no non-member has such a resemblance to all the members. In geometrical terms this means that the members of a clique would lie within a hypersphere whose diameter is related to θ. This definition is unsatisfactory in cases where we have two clusters that are very close to, and, as it were, partially surround, one another. Putting it in two-dimensional terms, if we have a number of objects distributed as follows:



they will be treated as one round clique, and not as two separate cliques, although the latter might be a more appropriate analysis.

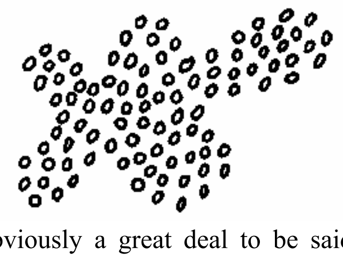
This approach also suffers from a substantial disadvantage in depending on a threshold, although in most applications there is nothing to tell us whether one threshold is more appropriate than another. The choice is essentially arbitrary, and as the precise threshold that one chooses has such an effect on the clustering, this is clearly not very satisfactory. The only cases where a threshold is acceptable are those where the clustering remains fairly stable over a certain range of the threshold. This is hard to define properly, and there is no evidence, experimental or theoretical, that it happens.

IHM CLUSTER

The classification methods used by P. Ihm depend on the use of linear transformations on the data matrix, with a view to obtaining clusters that are, in a suitable space, hyperellipsoids. An account of them may be found in Ihm's contribution to *The Use of Computers in Anthropology.*[2]

This definition is unsatisfactory because it assumes that the different attributes or properties are independently and normally distributed, or can be made so.

Both these definitions depend on fairly strong assumptions about the data. Ihm, for example, is taking the typical biological case where the properties may be regarded as independently and normally distributed within a cluster. If these assumptions are justified, this is all right. But in many applications they may not be. In information retrieval, for instance, the following might be a cluster:



There is obviously a great deal to be said, if we are trying to construct a general-purpose classification procedure, for making the weakest possible assumptions. The effects of these definitions can usefully be studied in more detail in connection with the similarity matrix. First, for cliques. Suppose that we re-arrange the matrix to concentrate the objects with resemblance above θ, given as 1, in the top left-hand corner (and bottom right). Objects with less than θ resemblance, given as 0, will fall in the other corners. Ideally, this should give the following*:

$$\begin{array}{c|c} 1111 & 0000 \\ 1111 & 0000 \\ 1111 & 0000 \\ 1111 & 0000 \\ \hline 0000 & 1111 \\ 0000 & 1111 \\ 0000 & 1111 \\ 0000 & 1111 \end{array}$$

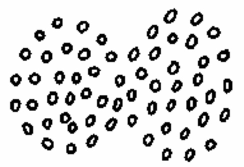
However, consider the following:

$$\begin{array}{c|c} 1101 & 1100 \\ 1101 & 1001 \\ 0011 & 0001 \\ 1111 & 0000 \\ \hline 1100 & 1111 \\ 1000 & 1111 \\ 0000 & 1111 \\ 0110 & 1111 \end{array}$$

One would want, intuitively speaking, to say that the first four objects form a cluster. But on the clique definition this is impossible, because of the 0's in the first 4-square. In fact we have found, with the empirical material that we have considered, that the required distribution never occurs; raw data just does not have this kind of regularity, at worst if only because it was not written down correctly when it was collected. Even with θ quite low, one would probably only, unless the objects to be grouped were very inbred, get pairs or so of objects. In the information retrieval application this definition has the added disadvantage that synonyms would never cluster because they do not usually co-occur, though they may well co-occur with the same other terms. The moral of this is that we should not look for an "internal" definition of a cluster, that is, one depending on the resemblance of the members to each other, but rather for an "external" definition, that is, one depending on the non-resemblance of members and non-members. The first attempt at such a definition was as follows: S is a cluster if no member has a resemblance greater than a threshold *6* to any non-member, and each member of S has a resemblance greater than θ is some other member.† In terms of our resemblance matrix we are looking, not for the absence of 0's in the top left section, but for the absence of 1's in the top right section. We may still, however, not get satisfactory results. For example, the anomalous 1 in the top right corner of the matrix below means that the first four objects do not form a cluster, although we would again, intuitively speaking, want to say that they should.

$$\begin{array}{c|c} 1111 & 0010 \\ 1101 & 0000 \\ 1011 & 0000 \\ 1111 & 0000 \\ \hline 0000 & 1111 \\ 0000 & 1111 \\ 1000 & 1111 \\ 0000 & 1111 \end{array}$$

This definition again may work fairly well in biology, but it suffers, like the clique definition, from the problems connected with having a threshold. It also means that if we have a set of objects as follows



they will be treated as one cluster and not as two slightly over-lapping ones. On this definition, that is, we cannot separate what we might want to treat as two close clusters.

These definitions all, therefore, suffer from the major disadvantage that a single aberrant 0, in the first case, or 1, in the second, can upset the clustering, and for the kind of empirical material for which automatic classification is really required, where the set of objects does not obviously "fall apart" into nice distinct groups but appears to be one mass of overlaps, and where the information available is not very reliable, as in information processing, definitions like these are clearly unsatisfactory. In many applications, that is, the data is not sufficiently uniform or definite for us to be able to rely on the classification not being affected in this way.

What we require, therefore, is a definition that does without θ, and is not affected by a single error in the matrix. We can get a lead on a definition by looking at the matrix distributions for the other definitions. Considering for the moment the first four rows of the sample matrix, we found that our previous cluster definitions were not satisfied for the first four objects if there was a 0 in the left half of any of the first four rows, or a 1 in the right half; we wanted, that is, to have either the left half of each row all 1's, or the right half all 0's. An obvious modification would be to say that there should be more 1's in the left half than in the right half of each of these rows, without saying that there should be no 0's in the left, or 1's in the right

* These matrices have been drawn in this way for illustrative purposes. In any real similarity matrix successive objects would almost certainly not form a cluster, and one would have to rearrange it if one wanted them to do so (though this is obviously not a necessary part of a cluster-finding program). One would not expect an equal division of the objects either: in all the applications so far considered a set containing half the objects would be considered to be too large to be satisfactory. (In the definition adopted both the set satisfying the definition and its complement are formally clusters, though only the smaller of the two is actually treated as a cluster).

† This definition was the first to be tried out in the C.L.R.U. research on classification under the title of the Theory of Clumps; in this research clusters are called "clumps" and these clusters were called "B-clumps."

half. This would clearly be a move in the right direction, away from the extremes of the other definitions. It would mean, for example, that the following distribution would give us a clump.*

$$\begin{array}{c|c}
1101 & 1100 \\
1101 & 1001 \\
0011 & 0001 \\
1111 & 0000 \\
\hline
1100 & 1111 \\
1000 & 1111 \\
0000 & 1111 \\
0110 & 1111
\end{array}$$

A definition on this basis was adopted for use in the C.L.R.U. research, where a cluster was called a “clump,” as follows: A subset S is a cluster, or *clump,* if every member has a total of resemblances to the other members exceeding its total of resemblances to non-members, and every non-member has a greater total of resemblances to the other non-members than to the members. At present, “total of resemblances” may be taken as “total of resemblances exceeding θ”; however, this use of a threshold may be dropped, and the total is then simply the arithmetic sum of coefficients.** The complement of a clump is thus a clump. There are many equivalent forms of this definition. For instance: If, in the previous matrix diagrams, we label the clump in the top left section “A,” and its complement in the bottom right “B,” we can define “the 'cohesion' of A and B”: Let C be the total of resemblances between any two sets of objects. We can set up a ratio of resemblances

$$\frac{CAB}{C\,AA + C\,BB}$$

which we call the “cohesion across the boundary between A and B.” A partition of the matrix marking off a clump will correspond to a local minimum of C. Let A be the resemblance matrix. We set up a vector v defining a partition of the total set of objects, with elements +1 for objects on one side of the partition and — 1 for those on the other. Q is a diagonal matrix defined by the equation

$$Av = Qv \bullet$$

Since the elements of v are all + or — 1, the multiplication Av simply adds up, for each element, the resemblance to the members of the subset specified by + 1 and subtracts the resemblance to the other elements specified by —1. Thus, it is clear that if the subset specified by +1 is a clump, the entries in the result vector Av will have to be positive in those rows where v *is* positive, and negative elsewhere. This corresponds to the case in which all the elements of Q are positive.

There is clearly some relation between clumps and the eigenvectors of A corresponding to positive eigenvalues, but we cannot say just what this relation is. This approach does not, moreover, lead to any very obvious procedure for clump-finding. In matrices of the order likely to arise in classification problems, the solution of the eigenproblem would almost be a research project in itself. If we could get over this difficulty we might abandon as too difficult the attempt to relate eigenvalues and eigenvectors to clumps as defined, and try to set up some other definition of a class in which the connection was more straightforward. Investigation shows, however, that the interpretation of eigenvectors and eingenvalues as the specification of a class is not at all obvious. This approach is also open to the methodological objection that information is abandoned at the end, and not at the beginning, of the classification process.

We may, however, still learn something useful from considering these alternative definitions, and the equation defining the cohesion of A and B indeed suggests that an arbitrary partition of our set of objects with interchanges of objects from one side to the other to reduce the cohesion between the two halves can be used as a clump-finding procedure. As this is used as the basis of the procedures we have developed, we can now go on to consider the question of programming.

Programming Procedures for the Theory of Clumps

In programming, the first step is to organize the data into some standard form. We have found it most convenient to list the properties and attach to each property a list of the objects that have it. Listing the objects with their properties is much less economic, as the data is usually very sparse. (The data can of course be presented to the machine in this form, as it can be transformed into the desired form very easily.) The properties and objects can be identified with serial numbers, so that if one were dealing with text, for example, one would sort the words alphabetically and give each distinct word a number.

The next stage is to set up our similarity matrix. This is done in two stages, collecting the co-occurrence information, and working out the actual similarity coefficients. In the first, we consider each property in turn, and count one co-occurrence for each pair of objects having the property; we are thus only opening a storage cell for the items that will give positive entries in the similarity matrix. The whole is essentially a piece of list-processing, in which we list our objects, and for each item in the list we have a pointer to a storage cell containing information about the object concerned. As we can store only information about the relation between the given object and one other object in a cell, we require a cell for every object with which

* It was found expedient to treat the diagonal elements (which carry no information anyway) as zero rather than units. This makes the algorithm easier to describe and implement.
** These clumps have been called GR-clumps in earlier publications.

a particular object is connected. These are arranged in the serial order of the objects, with each cell pointing to the next one. The objects connected with a given object are thus not linked directly with this object, but are given in a series of storage cells, each leading to the next.

If we are given n objects, we have, for any one of the objects, n—1 possible co-occurrences with other objects (by co-occurrences, we mean possession of a common property). We could therefore have a chain of n—1 empty storage cells attached to each item in our object list, and fill in any particular one when we found, on scanning our property lists, that the object to which the chain concerned was connected and the object with the serial number corresponding to the cell had a common property. This would, however, clearly be uneconomic, as we would fill up our machine store with empty cells, and only use a comparatively small proportion of them. What we do, therefore, is open a cell only for each object we find actually co-occurring with a given object, when we are scanning our property lists. We will thus, as we go through our property information, add or insert cells in our chains. As we shall not meet the objects in their serial order,* but want to store them in this order, we have to allow insertion as well as addition in our chains of storage cells.

We may find also that two objects have more than one property in common. When we open a cell for a co-occurrence, we record the co-occurrences as well as the objects that co-occur; the next time we come across this pair of objects we add 1 to our record of the number of co-occurrences, and so on, adding to the total every time the two objects come together. (It should be noticed that as co-occurrence is symmetrical we will need** a cell under each of the objects, and will record the co-occurrences twice).

What we are doing, therefore, is accumulating information by list-processing, either opening new cells for new co-occurrences, or adding to the total of existing co-occurrences. Each storage cell contains the name of an object, the number of times it has co-occurred with the object to which the chain concerned is attached, and a pointer to the next cell in the series. As this looks rather complicated when written out, even though the principle is very simple, we can illustrate it with a small example as follows:

P = property, O = object, ( ) = storage cell, → = "go to";

Data	P1 : Ol	O5	O8
	P2 : Ol	O5	O7
	P3 : 03	04	

Store Ol
O2
O3
.
.

* Because the initial data comes with serially-ordered properties.

** The duplicate storage of the co-occurrence information doubles the size of the matrix, but makes it much easier to handle.

Operations

1. Scan P1 list; Ol, O5 co-occur; open cell for O5 under Ol, for Ol under O5; note 1 co-occurrence in each; the entry for Ol now reads:

   O1 → (O5,1)

   for O5:

   O5 → (O1,1)

2. Scan P1 list; Ol, O8 co-occur; open cell for O8 under Ol, for Ol under O8; note 1 co-occurrence in each; the entry for Ol, with the new cell added to the existing chain now reads:

   O1 → (O5,l) → (O8,1)

   for O8:

   O8 → (O1,1)

3. Scan P1 list; O5, O8 co-occur; open cell for O8 under O5, for O5 under O8; note 1 co-occurrence in each; the entry for O5, with the new cell added now reads:

   O5 → (Ol,l) → (O8,l)

   for O8, with the new cell added:

   O8 → (Ol,l) → (05,1)

4. Scan P2 list; Ol, O5 co-occur; add 1 to the co-occurrences totals for O5 under Ol, for Ol under O5; the entry for Ol now reads:

   Ol → (O5,2) → (O8,l)

   for O5:

   O5 → (Ol,2) → (O8,l)

5. Scan P2 list; Ol, O7 co-occur; open cell for O7 under Ol, for Ol under O7; note 1 co-occurrence in each; the entry for Ol with the new cell inserted now reads:

   O1 → (O5,2) → (O7,1) → (O8,1)

   for 07:

   O7 → (Ol,l)

6. Scan P2 list; O5, O7 co-occur; open cell for O7 under O5, for O5 under O7; note 1 co-occurrence in each; the entry for O5, with the new cell inserted now reads:

   O5 → (O1,2) → (O7,1) → (O8,l)

   for O7, with the new cell added:

   O7 → (O1,1) → (O5,1)

7. Scan P3 list; O3, O4 co-occur; open cell for O4 under O3, for O3 under O4; note 1 co-occurrence in each; the entry for O3 now reads:

   O3 → (O4,l)

   for O4:

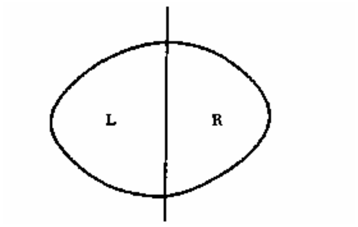
   O4- → (O3,1).

When this information has been collected it is transferred to magnetic tape in a more compact form, in which the name of each object is given, together with a list of all the objects it co-occurs with, with their respective total co-occurrences. The matrix is thus stored in a form in which it can be easily updated if necessary. Some other information is also included: the total number of objects each object co-occurs with, and the total number of properties it has. This gives us all the information we need for working out any similarity coefficient. When we have worked out our coefficient for each pair of objects, we replace the co-occurrence totals by the appropriate similarity coefficients. Our entry for O9, say, might read:

O9 : O2 = .35, O4 = .07, O28 = .19,...........

The serial list we obtain is our similarity matrix, and we are now in a position to start clump-finding.

This is where the matrix terminology introduced earlier is useful. What we want to obtain is a partition of our set of objects, into, say L and R, such that we have a clump and its complement. If we imagine our set and a partition as follows



what we have to do is consider the sets of objects on each side of the partition to see whether they form clumps, and if they do not, try moving objects across the partition until we get the required distribution. To see whether a set is a clump, we have to take each object in turn and sum its connections to the set and complementary set respectively.

The initial partition will be defined by a vector v, and we can, as we saw, obtain the diagonal matrix Q in the equation

$$Av = Qv$$

after multiplying the similarity matrix A by v. We know that if all the elements of Q are positive, we have found a clump. If we have a negative element in Q, this means that the partition is unsatisfactory, either because we have an object in R which should be in L, or an object in L which should be in R. (The sign attached to the corresponding element of v will tell us which). We can deal with the anomalous object by shifting it across the partition,* but we have to see what effect this has on our two sets. We mark the shift by reversing the sign of the element in v which corresponds to the negative element in Q, and then use the new vector, defining the new partition, to recompute Q. If we still have a negative element in Q, we repeat the whole process. We thus have an iterative procedure for improving an unsatisfactory Q by removing the next negative element in the series. Rectifying one negative element can mean that we get others that we did not have to start with, but it can be shown that the procedure is monotonic.

The important point is that we carry out the whole multiplication Av only once; after this, as we are only dealing with one element of Q, corresponding to one object, at a time, we have only to consider one row of A. We have, that is, changed only one element of v, and therefore have only to carry out the multiplication

* Thus diminishing the cohesion between the two sets.

on the corresponding row in A to get the new result vector Av. This all means that the procedure is quite economic, and that we can store A, row by row, in a fairly compact form. Recomputing Q is not a very serious operation. We have to do it all because we are dealing with the totals of connections between objects, and shifting one object could affect the totals for all the other objects in our set.

We can describe this iterative series of operations, in which we modify our initial partition, as one round of clump-finding; we will either find a clump, or finish up with all our objects on one side of the partition. When we do not find a clump, that is, it is because we have, in trying to improve on our initial division, moved all our objects onto one side of the partition, so that the whole partition collapses. After each round, whether we find a clump or not, we have to start again with a new partition. It is clear that the way we partition the set initially can influence our clump-finding; it can also affect the speed with which we find clumps. Again, when we start a new round, we want to take account of the partitions we have already made. We obviously do not want to repeat a partition we have already tried, and we may also be able to take account of previous partitions in a more sophisticated way. How, then, should we set up our partitions, either to begin with, or for a new round? How should we set about getting a useful partition?

We first tried using some very crude cluster, which we had found by another method, as a sort of “seed”; it would partition off a potential clump. In one experiment, for instance, we used cliques as starting points. This is not, however, very satisfactory. In many applications we have found that we cannot obtain any cliques, and so cannot use them as a lead; this was true of the information retrieval application, with which we were most concerned at the time, so we did not pursue the approach. The procedure is also rather inefficient; it is no better than other methods, and involves the additional preliminary stage in which the crude clusters are set up.

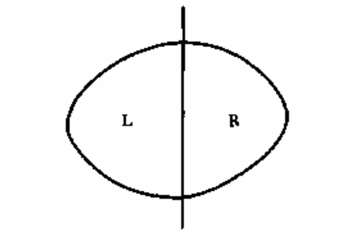
We then thought that as we have an iterative procedure, we could start with a random equipartition; we can start, that is, in a comparatively simple-minded way because clump-finding is not a hit or miss affair: we can improve on whatever division we start with. When we start a new round, we make another equipartition, though we found it more efficient if partitions after the first are not made at random, but are adjusted so that we do not start with anything too close to the partitions we have already tried.* We thus have a kind of orthogonal series of equipartitions.

This procedure has, however, one defect: although we sometimes find a clump, in general any partition that we make is far too likely to collapse. The whole process becomes a succession of collapses, each fol-

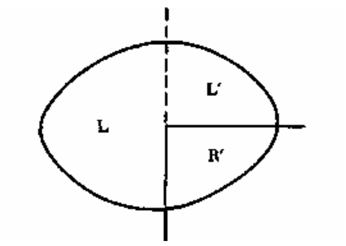
* This is effected by a rule for modifying the vector v.

lowed by an entirely new start. This is unfortunate, because although a given partition is not right, something near it may well be, and this is clearly worth looking for. We found that we could avoid the unfortunate consequences of a collapse by using a binary section procedure. When we fail to find a clump, we take successive binary sections, with respect to our starting partition, inspecting each in a round of iterations, either until we find a clump or the binary chopping reaches its limit. We thus have a series of rounds, and not merely one round associated with each starting partition, each testing a partition which is a modification of the original one.

The actual procedure is as follows: Suppose that we partition our set into two parts, L and R, with the elements of L corresponding to +1 in our vector, and those of R to —1:

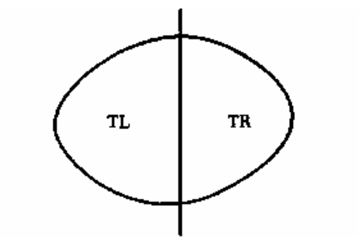


Now suppose that we carry out our iterative scan and transfer, and find that L collapses. We do not start afresh with a quite independent partition, but try to give L a better chance: we inspect R, find the mean total of resemblances to L, and restart with the elements with greater than average resemblance to the old L in a new L:



We now scan again, and with a bigger L, may find that it no longer collapses.

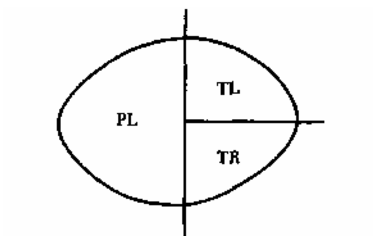
We can illustrate the process in more detail by using the notions of "temporary," T, and "permanent," P. We label our initial parts TL and TR:



Suppose we find on iterating, that L collapses, and we want to give it a better chance. We make alterations as follows:

TL becomes		P1
TR	"	TL ("best" half)
		TR (rest)

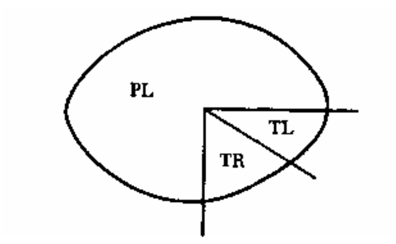
In any subsequent partition the permanent part stays permanent, while the temporaries are reconsidered. Suppose we have



and L still collapses. We then set up:

PL becomes		PL
TL	"	PL
TR	"	TL ("best" half)
		TR (rest)

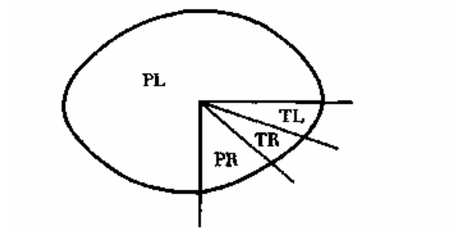
that is



Suppose we now find that R collapses, and we must give it a better chance. We now set up:

PL becomes		PL
TR	"	PR
TL	"	TL ("best" half)
		TL (rest)

that is



The procedure thus consists of a continual reduction of the temporary sections, in an endeavor to build up the permanent sections in a satisfactory way.

If we find a stable partition where neither side collapses, this gives us a clump. It in fact gives us a clump and its complement, which is also formally a clump, though we only treat the smaller one of the two as a clump in listing our results. If we go on partitioning until there are no more elements to partition,* we have failed to find a clump, and have to start all over again with a wholly new division of our set. In any given attempt at clump-finding, therefore, we are always concerned with a partition which has some relation to our initial one, as we want to find out whether anything like the one we started with will give us a clump; and as we think that it is worth making a fairly determined search for one, we go on trying until it becomes clear that there is none. It is clear that this improved procedure for clump-finding is a general one and can be used with any method of choosing starting-partitions; thus if we have an application where we think that we can suitably use other clusters as seeds, we start with them and then go about our clump-finding this way. The procedure as it stands can be usefully refined in a number of ways; in many applications we are not interested in clumps with only two or three members, and so there is no point in carrying on the partition procedure when one side is very small. We can avoid this if we redefine 'collapse', so that, for instance, we say that a partition has collapsed if one side has, say, less than 10, elements in it. In some applications we may be interested in clumps centered on particular elements, or have reason to think that particular elements will lead to clumps; if this is the case we can start with a single element, making our initial partition between this element and the rest of the set. We will clearly get an initial collapse, as all the element's connections will be to the other side of the partition, but after this we can proceed.

Setting up the initial partition between one element and the rest has in fact turned out to be a better way of starting in general. The trouble with equipartitions is that they tend to lead to aggregate clumps. The definition of 'clump' is such that the union of two clumps may be a clump, and if we start clump-finding by considering half of a large set of objects, we are very likely to find that the nearest clump is a large one which is an aggregate of smaller ones. This is not necessarily a bad thing, but we found that the aggregates we got in our experiments were too big to be suitable for the purpose for which the classification was required. Starting with one element avoids this difficulty, and as we have a clump-finding procedure in which the collapse of a partition is not fatal, we can begin with a partition which cannot but collapse, but from which we may be able to derive the kind of clump we want.

This procedure seems to work satisfactorily, though some problems do arise: we do not know

1) when we have found all the clumps, or
2) how many there are to be found.

* Experience shows that the total disappearance of elements to partition is most unusual.

These facts are most objectionable. They illustrate an important aspect of work on classification at present, namely that approaches that are amenable to theoretical treatment are not good in practice, largely because they embody assumptions that are often inapplicable to one's data, whereas approaches that do seem to work in practice are very unamenable to proper theoretical analysis. Until a method is found that can both be theoretically analysed, and works well on real data, we cannot be satisfied. We are, however, convinced that the way to progress at present lies through experiment, A valuable aid at this point is to have an operational test of the usefulness of the classification found. If such a test is available, we may simply continue to find clumps until it is satisfied. It is at any rate possible that such tests connected with the usefulness of the product may continue to be more helpful than theoretical termination rules; they need, after all, to be satisfied regardless of what the theory predicts.

Within these limits we want to be as efficient as possible. We want to find clusters quickly, and if there are quite different ones to be found, to find at least some of them, and we can legitimately use any information as an aid. We may, for instance, find that we can use an existing classification, or clusters found by some other, perhaps rather crude, method, as a starting point. This kind of thing is not always possible or appropriate, and we may have or want to apply our procedure to our data without making any assumptions about it at all. In this case we may be able to make our procedure more efficient for example by looking for clumps centered on a particular element that has not already occurred in a clump; we can note when we have found the same or very similar clumps, so that we start somewhere different.

3) We may get into another difficulty over our resemblance coefficients: many of these coefficients are rather small, and we have to decide the precision that we should store them to, as this can affect the size of the clumps we find. For example, suppose that we have an element x in L: we may find that x is pulled to R by the aggregate of its very small resemblances to members of R, when we want to keep it in L, as it genuinely fits into the L-clump. We can counteract this tendency only by making L bigger, which may be unsatisfactory for other reasons. We have found, however, that this defect may nevertheless be turned to advantage, because we can use this information as a parameter in relation to the clumps we require.

The definitions and procedure just described have been worked out over a period of time and have been tested on different kinds of material. They are not at all regarded as perfect, and in fact are subject to continual improvement. They have, however, reached a stage where they can be applied fairly easily, and their various applications will therefore be considered next.

The Application of the Theory of Clumps to Information Retrieval

The most important application of the Theory of Clumps has been to information retrieval, and this will therefore be described in some detail. We saw that we might be able to group terms on the basis of their co-occurrence in documents, and then use these clusters in the way in which immediately-connected sets of terms were used in the original C.L.R.U. library scheme. This system currently contains some 2000-odd terms, and any attempt to classify them will therefore be a large computing problem; this is in spite of the fact that the library is quite small—nearly 800 documents. This point emphasizes the main problem in automatic classification, which is the quantity of data we must be able to handle. Automatic classification procedures are no use unless they will deal with realistic amount of material—their basic purpose, indeed, is to be able to deal with more data than human beings can manage. In the information retrieval case, therefore, we need to be able to deal with large numbers of terms.

We have discussed the general problem of defining co-occurrence measures, but it is perhaps worth looking at the problem in connection with a particular application in some detail. There is no great difficulty about finding co-occurrence measures, as they can depend only on a small number of factors. Putting them in a specifically information retrieval form we have:

1) the number of times each pair of terms co-occurs;
2) the number of times each term occurs;
3) the number of terms for each document in which each pair of terms co-occurs (that is, has each document got 2 or 50 terms);
4) the number of terms altogether;
5) the number of documents altogether.

The only ones involving any computation are 1) and possibly 3). 1), as we saw, gives us a matrix; in the most recent information retrieval experiment we had 342 terms (representing some 400 documents), which would give us a 342^2 matrix. From a computing point of view this is quite impracticable,* and we have to hope that the potential matrix is fairly empty, and can therefore be stored economically. In most of the experiments so far carried out the matrices were quite empty, and we have to hope that this will usually be the case. In the library case, for example, there were only some 12,000 entries out of a possible total of nearly 125,000. The best way of tackling the problem of choosing the most suitable coefficient is to try a number of alternatives, accumulating information under 1) and 3), and evaluating the results in relation to 2), 4) and 5). As we saw, the best coefficient may vary with the application, and there is much to be said for choosing a simple one. In the information retrieval experiments we began with the first one we defined, (A), and after we had found that the second, (B), worked better in another case, adopted it for the library classification as well, where it also gave more satisfactory results.

* This is not strictly correct. A 342^2 matrix, even if it is quite full, can be handled, but the matrix does not have to be much bigger before it cannot be handled when full, and we must take this point into account when we design our programs.

We now have to consider our requirements on the classes to be found, if we are to use only mechanically generated clusters, and not make any modifications of our data or results by hand:

1. We must be able to cope with synonyms and near-synonyms.
2. The classes must not be too large, or the disjunction we get when we replace a term by its class will be too long. (This means that we lose discriminating power when we retrieve. We get the same result if we proceed by too many steps at once in the lattice in the scale or relevance operation in the old library system. If our clusters are too small, on the other hand, we are liable to find that we are not generalizing enough.)
3. We must be able to obtain overlapping classes. (This is to cover cases where keywords are used in different ways, and corresponds to meets in the lattice structure, where one term leads to two or more terms on the next higher level. The number of meets in the existing lattice show that if we were to look for mutually exclusive classes we would either fail to find any, or get very bad ones. The situation is obviously different, if we are clustering documents.)

The definitions and procedures we have outlined do satisfy these requirements. We can cope with synonyms, because we have not defined a clump in such a way that every pair of members has to be directly connected; experiments have shown that we do not get either very large or very small clumps, because our definition is neither so weak that any set of elements that are connected chain-wise, for example, will satisfy it—in which case we would get very large clusters, —nor so strong that we would only get sets in which every pair of elements is directly connected—in which case we would probably get very small clusters; and we can get overlapping classes, because total resemblances can be compounded in different ways.

The following are some sample clumps:

1. Grammar, paradigm, parts of speech, adjective, preposition, phrase, phrase marker, tense, ending, stem, syntax, diacritic.
2. Number, deductive system, Brouwerian terms, intuitionism, logic, observation.
3. Phrase marker, kernels, string operation, transformational grammar, terminal language, Markov process, finite state process.
4. Term abstract, descriptor, request, term vocabulary.
5. Style, text, paragraph, information retrieval, classifying, indexing, term abstract, thesaurus head, Roget, thesaurus, descriptor, encode, bits, partial ordering, Boolean algebra, confounding, request, parody, term vocabulary, reality.
6. Style, text, paragraph, chunking, interlingua, posteditor,

source language, target language, thesaurus head, Roget, Pask machine, Latin, technical language.

Some of these, for example No. 3, are reasonable by any standards; others, which appear less obvious, are quite reasonable given the character of the documents from which they are derived. No. 2, for instance, is based on a number of papers on intuitionist mathematics ('Brouwerian terms' is a portmanteau term for several exotic terms occurring in some papers by Brouwer that we did not think it worth treating separately). No. 5 comes mostly from C.L.R.U. workpapers on the use and encoding of a thesaurus, though there is some "noise" in the clump. No. 6 is also from C.L.R.U. workpapers, this time early ones on machine translation. These two clumps show how the same terms, when used in different contexts, may appear in different clumps. The way in which the classification reflects the documents is especially shown by No. 1; the clump is quite plausible, but does not include all the parts of speech; we have papers dealing with, for example, Russian prepositions, but not with Russian conjunctions.

What we now have to consider is what we do with our clumps when we have found them. We know that we want to use them to the same effect as our original lattice, but something more detailed than this is required as a specification of how they are to be used. We start with a set of documents and their keyword lists; from this information we derive a set of clusters of keywords. What have we gained by doing this? The only way to answer this question is to set up a system embodying the clusters and test it in retrieval. This, however, only brings us up against the problem of how to test any system we devise. Testing any retrieval system involves a vast amount of work, and we get into the evaluation problem as soon as we start: how are we to say whether a retrieval system is any good or not? We can really only see how it works out in practice, and this is rather late. We want to find out whether a proposed system is likely to work quite soon.

The most useful approach seems to be to try a number of variations and compare them; if we change our resemblance coefficient, or our clustering procedure, we can see what effect this has on a given body of material. This is of course only an internal comparison; we can clearly learn more if we compare a system constructed on this sort of basis with one that is constructed in a completely different way. But this is just what is so difficult, and we must therefore set our sights lower. We have not got out of the evaluation difficulty either; we can compare the effects of any changes we make in our procedure, but we have to have some means of evaluating them as well. What can we do if we do not want to go to the lengths of constructing a whole retrieval system and trying it out for, say, a year or two?

1. We can simply look at the clumps we get. The trouble here is that our intuitive criteria of what looks all right may in fact be inapplicable; we may, that is, get clusters that do not look all right, but in fact are entirely plausible, given the data from which they are obtained. In grouping index terms we tend to assume that the clusters we get will have some obvious kind of conceptual or semantic coherence, for instance, that they will all be concerned with one subject, or be a collection of terms that are more or less synonymous. We may not get clusters like this at all, and when we look into the question we can see that there is no real reason why we should. The clusters we get are after all based on the co-occurrence of terms, and while this usually means that there is some conceptual connection between them, it may not be a very obvious or straightforward one. We may also have terms that are being used in exotic ways, and this will affect the way they cluster. What this all means is that we have to take the character of our sample into account, and if we get apparently odd results, to check back with the documents. We may also get into difficulties if we have to inspect large clusters. In some of the other applications of the procedure the clusters are so large that they can hardly be looked at properly, and this could easily happen in the information retrieval case.

2. We may be able to learn something from transposing the results, as follows: we can form a list of clusters for each keyword, and then sort so that we can compare the keywords with the same clusters. These should have something in common. This method is again of only limited usefulness, because although clusters with the same lists usually do have something in common, there are hardly any keywords with identical lists, and we are therefore faced with deciding whether the lists are sufficiently alike for the comparison to be valid.

3. We may alternatively try a third method as follows: suppose that we set up an index code for our documents by taking the list of keywords for each document, and replacing it by the list of clusters in which these terms occur. We will thus obtain a cluster list for each document. We can then sort and compare the documents with the same cluster lists. This is a better approach to the problem than 1 or 2 if we have a reasonably-sized library, though it does mean that we have to be fairly familiar with the contents, or the whole process will take too long. This third method of evaluating any classification we get is essentially testing it by retrieval, and we have in fact tried the crudest possible retrieval system using our automatically-obtained classification, without any hand modification.

If we obtain cluster lists for documents, as described, we have found in our experiments so far that this gives a gross classification of the documents, which is satisfactory as far as it goes, but does not go far enough. Keywords alone, on the other hand, resolve the documents too much.* The obvious approach to try, there-

* This is not necessarily true of any keyword-using system. The keyword classification in our own system, for example, is very nearly just right, but it is not quite right, and something less refined seems to be required. There is equally no theoretical reason why a classification in terms of clusters should be too crude; it just turned out that way in this case.

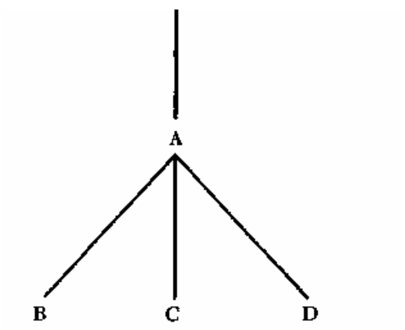
fore, is to combine the two; for each document we have a term list and a cluster list, and we treat each request in the same way. A request as it stands gives terms, and we can look up the clusters for these in the same way as we do for the terms derived from the documents. We can then carry out our retrieval in two stages. Retrieving on clusters alone is too crude, but retrieving on keywords alone is inefficient over a whole library. Retrieval on keywords, that is, is all right if we get an exact match with a document list, but not otherwise, and as we hardly ever do get an exact match, we want an alternative approach by which we can get better results than inexact matching with keywords will give us.

The most efficient procedure is not to match our request term list against the whole library, but only against the term lists derived from documents that have a cluster list relevant to the request cluster list. We make, that is, a first-stage selection from the library as a whole by matching cluster lists, and then a second stage selection from this subset of the library by matching keyword lists. We select the documents with a cluster list relevant to the request by taking all those whose cluster list includes that of the request, and we pick out all the documents with more than one keyword in common with the request.

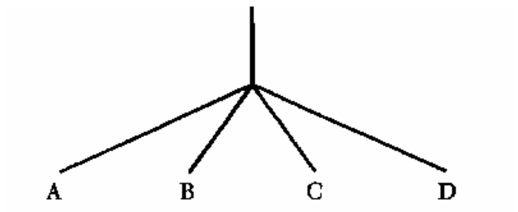
Although we could evaluate the results only by using our knowledge of the library, this seemed to work quite well. An interesting point is that we did not look at the clusters we used in the process; we generated them and then used them without inspecting them. The classification was thus judged solely by the quality of its retrieval output. It might be argued that we cannot be sure that the procedure is more efficient (in output as opposed to operation) if we use clusters as well as keywords, rather than keywords alone. We can, however, find out whether the clusters do anything for us by carrying out retrieval in the opposite way. We can pick out all the documents with keywords in common with the request, and then select those that have cluster sets including that for the request. We tried this out and got very good results. A document with, for example, three terms in common with the request *and* cluster inclusion was more relevant to the request than one with five keywords in common but not cluster inclusion. A similar test is to make requests using random sets of terms. If we match a random set with the documents sets we may not get much, but we will probably get something. If, however, we try a cluster match as well, we do not get any retrieval at all. This again shows that the clusters do make retrieval more effective.

This was all very much in the way of a first stage experiment. Clump inclusion as a criterion for matching may well be too simple. We might find that we could get better results by refining the cluster lists for documents and requests in some way. Instead of taking all the clusters generated by the keywords concerned, we could try taking the intersection of the sets generated by the keywords, and use only those clusters that recur to represent our document or request, or we could select the most frequent clusters.

We should now look at our automatic classification from the point of view of the specific problem that we started with: how to construct the library lattice. We wanted to see whether we could set up our term lattice automatically, and we should now, therefore, see whether we can actually construct a lattice for our terms like the one we would have set up by hand. (This is not to say that the hand-made and machine-made lattices for the same set of terms should be identical; we want only to see whether we can construct lattices from clumps.) We do not have enough information, however, to do exactly what was done by hand. When we had our lattice we could say that



could be replaced by 'A or B or C or D'. When we start with the latter, however, there is nothing to tell us which is the inclusive term; should A include the other three, or B, or C, or D? The only lattice structure we can obtain is



and this is not the same thing. There is also more overlap in the mechanically-generated system than in the human one, so that any lattice we tried to set up would be more complicated.

The real point, however, is whether we really do want to obtain a lattice structure, and specifically, a lattice like the original one. It may have been what we started out to try to get, but it is not clear that we need it. We want a classification to do a job, namely, organize a set of documents in such a way that we can retrieve from them effectively, and the way to test our classification is therefore to try it out directly in retrieval. There is no reason for thinking that the classificatory structure we obtain by one method will be like the earlier one we obtained by a different method, nor is there any reason why they should be. The point is that we are concerned with two classifications that are to be used for the same purpose, but not with two classifications that are to be the same, and the methods we use to construct them are too different for the results to be comparable.

One or two final points can appropriately be made in connection with the use of automatic methods for information retrieval in general. The library has to be a handleable size. This is not determined simply by the number of documents in it, but varies with its content. The determining factor is the number of keywords and keyword co-occurrences. This in turn means that the level of detail of analysis that the keywords represent, and the number per document, is relevant. We may thus be able to classify quite a large library if we keep the number of keywords in the document lists down.

We can appropriately conclude this section on classification and information retrieval with a suggestion for a future experiment. As we saw, we can either start from scratch in our clump-finding, or use "seeds" or "proto-clumps" obtained in some other way. The use of existing classes as a lead suggests an experiment of a thoroughly down-to-earth character, and one that is perhaps more relevant from the point of view of the ordinary working librarian than many of the ones carried out in this field. What we can do is make use of any existing library or semantic classification, such as the U.D.C., Roget's Thesaurus, or the ASTIA Technical Thesaurus, in a quite straightforward way. We can include pieces of the U.D.C. or a thesaurus in our data just as if they were keyword lists representing documents. We cannot, of course, take the precise structure of a set of related terms into account, but can only list them, though by treating any such piece as the list for a document, we are treating them as connected, and therefore are indirectly taking their structure into account. We have also to make our own choice of the pieces we take out of the existing classifications, but the use that we are going to make of them is such that the particular details are not important, and so the problem of whether we have chosen "properly" does not matter. We then give these terms a fairly low weighting, so that these previously-constructed classes will not affect the classification we want to get from our actual documents too much. What we essentially want to do is to use these pieces of other classifications as rather nebulous proto-clumps to be modified by what we get from the documents we are really concerned with. It might be objected that if we are going to do things like this we might just as well not bother with pure automatic classification procedures at all. The point, however, is that we are relying on our automatic procedures in constructing a classification for the documents to hand; it is just that existing classification schemes are based on a great deal of experience, and if we can make use of them as an aid, without accepting them as gospel, there is obviously something to be said for doing so. We give the terms we take over a low weighting specifically in order that they shall not influence our machine classification too much. What we want them for is to give us something to start with, which will either be reinforced by what we get from the documents, and which will therefore turn into a genuine clumps, or be cancelled out by the different clumps that we get from the genuine terms. All we have to do is adjust our weightings so that the terms we take over do not, as it were, turn non-clumps into clumps.

Other Applications of the Theory of Clumps

We have, as we said earlier, tried out the procedures we have constructed on material derived from quite different fields. We thought that any classification scheme would work better for any particular application if it had some general validity, and was not designed for one purpose only, and we found, in practice, that what we learnt from one application was often useful in another.

The most instructive application of the procedure, apart from the information retrieval one, was to some semantic material. This is fully described elsewhere,[3] and will only be described here in sufficient detail to show how it is connected with the general research on classification. The object of the research is to construct a thesaurus-type classification, and to carry out as much of it as possible automatically. The data consists of small sets of words that are synonymous in at least one context; these "rows" are to be grouped on the basis of common words to give conceptual groupings of the kind exemplified by the sections in existing thesauri such as Roget's. A clump, that is, consists of semantically similar rows. The results of tests on 500 rows (some 350 words) so far carried out are satisfactory, but the sample is not really large enough. The need for classification programs that will deal with large quantities of material is indeed very well exemplified by linguistic applications, where we have to be able to deal with at least several thousand objects.*

As in the library case, we were faced with the evaluation problem. We set out to produce something like *Roget's Thesaurus*[4] but this was not a detailed enough requirement to tell us whether any particular clump was satisfactory, and even if we felt, on intuitive grounds, that the clumps obtained were plausible, this was again not enough when it came to deciding whether each member of a clump should indeed be a member, and also whether any non-member should be a member. There did not, however, seem to be any alternative to just looking at the results, and we therefore had to hope that this was sufficient. (This appli-

* The size of the classification problem is well brought out by the fact that good dictionaries and thesauri may contain hundreds of thousands of words. Our only hope at present is to divide the material we have to classify up into subsets, perhaps in alternative ways, and deal with them separately.

cation was the one which showed up the defects in the first similarity definition.)

The program has also been applied to some anthropological material. This consisted of a list of American Indian tribes each characterized by the rituals they practiced in connection with the puberty of their young girls. There were 118 tribes and some 120 rituals altogether. The program again worked fairly well, though some difficulties arose over “doubtful” entries in the data array; these were read as 'yes', and it turned out afterwards should have been read as 'no'. The results of the automatic classification could fortunately be compared with the (very carefully carried out) hand classification for which the material was originally collected. The classification we obtained was substantially the same as the earlier one, and what differences there were were mostly due to the doubtful entries.

An early version of the program was used for a small and rather tentative experiment in the classification of blood diseases. The data consisted of a list of 180 patients with their symptoms (there were 32 symptoms altogether), and the classification was a genuinely “blind” one, as we did not know what the symptoms, which were merely given by numbers, were. We were in this case able to compare the results with the conventional classification of the diseases, and we found that we had got the same groupings, with the exception of one disease, which is indeed very ill-understood, and is not defined by a well-marked set of symptoms.

We have also carried out an experiment in archaeological classification. This concerned 500 Neolithic pottery beakers, each represented by a list of decorative and other properties (45 altogether). The beakers were to be grouped on the basis of their properties. This application brought up a number of general problems about data to be classified. We have argued that we have to treat any data as sacred, that we should assume that the properties concerned are independent—if not in the pure logicians' sense, at least from a common-sense point of view.* We started out, too, with the belief that all the properties were equally informative, that is, that their relative frequency of occurrence did not vary too much. In this application, as in the semantic case, we found that we had to take the number of times a property occurred into account, and therefore that the second similarity definition was more appropriate than the first. We also found that some of the properties were so obviously dependent on one another, that we had to take this fact into account. A large number of the properties, for instance, were concerned with various different forms of decoration, such as zig-zag, or hatching, but there was also a property which was simply “being decorated,” which was clearly associated with any particular form of decoration. The archaeologist concerned said that these general properties could be disregarded, but there is something unsatisfactory about having to deal with the problem in this way. Otherwise the clumps obtained were plausible, and could again be checked independently, for example by stratificational evidence.

We have also used the program for a rather different purpose, which is instructive as a comparison, as it in a sense concerns division rather than grouping. Given the logical diagram for a computer consisting of nodes and connections between them, we want to divide it (for handling and printing purposes) into sections, and preferably into sections with the smallest number of connections to any other section. We are looking, that is, for the areas with the least cohesion between them. This is not a question of classification, except in a second-order derivative sense, but it is a useful application of the program, as it saves a great deal of hand work and is quite efficient.

* For the abstract theoretical issues, see Carnap, *Der Logische Aufbau der Welt*[5], and Goodman, *The Structure of Appearance*[6].

Conclusion

It is to be hoped that these notes on the different applications of the classification procedures we have developed are sufficient to show that we have a general theory of classification, and to support the argument that a generally-applicable theory is more use in any particular application than an individually-tailored one. We have in any case constructed a program which can be applied to any data that can be presented in the form of a binary incidence array, and as most information can be given in this form, this does mean that we can try out our procedure without much difficulty. We have indeed concentrated on experiment rather than theory in our research, because we felt that we could learn more from it. A lot of work has been done on classification theory, but very little on automatic classification practice, and we thought that even if more work should be done on the theory, there was more to be learned from trying to develop computer techniques, which are still in their infancy, than from refining on already refined theories.

Received August 11, 1964

References

1. Tanimoto, T. T., “An elementary mathematical theory of classification and prediction,” I.B.M. Corporation, New York, 1958.
2. Ihm, P., “Methoden der Taxometrie,” Eur 1671.D, Euratom, Brussels 1964
3. Sparck Jones, K., “Experiments in semantic classification,” *Mechanical Translation,* this issue.
4. Roget, P. M., *Thesaurus of English Words and Phrases,* Penguin Books, London, 1953.
5. Carnap, R., *Der Logische Aufbau der Welt,* Berlin, 1928.
6. Goodman, N., *The Structure of Appearance,* Harvard University Press, Cambridge, Mass., 1951.