

A Computational Model for the Data Compression Metaphor

Chris Thornton
Cognitive and Computing Sciences
University of Sussex
Brighton
BN1 9QH
UK

Email: Christopher.Thornton@firenet.uk.com
Tel: (44)1273 678856

May 21, 2003

Abstract

Although it is often noted that the process of *data compression* forms an illuminating metaphor for concept learning mechanisms, the computational basis of the metaphor is not usually made explicit. In this paper, an algorithm is described which implements `climbing-tree` inductive generalisation as a form of clustering process. The algorithm is correctly classified as a generalised data compression algorithm; it therefore provides a plausible model for the data compression metaphor.

1 Introduction

It is frequently noted that there appears to be a close relationship between concept learning and data compression [cf. 1,2,3,4,5]. Wolff, in fact, has made the idea that concept learning can be construed as a form of data compression a central feature of his research on grammar acquisition [6,7,8,9]. Other researchers have observed that concept learning mechanisms appear to produce data compression as a side-effect [e.g. 10,3,11,12] and that *compressibility* can be successfully used as a selection criterion for inductive inference [13,14,15,16,17].

A recurring theme in this general area of research is the idea that data compression forms an illuminating *metaphor* for concept learning processes. But its not clear how seriously we should take this metaphor. The general appraisal seems to be that it is a helpful aid, but not to be taken literally at the level of computational explanations. Whether or not this appraisal is correct is an open

question [18]. If there is a deep and robust analogy between concept learning and data compression then it may substantially understate the case.

A demonstration of a computational relationship between the two processes in question would tend to validate those approaches which treat the concept learning task *as* a data compression task; i.e. which try to solve concept learning problems by invoking data compression processes [cf. 18,6,7,8,9]. This paper attempts to provide such a demonstration by showing that there is a data compression algorithm which can implement a simple form of concept learning; namely, inductive generalisation [cf. 19,20,21].

2 Generalised data compression

The basic observation underlying the data compression metaphor is that a concept definition provides a representation of the objects (or events) covered by the concept, which is more compact than the union of the representations of the objects. On this view, deriving a concept definition is, in some sense, the same thing as deriving a more compact representation for a set of objects; and that it is therefore equivalent, in some sense, to data compression applied to the original representations.

Now, in ordinary data compression it is always possible to perfectly recover the original data from the compressed representation. But it is a general property of concept learning techniques that concept definitions do *not* perfectly identify the set of instances covered by the concept [22]. This implies that the process invoked in the data compression metaphor must be a sort of **generalised** data compression; i.e. a process which is not guaranteed to produce representations from which the original data can be deterministically recovered. Now obviously, concept learning processes do not produce just any representation – they try to produce the optimal representation. But the question is, how can we put a precise (i.e. computational) meaning on this?

A convenient approach involves reading **optimal** as **optimally accurate**; i.e. it involves assuming that concept learning processes produce the best representation of the input objects which can be produced of the given *size*. We will describe this as *generalised data compression*. The distinction between generalised and ordinary data compression can be clarified as follows.

An ordinary data compression algorithm is guaranteed, by definition, to generate an output from which the input data can be deterministically recovered. A generalised data compression algorithm, on the other hand, is *not* guaranteed to do this. Instead, it is guaranteed to generate an output which preserves the maximum amount of information from the input given a constraint on the *quantity* of output.

Given this definition, the data compression metaphor can be said to construe concept learning in terms of generalised data compression rather than standard data compression. Thus, in order to build a computational model for the data compression metaphor, we need to build a generalised data compression algorithm and show the case(s) in which it implements a concept learning process.

3 Choosing an input language

A first step involves deciding on an input language, i.e. a data format in which the algorithm to be developed will accept input. The input language should be a suitable medium in which to present concept learning input; i.e. it should be a general purpose language for describing instances.¹ It should also provide a suitable context in which to formalise the notion of generalised data compression.

One language having both these properties is a slightly constrained form of vector space (i.e. vectorial) notation, called the *point-space* notation below. In vector space notation, descriptions correspond to points in some N-dimensional space. Thus, a description of an instance such as

large wedge

might be represented as a single point in a 2-dimensional space whose dimensions range over the values {large, medium, small} and {sphere, pillar, brick, wedge}; cf. Figure 5.

A body of vector space data is said to be in *point-space format* if it satisfies the following two constraints. (i) the data have a purely compositional semantics; that is to say a configuration of points (data elements) has no global semantic properties which are not computationally derivable from the semantic properties of the individual points in the configuration; (ii) there is a distance function over points in the space (i.e. over descriptions) which can be interpreted as a measure of dissimilarity.

Thus, vector space data constitute point-space data provided that there is a *dissimilarity* function on pairs of data elements, and provided that the semantic properties of the data can always be derived via a purely compositional process. Below, the vector space in which (point-space) data are expressed as points will be referred to as the *description space* for the data. The *configuration* of a description space will be assumed to involve the specification of an averaging function for coordinate values.

4 Derivation of the algorithm

Given the description of generalised data compression provided above, a generalised data compression algorithm for point-space data can be characterised as an algorithm which, presented with some input data D, generates a data structure which is the best representation of D which can be constructed of the given size. We can arrive at a specification for such an algorithm via the following thought experiment.

Consider the point-space data depicted in Figure 1. The data are points in a 2-dimensional description space where the dimensions range over ordinary numeric values and the distance (dissimilarity) function is **euclidean** [23]; i.e.

¹Throughout, the word **instance** is use **example**.

the data are just points in an ordinary 2-d array. Imagine that we want to produce an optimally accurate representation of these data which consists of just two data elements; i.e. imagine we want to find the best representation of these data which **gets rid of** one data element from the original.

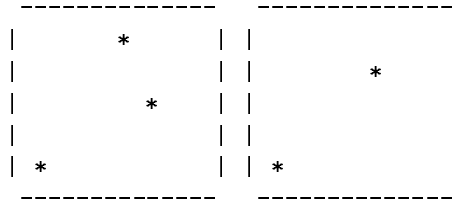


Figure 1

Figure 2

The constraints on the data allow us to describe the precise form of this new version: it will consist of a set of two data elements such that the average distance between points in the original and their nearest neighbours *in the representation* is minimised. This can be shown as follows.

The difference between any two bodies of point-space data is, by definition, equal to the average distance between *corresponding* points² in the two bodies of data.³ Therefore an optimally accurate version of a specific body of data is one which minimises the average **nearest-neighbour** distance (i.e. the distance between points in the original and their nearest neighbours in the representation).

Extending this line of thought a little further, we note that an optimal (N-1)-point version of an N-point body of data D must be a version which substitutes the two closest points in D with a single *centroid* point (since, by definition, the *centroid* of a collection of points P minimises the distance between itself and all of P).⁴ Thus the optimal 2-point representation of the data depicted in Figure 1 will be roughly as depicted in Figure 2.

A number of conclusions can be drawn from this thought experiment. Firstly, it is the case that a mechanism which constructs optimally compact representations of bodies of point-space data will have to find groups of close points in the original data and **fuse** them; i.e. construct their centroid.

In order to get rid of more points from the original (i.e. in order to produce an output containing less points) the mechanism would have to be less fussy about how close the to-be-fused points actually are. Thus, to generate the entire range of possible outputs (the 1-point representation, the 2-point representation and so on), it would have to work up through the range of inter-point distances, fusing increasingly large groups. At any one stage it would construct a set of clusters such that the distance between any two members of a cluster falls below the corresponding inter-point distance.

²I.e. nearest-neighbour points

³This follows directly from our assumptions about the distance function, and the compositionality of the semantic properties of point-space data.

⁴In cases where there is no unique pair of closest points there will be multiple versions satisfying this description.

As may be clear, this process is functionally similar to a hierarchical clustering process [24,25]. In fact, to be strictly accurate, we should say that since it would appear to be capable of constructing overlapping clusters it is similar to a *clumping* process [26]. But even here the resemblance is not perfect.

The whole point of fusing clusters (in the envisaged process) is to produce representations which effectively show the original data viewed at a lower level of resolution (i.e. in a form where certain close points are **approximated** by a centroid). But, clearly, the fusing of *overlapping* clusters will lead to single data points being approximated by more than one centroid and therefore to the production of representations which are definitionally sub-optimal. Thus an *optimal* representation-constructing mechanism must be based on a version of the envisaged process which *discards* overlapping clusterings.

5 Specification of the algorithm

Given these considerations, an algorithm which constructs the complete range of optimal representations of a body of point-space data can be easily derived. The envisaged clumping process is similar to the method of **mode analysis** clustering [27] which works by identifying **disjoint density surfaces in the sample distribution** [26, p. 33] It also resembles the density-search technique associated with Gengerelli and Charmichael *et al.* [????]. However, it is not isomorphic with any clustering algorithm known to the author.

Initialisation of the algorithm involves setting the distance threshold d to some arbitrary negative value. The main processing loop executes an iteration of clumping and centroid-derivation the steps of which are as follows.

- (1) set d to be the lowest, observed inter-point distance greater than the current value of d .
- (2) form a clustering of data points such that the distance between all the points in any *one* cluster is less than or equal to d .
- (3) if none of the clusters overlap, construct and output their centroid points.
- (4) goto 1.

Obviously, an implementation of the algorithm can be easily derived from this specification. In order to show that it provides a model for the relationship between data compression and concept learning it is necessary to show that it is capable of implementing (i.e. simulating) concept learning *behaviour*. This is most conveniently done by showing the way in which the algorithm simulates *inductive generalisation* as performed by the Focussing algorithm [28,21].

6 Focussing-style inductive generalisation

In general, concept definitions classify sets of instances without explicitly enumerating the elements; thus they correspond to *intensional* descriptions [22].

large pillar

and

small brick

A description is said to *mark* a node whose label it includes. *Generalised* descriptions mark at least one non-leaf node; e.g.

large block

?size wedge

A generalised description D is said to *cover* an instance if the description of the instance can be constructed by gathering together leaf-nodes (i.e. leaf-node values) which are in the subtrees of the nodes marked by D. For example

large block

can be said to cover

large brick

large wedge

since both of these descriptions can be constructed by pairing up leaf-nodes from the subtrees of the nodes marked by the generalised description.

In this framework, an intensional definition of a concept C corresponds to a generalised description D, where the extension of C is just the set of all descriptions which are covered by D. Note that the extension of a non-generalised description is a singleton set containing just the description itself whereas the extension of a generalised description is always a non-singleton set.

In constructing a definition of a concept, the Focussing algorithm works in a very simple way. It constructs and then manipulates two separate, but related concept definitions. One of these identifies the most-general definition which is compatible with the instances seen so far; the other identifies the most specific definition which is compatible with the instances seen so far.⁶

Each time it is presented with a new positive instance, it updates the most specific definition so as to ensure that it covers the new instance. Following presentation of a new negative instance, it updates its most general definition so as to ensure that it does *not* cover the new instance. Thus it always ensures that its most general definition does not cover *any* negative instances and that its most specific definition always covers *all* of the positive instances. If ever it finds that its most specific definition has become identical to its most general definition, it terminates. The concept definition is, then, simply the identified description.

⁶The most general and most specific descriptions correspond to the G and S sets respectively, in Mitchells version-space algorithm [29].

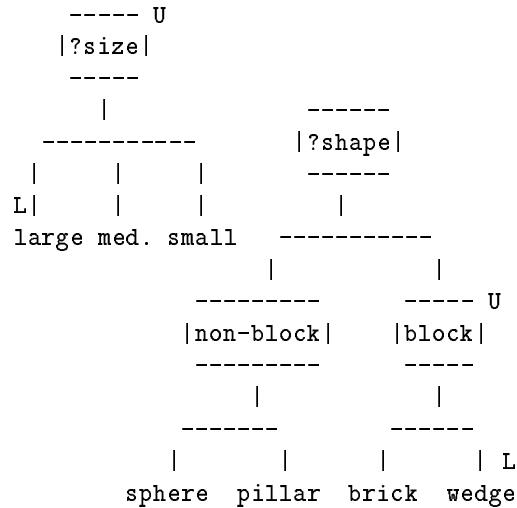


Figure 4. Upper-marks corresponding to most-general description: (?size block), and lower-marks corresponding to most-specific description: (large wedge).

The process by which these working definitions are updated is very simple. The terms in the most general definition mark one node in each tree. Thus the definition can be represented by a complete set of marks (cf. the U characters in Figure 4). The terms in the most specific definition also mark one node in each tree. Thus this definition can also be represented by a complete set of marks (cf. the L marks in Figure 4).

Representing the most general and most specific definitions in terms of explicit marks is very convenient from the point of view of the generalisation and specialisation operations. Generalisation of the most specific definition can now be performed simply by *raising* lower marks while specialisation be performed simply by *lowering* upper marks.

To generalise the description (**large wedge**) so as to ensure that it covers the new positive instance (**large brick**) we simply replace the second term in the description with the label **block**. This obviously corresponds to raising the lower mark in the ?shape tree. To specialise the description (?size **block**) to ensure that it does not cover the description (**small brick**) we need to replace the first term with the label **large**. This corresponds to moving a mark down from the root of the ?size tree to a leaf-node.

Technically, generalisation means raising the lower-mark *in each tree* to the lowest upper bound of the nodes marked by the instances to be generalised (normally the current instance and the existing most general description). Spe-

cialisation, on the other hand, can be carried out in a number of ways [cf. 21, pp. 158-159]. It involves lowering an upper-mark such that the negative instance is no longer covered. In some cases there will be only one possible way to do this. This situation is referred to as a **near miss**. Where there are more than one way to lower the upper mark, the negative instance is a **far miss**.

7 Inductive generalisation in the generalised compression algorithm

In order to clarify the relationship between the generalised compression algorithm and inductive generalisation (as carried out by the Focussing algorithm) it is necessary to show how tree-based descriptions can be rewritten in a vectorial notation. This is relatively straightforward. Imagine that there is a vector space V which corresponds to the description language shown in Figure 4. For each tree in the description language there is exactly one dimension in V . Any one dimension of V ranges over the leaf-nodes of the corresponding tree.

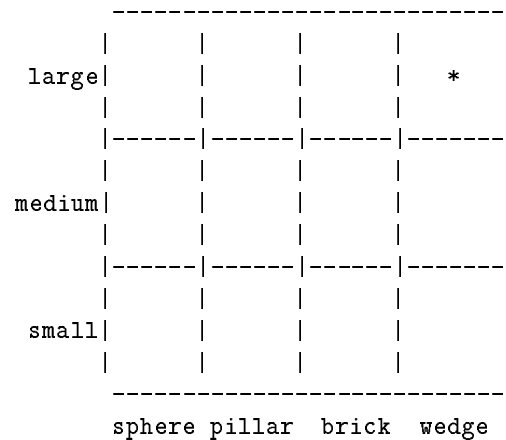


Figure 5. The representation of (large wedge) in vector space notation

Thus a description such as

large wedge

might be represented (in vectorial notation) as a point in V with coordinates equal to **large** and **wedge** (cf. Figure 5).

Note that a complete set of marks in the tree-based description-space identifies a rectangle (or, in general, a hyper-rectangle) in the vector space. In any one dimension the boundaries of the hyper-rectangle enclose the complete range of leaf-nodes in the subtree of the mark for the corresponding tree. Thus a

complete set of upper marks and a complete set of lower marks identify two hyper-rectangles in the vector space.

Since the trees descended from the lower marks will always be subtrees of the trees descended from upper-marks, the hyper-rectangle corresponding to the upper marks will always *enclose* the hyper-rectangle corresponding to the lower marks. Thus we can speak of an outer-hyper-rectangle and an inner-hyper-rectangle (cf. Figure 6.)

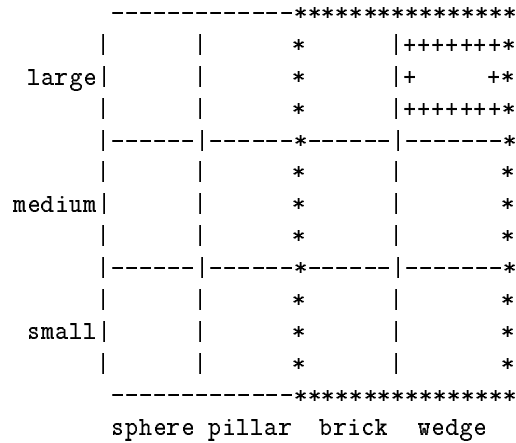
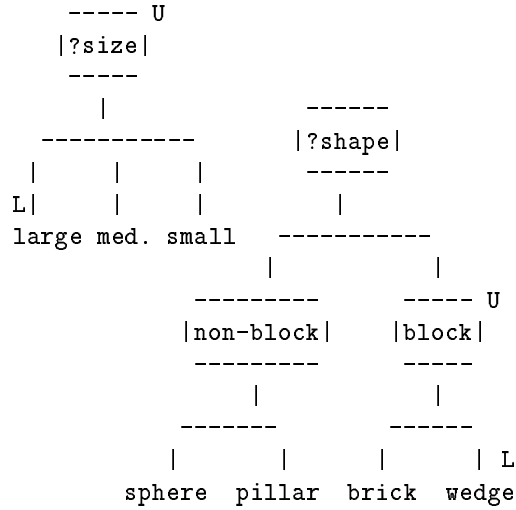


Figure 6. Vector-space hyper-rectangles corresponding to tree marks. Outer hyper-rectangle = upper marks, inner hyper-rectangle = lower

marks.

Having this alternative construal of the description space is useful in that it allows us to directly compare the generalisation step of the Focussing algorithm with the clustering behaviour of the generalised compression algorithm. Suppose that we have a collection of instances in the description language defined by the generalisation trees in Figure 6. We can present these data to the Focussing algorithm or, we can re-represent them as points in V and present them to the generalised compression algorithm.

The basic step in the generalised compression algorithm is the clustering operation. Let us suppose that the construction of a cluster always identifies the smallest hyper-rectangle which encloses the clustered points. In this case the clustering of a set of points can be construed as the identification of a hyper-rectangle, and the expansion of a cluster can be thought of as the expansion of the corresponding hyper-rectangle. In this framework the behaviour of the generalised compression algorithm is commensurable with the mark-raising (i.e. inductive generalisation) behaviour of the Focussing algorithm: both algorithms can be said to construct and then expand hyper-rectangles. We can therefore ask, in what circumstances will the behaviour of these two algorithms be the same?

Assuming that only positive instances are presented (i.e. assuming that the Focussing algorithm implements generalisation only), the behaviour of the two algorithms will be identical just in case the sequence of clusters constructed by the generalised compression algorithm correspond exactly to the sequence of inner-hyper-rectangles constructed by the Focussing algorithm. This will occur if the distance function over points in V is such that the distance between any two data points having coordinates (i.e. leaf-nodes) in the same subtree is always greater, *ceteris paribus*, than the distance between two data points having coordinates in different subtrees.

In other words, the behaviour will be the same if the values of the distance function accurately express the pattern of similarities represented in the generalisation hierarchies. Or, putting it more crudely, the behaviour will be the same if the distance function *corresponds* to the generalisation hierarchies. The reason for this is as follows.

Mark-raising generalisation always moves marks *from* children nodes *to* parent nodes. Thus in expanding a hyper-rectangle, the Focussing algorithm always moves boundaries between positions that enclose complete sets of children nodes. Clustering of points will have the same effect just in case the distance between the children of any given parent is always smaller than the distance between children and non-children. This corresponds to the condition stated above.

The general conclusion is that in the special case where only positive examples are presented and the distance function accessed by the generalised compression algorithm effectively represents the information expressed in the generalisation hierarchies accessed by the Focussing algorithm, the behaviour of the two algorithms will be identical. Moreover, the output produced by the generalised compression algorithm will be identical to the concept definitions

constructed by the Focussing algorithm just in case the **average** of any set of child-nodes (as computed by the averaging function accessed by the generalised compression algorithm) is just the label of the corresponding parent node.

8 Concluding comments

To summarise, the generalised compression algorithm can simulate the inductive generalisation behaviour of the Focussing algorithm. Thus, the algorithm can, in effect, perform a rudimentary form of concept learning. It therefore forms a basic model for the relationship between data compression and concept learning. A further elaboration of the model is presented in [18].

9 Acknowledgements

I would like to thank Ben du Boulay for making helpful comments on an early draft of this paper.

References

- [1] Watanabe, S. (1969). *Knowing and Guessing: A Quantitative Study of Inference and Information*. New York: Wiley.
- [2] Mozetic, I. (1986). Knowledge extraction through learning from examples. In T.M. Mitchell, J.G. Carbonell and R.S. Michalski (Eds.), *Machine Learning: A Guide to Current Research* (pp. 227-231). Boston: Kluwer Academic.
- [3] Rendell, L. (1983). A learning system which accommodates feature interactions. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence: Vol I* (pp. 469-472). Los Altos: Morgan Kaufmann.
- [4] Rendell, L. (1983). Towards a unified approach to conceptual knowledge acquisition. *AI Magazine*, 4 (pp. 19-27).
- [5] Michalski, R. (1983). A theory and methodology of inductive learning. In R. Michalski, J. Carbonell and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Palo Alto: Tioga.
- [6] Wolff, J. (1978). Grammar discovery as data compression. *Proceedings of the AISB/GI conference on Artificial Intelligence* (pp. 375-379). Hamburg.
- [7] Wolff, J. (1980). Data compression, generalisation and overgeneralisation in an evolving theory of language development. *Proceedings of the AISB-80 conference on Artificial Intelligence*. Amsterdam.

- [8] Wolff, J. (1982). Language acquisition, data compression and generalisation. *Language and Communication*, 2 (pp. 57-89).
- [9] Wolff, J. (1984). Cognitive development as optimisation. In L. Bolc (Ed.), *Knowledge Based Learning Systems*. Berlin: Springer.
- [10] Quinlan, J. (1983). Learning efficient classification procedures and their application to chess end games. In R. Michalski, J. Carbonell and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Palo Alto: Tioga.
- [11] Forsyth, R. and Rada, R. (1986). *Machine Learning: Applications in Expert Systems and Information Retrieval*. Chichester: Ellis Horwood.
- [12] Michalski, R. (1987). How to learn imprecise concepts: a method for employing two-tiered knowledge representation in learning. *Proceedings of the Fourth International Workshop on Machine Learning* (University of California, Irvine, June 22-25). Los Altos: Morgan Kaufmann.
- [13] Segen, J. (1985). Learning concept descriptions from examples with errors. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence: Vol II* (pp. 635-636). Morgan Kaufmann.
- [14] Georgeff, M. and Wallace, C. (1984). A general selection criterion for inductive inference. In T. O'Shea (Ed.), *ECAI-84: Advances in Artificial Intelligence* (pp. 473-482). Elsevier Science Publishers B.V. (North-Holland).
- [15] Solomonoff, R. (1964). A formal theory of inductive inference I and II. *Information and Control*, 7 (pp. 1-22 and 224-254).
- [16] Cover, T. and Wagner, T. (1980). Topics in statistical pattern recognition. In K. Fu (Ed.), *Digital Pattern Recognition* (2nd edition) (pp. 15-44). Berlin: Springer-Verlag.
- [17] Salzburg, S. (1985). Heuristics for inductive inference. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence: Vol I* (pp. 603-609). Morgan Kaufmann.
- [18] Thornton, C. (1989). *Concept Learning as Data Compression*. University of Sussex: School of Cognitive Sciences (Doctoral Thesis).
- [19] Winston, P. (1970). Learning structural descriptions from examples. AI-TR-231, Ph.D thesis, Cambridge, Mass.: MIT AI Lab.
- [20] Winston, P. (1975). Learning structural descriptions from examples. In P. Winston (Ed.), *The Psychology of Computer Vision*. McGraw-Hill.
- [21] Bundy, A., Silver, B. and Plummer, D. (1985). An analytical comparison of some rule-learning programs. *Artificial Intelligence*, 27, No. 2 (pp. 137-81).

- [22] Langley, P. and Carbonell, J. (1986). Machine learning: techniques and foundations. Tutorial on Learning and Knowledge Acquisition at ECAI-86.
- [23] Diday, E. and Simon, J. (1980). Clustering analysis. In K. Fu (Ed.), *Digital Pattern Recognition*. Communications and Cybernetics, No. 10 (pp. 47-92). Berlin: Springer-Verlag.
- [24] Romesburg, H. (1984). *Cluster Analysis for Researchers*. London: Wadsworth.
- [25] Sokal, R. (1977). Classification: purposes, principles, progress, prospects. In P. Johnson-Laird and P. Wason (Eds.), *Thinking: Readings in Cognitive Science* (pp. 185-199). Cambridge: Cambridge University Press.
- [26] Everitt, B. (1974). *Cluster Analysis*. London: Heinemann.
- [27] Wishart, D. (1969). Mode analysis. In A. Cole (Ed.), *Numerical Taxonomy* (pp. 282-308). New York: Academic Press.
- [28] Young, R., Plotkin, G. and Linz, R. (1977). Analysis of an extended concept-learning task. In R. Eddy (Ed.), *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (p. 285).
- [29] Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, 18 (pp. 203-226).