Pattern Recognition Pergamon Press 1969. Vol. 1, pp. 219–232. Printed in Great Britain

A Teachable Pattern Describing and Recognizing Program

R. W. SAUVAIN

Department of Computer and Communication Sciences and
Mental Health Research Institute, The University of Michigan

and

L. UHR

Department of Computer Sciences, University of Wisconsin

(*Received* 24 *October* 1968)

Abstract—The paper discusses an adaptive pattern recognition program that learns to recognize coded line drawings, and to describe the structure of the pattern by giving the hierarchical organization of its subparts, and the spatial relationships between them. Emphasis is on mechanisms that allow learning (directed by feedback from a human trainer), and on the methods of abstracting, storing, and retrieving chunks of graphic information for use in subsequent pattern recognition and description. Some similarities to the perceptual process in humans are noted.

1. INTRODUCTION

MUCH pattern recognition research has been oriented towards the problem of categorization—assignment of one of a fixed set of names to a given pattern. The program described herein is a result of work on what we feel is a more difficult problem—producing a structural description of a pattern.

By a structural description, we mean an explanation of what the meaningful subpatterns or elements of the pattern are, how they are related to each other, and how they group together to form higher level meaningful subpatterns. This is the sort of thing that people do every day in describing a visual scene to someone else, but little is known about the processes they use, even in the simple case of a static line drawing.

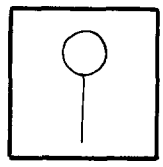
We feel that the ability to construct such descriptions is important to a pattern recognizer for two reasons. Firstly, an internal hierarchical representation of the structure of the pattern allows the same recognition process to operate iteratively on higher and higher levels of representation as recognition progresses, thus increasing the complexity of the recognition without increasing the complexity of the process which performs it. Secondly, output of a structural description (as opposed to a single name) allows meaningful feedback to be generated.

Our primary methodological viewpoint is that, when a problem solving program is to perform tasks of the order of difficulty of human perception, the reasonable thing to do is endow the program with the ability to "learn" the desired behavior from one who knows how to do it. The production of a "tentative" structural description of a pattern provides a trainer with material from which he can infer how the program arrived at its answer and what it needs to do better next time. Proper feedback then allows the program to modify itself and improve its performance.

The program we have developed is the outcome of attempts to program such an adaptive process in the area of pattern recognition and to explore ways of representing data and ways of organizing a growing memory. It is written in SNOBOL3, an interpretive string manipulation language, is quite long, and is by no means efficient. Nor has it been tested on a substantial number of patterns. Many crucial parts of the program are filled with stopgap measures.

But the program does give structural descriptions of patterns, can be trained in the above sense, develops a set of pattern characterizers that is hierarchically organized and can grow almost indefinitely, and in general exhibits several kinds of behavior which we are inclined to call "learning."

The following sections describe the data structures and operation of the program, and include examples of output descriptions and training sequences. Figure 1 shows a small-scale and slightly idealized sample of the kind of behavior the program exhibits:



Output: THERE IS A (SMALL SEMICIRCLE TO THE LEFT OF SMALL SEMICIRCLE) ABOVE A LONG STRAIGHT LINE
Feedback: CIRCLE ABOVE LONG STRAIGHT LINE
Output: THERE IS A CIRCLE (CONSISTING OF SMALL SEMICIRCLE TO THE LEFT OF SMALL SEMICIRCLE) ABOVE A LONG STRAIGHT LINE
Feedback: CIRCLE ABOVE STRING
Output: THERE IS A CIRCLE ABOVE A STRING
Feedback: BALLOON
Output: THERE IS A BALLOON (CONSISTING OF CIRCLE ABOVE STRING)

FIG. 1. Sample of descriptive and learning behavior of program.

It should be pointed out that the program is designed to handle patterns of much greater complexity than this example, including patterns with more than one object, i.e. with several non-contiguous groups of strokes.

2. DATA REPRESENTATION

Input

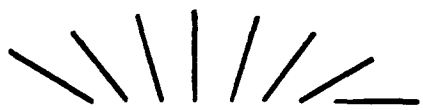
With the following limitations, the program attempts to handle any line drawing, that is, any pattern made up solely of thin lines, black on white. Input to the program is not the line drawing itself, but a coded abstraction of the drawing containing much less information than the original. The drawing is digitized in the usual manner, i.e. projected on a grid, 100×100, a cell in the grid being considered black if any line passes through it. Then the digitized image is segmented, at least at every line intersection, and possibly more often, in order to turn the picture into a network of straight lines and arcs. This is done by hand, but could be done by a curve-following device (e.g., KAZMIERCZAK;[5] SHERMAN[12]). Each of these "primitive strokes" is then coded as a triple of numbers, the numbers being measures of the slope, length, and curvature of the stroke. The precision of this coding can be varied, but there can be no *more* than nine values on each scale.

Next, the smallest possible rectangle is drawn about each stroke, and each stroke is given a serial number. Input to the program is a list of stroke descriptions, each description

containing serial number, grid coordinates of the circumscribing rectangle, slope–length–curvature code, and the numbers of all other strokes which it touches. A stroke description has an internal format which looks like:

$$4 = 04090906,230.T.3,5$$

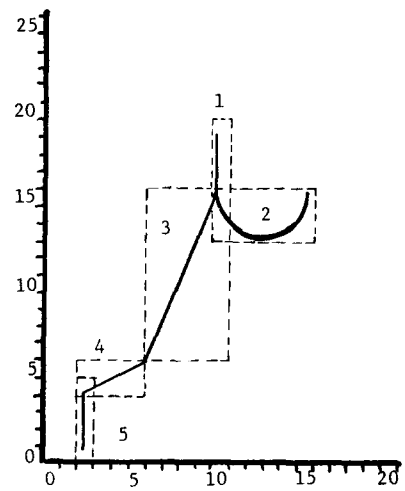
which says that stroke 4 lies within the rectangle whose left edge is at 04, right edge at 09, top edge at 09, and bottom edge at 06; that it has slope of 2, length 3, and curvature zero; and that it touches strokes 3 and 5. This coding is highly redundant, but the redundancy simplifies subsequent pattern matching.



Slope

actual	code
1	1
2-3	2
4-8	3
9-20	4
21-99	5

Length



B) SAMPLE LINE DRAWING, SHOWING BOUNDING RECTANGLES AND STROKE NUMBERS

Curvature

1=10101916,420.T.2,3
2=10151513,034.T.1,3
3=06101506,340.T.1,2,4
4=02050504,230.T.3,5
5=02020400,430.T.4

A) THE STROKE CODING SYSTEM C) CODED FORM OF THE DRAWING

FIG. 2. The scheme for encoding line drawings.

There is fairly good evidence (see, for example, HUBEL;[4] BARLOW,[2] and LETTVIN, MATURANA, MCCULLOCH and PITTS[6] that in animals, and presumably humans, the network of neurons in the retina, lateral geniculate, and the early stages of the visual cortex performs a comparable abstraction of an input pattern, i.e. reduction to lines and contours, and coding of attributes like position in the visual field, orientation, length, and curvature. This, we think, is support for the proposition that preprocessing of the type assumed by the program *does* occur in human pattern recognition.

The only assumptions made on the preprocessing (which hopefully could also be accomplished by a computer program) are that patterns be segmented in a consistent manner, with breaks at least at every line intersection, and that the number of "discernible" degrees of slope, length, and curvature is not more than nine per attribute—as suggested by the ability of human subjects to make absolute judgments as to these aspects of a pattern (MILLER[8]). Such coding thus defines the "smallest perceivable unit" for this program as an uninterrupted line segment, described to a certain precision as to slope, length, curvature, and position in the input pattern.

Relations

One of the problems in the design of an effective pattern recognition program is selection of the set of relations used to describe the ways the individual pattern elements are to be combined. This problem is one of those that our program was designed to investigate, and we have adhered to the philosophy that such a set is best acquired during real pattern recognition experience—with the program selecting relations which occur in the patterns themselves, and developing from these a set of useful ones.

The program works with an expanding set of binary spatial relationships between objects. For definiteness, let us refer to them as *S*-relations. The *S*-relation between two pattern elements is simply the relative position of the rectangles which surround the two objects, and is defined as the quadruple of signed numbers formed by subtracting corresponding coordinates of the two rectangles. To give a little more power to this relation format, tolerances are allowed on these relative differences—there is room in the relation format for both a minimum and a maximum value for each signed number. *S*-relations may be given names; internally they are referenced by serial numbers (e.g., R2, R37).

An *S*-relation is represented internally as in the following example:

$$3.3,0.9,-1.1,-40.-1,/*$$

If this relation were numbered R7, then if an object A is to be in R7 to B, (the left-most coordinate of A minus the left-most coordinate of B) can be at least 3 and at most 3, (the right-most coordinate of A minus the right-most coordinate of B) can be at least 0 and at most 9, and so on for top and bottom. If the relation has a name, the name replaces the asterisk.

This system seems to be general enough to express familiar spatial relations like "above", "to the left of," "enclosed by," "touching at the bottom," and so on, though the English names are often so ambiguous as to require a set of *S*-relations as referent. Moreover, the system can express relationships (like R7 above) which have no natural language correspondent, but which might prove useful in associating pattern elements on the micro level.

As an example of a "meaningful" relation, consider $0.99,-99.0,-99.-1,-99.-1$, which defines "under," in the sense that for A to be under B, it must lie completely within the left-right boundaries of B, its top must be below that of B, and its bottom must be below the bottom of B.

Certain types of spatial relations cannot be expressed in this format but it has proven useful, and general enough to allow the program to produce fairly natural pattern descriptions.

The set of *S*-relations which the program "knows about," and uses in its descriptions of the pattern, is drawn originally from the input patterns themselves, and is modified with experience. In this respect, the program is similar to those of UHR[14] and UHR and VOSSLER,[13] which also extract their operators from the input patterns. When a new pattern is given to the program, the *S*-relation between each pair of primitive strokes that touch is computed and added to the program's memory. Some of these *S*-relations prove useful in generating descriptions which satisfy the trainer, and are retained. Most of them, however, are not of general utility, and are erased before the next pattern is processed. Feedback from the trainer can cause broadening of tolerances on *S*-relations, assignment of a publicly understood name to them, or generation of *S*-relations between non-touching objects.

Thus there are continual additions to and deletions from the set of relations that the program uses in recognizing patterns, reflecting both the set of patterns seen to date, and the trainer's guidance.

Memory

The memory of the program consists of these relation definitions, and a set of "inferences" (so called because they are used to *infer* the existence of a familiar subpattern), which are template-like chunks of information that represent subpatterns which the program has seen before and has found useful. The format of an inference can be symbolized as:

$$\text{OBJECT 1} \quad S\text{-RELATION} \quad \text{OBJECT 2} \quad \xRightarrow{\text{WEIGHT}} \quad \text{NEWNAME}$$

The objects involved are either numerical codes for primitive strokes, or names learned through feedback for primitive strokes, or for a collection of them. NEWNAME is either a new name for OBJECT 1, or for a new whole formed by the pair of objects when they occur in the given relation. The first of these two renaming possibilities enables the program to attach a particular name to a part due to its context, and the second type enables it to combine two parts into a whole. The weight associated with the inference is meant to be a measure of the degree of confidence with which the program can use the inference. The weights are adjusted up and down (from an initial neutral value) by the program on the basis of its experience.

Varients of the scheme used to adjust weights have not been explored in this program. This process includes both what we sometimes called "conditioning," where the reweighting function may derive from very complex hypothesized mechanisms, and "apportionment of credit," where the problem solver that makes use of a large hierarchical network of transformations must decide how much credit (and therefore how much reweighting) to apportion to each part of the network, as a function of success or failure. The ability to do this may reasonably well be a crucial part of any adaptive problem solving program, and an appropriate set of rules can be quite complex (see, for example, AMAREL[1]).

The present weighting algorithm is very unsophisticated, sufficient only to allow the weights to change in the right directions: after each feedback session, each inference used correctly (to the trainer's satisfaction) in building the description is weighted up by one. Those which were incorrectly used are weighted down one. Those not used remain unchanged. The current range of weights is 1–9. If the weight becomes zero, the inference is erased from memory.

The inferences are represented in SNOBOL as strings of symbols, as in the following examples:

44-77-00.R1.66-77-00/P,9,ANGLE

Interpretation: If the two stroke types 470 and 670 (the doubling of numbers is a tolerance mechanism, explained shortly) occur in R1 in a pattern, they form, with certainty 9, an angle. 470 and 670 are both long straight lines, with different slopes. The "P" means that the inference does something to the Pair of objects.

CIRCLE.R42.HAT/F,5,FACE

Interpretation: If a circle occurs in R42 to a hat, then that circle (the First of the two objects), can henceforth be considered a face, with certainty 5.

Note that the inferences contain no absolute positional information. They can apply equally well anywhere on the pattern.

Original plans were to allow tolerances on primitive stroke descriptions in memory (that is, on the slope, length, and curvature numbers) so that an object in an inference could be a *class* of primitive strokes. This would be done by using two digits for each attribute—the first a minimum value, the second a maximum. Thus inferences could contain things like 07-11-00, which would match a stroke with any slope (0–7), length 1, and curvature 0, i.e. any very short, straight line.

Problems arose, however, in trying to develop an algorithm to modify these tolerances in accordance with feedback. Therefore, the data structures of the program (and those routines which operate on them) were written to allow tolerances in hope of eventual experimentation, even though only zero tolerance descriptions are currently used.

Recognition structure

As the program makes progress towards recognizing the picture, its developing concept of the pattern is stored as a labeled directed graph structure ("graph" being used here in the mathematical sense of a set of nodes connected by edges). The base is the network of primitive strokes making up the original line drawing, with edges representing the "touching" relation. Higher level nodes are constructed on this base by applying inferences to it, and those nodes in turn are used to construct still higher level nodes. This process results in a more or less pyramidal structure, with the highest node of the graph being the name of the pattern which said graph represents. Lower nodes are names of subordinate objects, while edges represent part-whole relationships, contiguity relationships, or spatial relationships. Internally, each node is represented by a list element containing pointers to sub- and super-parts, and a pointer to the inference which was used to create the node.

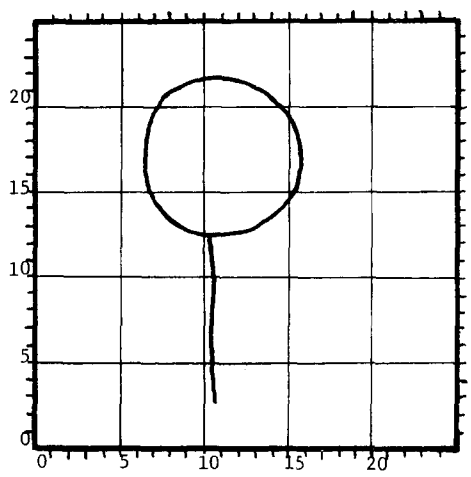
Such a structure is a detailed representation of a visual concept, in a form particularly well suited for transformation into a structural description of the pattern. In particular, the hierarchical nature of the structure makes it easy to vary the amount of detail in the description.

3. PROGRAM OPERATION

Having considered the data structures used by the program, let us turn to how these structures are built up and used.

After input of the pattern, program operation is an iterative process of "recognizing" the pattern, describing it as well as it can, processing feedback (i.e. learning), then back to recognition.

To motivate the detailed discussions that follow, we will begin with a short example, giving input to the program, and the program's response, for a training sequence leading to recognition of a drawing of a balloon. For simplicity, we assume the program begins with a blank memory.




```
1=10101202,440.T.2,3
2=06102213,441.T.1,3
3=11152213,444.T.1,2
```

ACTUAL DRAWING CODED INPUT

FIG. 3. Input for the balloon example.

Program's description: THERE IS A GROUP OF STROKES WITH THE FOLLOWING STRUCTURE
440/1/ IN R1 TO 441/2/
AND IN R2 TO 444/3/

441/2/ IN R3 TO 440/1/
AND IN R3 TO 444/3/

444/3/ IN R5 TO 440/1/
AND IN R6 TO 441/2/

(Note that this is simply an expansion of the input pattern graph, with the addition of labeled relations (whose definitions the program had now stored) between touching strokes—for the trainer's convenience in responding. Stroke serial numbers are set off by slashes.)

Feedback from trainer: (1) IN R1 TO (2) ARE CIRCLE
(3) IS STRING

(From this feedback, the program generates and adds to memory the following inferences:)

44-44-00.R1.44-44-11/P,4,CIRCLE
44-44-44.R5.44-44-00/F,4,STRING
44-44-44.R6.44-44-11/F,4,STRING

Program's description: ——THERE IS A——
CIRCLE/4/ CONSISTING OF 440/1/ IN R1 TO 441/2/
THERE IS ALSO A GROUP OF STROKES WITH THE FOLLOWING STRUCTURE
STRING/3/ IN R5 TO 440/1/
AND IN R6 TO 441/2/

(Note that a new object (CIRCLE) has been found, and that the previously unnamed 444 is now called STRING.)

Feedback from trainer: (4) AND (3) ARE BALLOON

(This feedback causes the program to generate the *S*-relation R7 between CIRCLE and STRING, and the following inference:)

CIRCLE.R7.STRING/P,4,BALLOON

Program's description: ——THERE IS A
BALLOON/5/ CONSISTING OF CIRCLE (CONSISTING OF 440/1/
IN R1 TO 441/2/) IN R7 TO STRING

Feedback from trainer: (4) ABOVE (3)
$MAXDEPTH = "2"

(Here he causes the name ABOVE to be associated with R7, and limits the "depth" of subsequent descriptions to two.)

Program's description: ——THERE IS A——
BALLOON/5/ CONSISTING OF CIRCLE/4/ ABOVE
STRING/3/

Recognition

In the recognition phase, the program considers one object at a time, checking its memory for elements which have been made out of that type of object (the type of an object is its name, e.g., ANGLE, BALLOON, 441) in the past. For each such potential new element, the pattern is searched for the requisite context or related object, and then checked to see if that object is in the requisite *S*-relation.

The problem of finding—in a possibly very large memory—just those inferences which might apply to an object (that is, those in which that object type occurs as a component) is simplified by accessing memory associatively. The program contains a dictionary which associates each object type with a list of pointers to those inferences which might apply to it. Given an object in the pattern, the program simply tries out inferences named on this pointer list, and does not have to search the entire set of inferences.

The recognition process starts with the primitive stroke which has the highest number of connections with other strokes (this hopefully is an information-rich part of the pattern). If no applicable inferences can be found, it simply moves on to the stroke with the next higher serial number. If an inference *is* found that matches the pattern, a new node in the recognition structure is created—with the name given in the NEWNAME part of the inference, and with a bounding rectangle which surrounds both subparts. For a renaming type inference, the new name is simply added to the old node as an alias.

When an inference matches, the newly created node is made the current object of the memory search. Thus if the program puts two semi-circles together to make a circle, it will begin trying out things it can make out of a circle. The program tends to keep working in an area in which it is having success.

This is a departure from a large number of conventional pattern recognition schemes—those which have a rigid recognition strategy which is independent of the pattern. They measure a certain fixed number of things about each pattern regardless of the structure of the particular pattern under observation. This program moves in the direction of having the pattern itself, in conjunction with memory of previous patterns, direct the recognition process.

Once a pattern element has been combined with something else to form a higher level element, it remains available for additional combination with other elements, and for use as a context object. This means that as the gross organization of the pattern is completed, the more detailed organization is still there for reference. Other hypotheses about what the pattern is can build on part of the organization contributed by the first hypothesis, without having to start over again at the primitive stroke level. In theory, when the program "gets off the track" on a particular section of the pattern, it could detect the step at which it was mislead, and go back to that step for a new approach. In practice, nothing this sophisticated is attempted. Whenever a sub-pattern looks to the program as if it might be more than one thing, *all* possibilities are pursued. The program may make simultaneous alternate hypotheses about the input.

The routine to make the final choice among alternate hypotheses is not yet written, and may be quite difficult. There could be many kinds of overlap and conflict in a multi-hypothesis structure: two hypotheses about the same area of the pattern, two about slightly overlapping areas, a hypothesis based on a very weak inference, etc. The general plan is to resolve conflicts using the weights of the inferences used to construct them, and the height of the structures which they represent. This may not be enough information to do a realistic job of decision making, since essentially it consists only of relative frequency of occurrence of the pattern elements.

One additional mechanism, which was found necessary to avoid repeated application of the same inference to the same component of the pattern, should be explained. As the recognition process proceeds, the program appends to each object description (each node) a list of inference numbers—those which possibly apply, but have been found not to (due to lack of the requisite second object in the pattern, or to the two objects not falling in the right relation). Thus whenever the program looks at an object, it knows not to try to apply a certain set of inferences. Included on this "don't try" list are numbers of those inferences which have been *successfully* applied to the object. These lists effectively block the program, as it goes over and over the pattern, from redoing work it has already done.

The lists of inferences not to try must be modified as new objects are recognized—since the new object may be the one that the program had earlier looked for and not found. This necessitates some complex housekeeping, but probably makes the program more efficient in the long run. One of the most time-consuming processes in the program is the graph-matching procedure used to check for match of an inference (which can be thought of as a two-node, one-edge graph) against the larger graph of the pattern. The pattern must be serially searched for all occurrences of the requisite context or second object, and a relation check must be made for each occurrence. Thus anything which avoids repetition of graph matching is a great help.

The program continues to cycle through its list of "found objects" (i.e. nodes in the recognition structure) until a complete pass has been made without finding any inferences which apply to any object. This generally takes many cycles, since the normal course of events is for an object found on one cycle to be combined with or used as context for

another object on the next cycle. To put it another way, the program reconsiders each old object at each new addition.

Description

Two examples of the program's output are given in Fig. 4. It is a parenthetically nested, structural description of the pattern, which can be varied in depth and detail. Areas of the pattern which were not recognized are restated more or less in input form, and linkages into those parts of the pattern which *were* recognized are given. The intent is to give the trainer material to work from in composing his feedback, and the descriptions are far from polished English.

```
——THERE IS A——
(ANGLE/10/ CONSISTING OF 620/6/ IN R12 TO 220/7/)——AND ALSO——
(LANGLE/9/ CONSISTING OF 020/2/ IN R4 TO 430/3/)——AND ALSO——
(LANGLE/8/ CONSISTING OF 430/1/ IN R2 TO 020/4/)
THERE IS ALSO A GROUP OF PRIMITIVE OR RENAMED PRIMITIVE STROKES WITH THE FOL-
LOWING STRUCTURE
BASE (5) IN R9 TO 620/6/
   AND IN R10 TO 220/7/

——THERE IS A——
(STICKBOY/23/ CONSISTING OF (UPPERBODY/20/ CONSISTING OF (ARMS/18/ CONSISTING OF
LEFTARM/13/ IN R32 TO RIGHTARM/15/) IN R34 TO (HEAD/14/CONSISTING OF SEMICIRCLE/1/
TOTHELEFTOF SEMICIRCLE/2/)) IN R35 TO (LOWERBODY/22/ CONSISTING OF (BODY/19/ CON-
SISTING OF NECK/11/ IN R22 TO TRUNK/12/) IN R33 TO (LEGS/21/ CONSISTING OF LEFTLEG/16/
IN R31 TO RIGHTLEG/17/)))
```

FIG. 4. Samples of the program's description format. The upper example is from an intermediate stage in learning to describe a square and triangle pattern, while the lower is the final description of a simple stick-figure.

Feedback and learning

Feedback is given as a series of simple, fixed format statements, each of which mentions one or two of the objects mentioned by the program in its description. The program, with reference, if necessary, to its internal concept of the pattern (what we have been referring to as the recognition structure), analyzes the feedback and makes changes to memory. These changes consist of addition of new inferences to memory, weighting down of old ones, or naming of relations.

To simplify analysis of the feedback, objects must be referred to by serial number rather than by name. This makes the feedback rather artificial looking, but it does eliminate the possibly complex problem of determining what the trainer meant when he used an object name which occurred more than once in the description. Feedback statements must be in one of the following fixed formats:

(1) (*n*) IN R*k* TO (*m*) IS newname
(2) (*n*) IN R*k* TO (*m*) ARE newname
(3) (*n*) WITH (*m*) IS newname
(4) (*n*) AND (*m*) ARE newname
(5) (*n*) IS newname
(6) (*n*) relationname (*m*)
(7) NO, NOT (*n*)

where *n* and *m* are object serial numbers, *k* is a relation serial number, and relationname and newname are any sequence of non-blank alphanumeric characters.

In Formats 1 and 2, the relation to be used in the inference is named explicitly, and must already be in the program's memory. This format is used primarily in responding to descriptions consisting of contiguous primitive strokes and primitive relations.

In processing Formats 3 and 4, the (zero-tolerance) *S*-relations between the two given objects is generated, added to memory, and then used in the inference. This allows inclusion of relations between nontouching objects.

Format 5 is an experimental one, meant to be a convenience in getting the program to rename a primitive stroke due to context. It causes generation of many F-type inferences as there are objects touching the given one: i.e. the program will henceforth rename the stroke due to any *one* of its possibly many contexts (using "context" here in the narrow sense of "relation to a contiguous stroke").

Format 6 is used for naming relations. The program finds the number of the relation between n and m, and assigns it the given name. In the event that the program already has in memory a relation with this name, it will broaden the tolerance on that relation to make it include the new instance. The program is thus able to broaden its concepts of spatial relations. It *cannot*, however, narrow them, or build disjunctive relational concepts.

Format 7 provides a means of telling the program "no" on any particular object, i.e. that the inference used to create that object was wrongly used, and should be weighted down. This is the only mechanism currently available for "extinguishing" the program's responses.

This set of formats is sufficient for at least crude training of the program, but is still somewhat unnatural and restrictive. Two more formats, as yet unimplemented, have been suggested to help alleviate the problem. One would be a way to tell the program that an object can be renamed *regardless* of context (e.g. that any stroke with curvature zero can be considered a straight line).

Another would tell the program "Yes, your name for object n is *correct*, but a better name in this instance is ———." This would provide a way of correcting the program without weighting down any inferences.

The trainer is not allowed to "skip a level" in giving his feedback. That is, he cannot give a name to a collection of more than two objects. Confronted with the initial description of the earlier example, he cannot simply say "This is a balloon," and let the program figure out how to organize all those pattern components into something which will be useful in later recognition of balloons. The trainer must work "from the bottom up" in training the program.

In one sense, this is not too unreasonable psychologically. Even humans get confused when feedback differs extremely from the way they see it. Prohibition of level skipping acts as a kind of "confusion filter," which serves the valuable purpose of preventing highly speculative learning.

4. DISCUSSION

The work reported herein can be viewed as relevant to numerous problem areas in pattern recognition research and artificial intelligence, including pattern description, use of a growing system of pattern inter-relationships, utilization of feedback, abstraction and storage of graphic information, organization of pattern characterizers, and learning.

In view of the reasonable similarity of the program's input, and some of its processing to certain stages of the perceptual process in living beings, it might be of some value as a

conceptual tool in thinking about visual perception—a very limited kind of visual perception. It would be premature, however, to call it a good model of the perceptual process.

The program is similar in some respects to the NAMER program of LONDE and SIMMONS.[7] Both recognize and generate sentences about line drawings, and both abstract concepts which correspond to spatial relationships. There are two major differences. NAMER does not produce hierarchical descriptions, being limited to sentences telling the spatial relationship of a single pair of objects. Also, NAMER incorporates a non-iterative, categorization type of pattern recognition, in which a fixed set of 96 characterizers is computed on each of the two input patterns, and a weighted correlation to stored sets of values of these characteristics produces the most probable name for the pattern. In our program, the set of characterizers grows with experience, and the part-whole structure of the pattern guides the recognition process: each new part "recognized" determines a new set of tests to be made on the pattern.

There are many areas for further research that we would like to pursue. One of these is to work more on the problem of interfacing programs of this type with the problem of comprehending and using natural language. Clearly the pattern description format could stand improvements, and the feedback language could be enriched in many ways. It is interesting to note, in this respect, that LONDE and SIMMONS[7] have had some success with using a simple generative grammar to generate sentences about line drawings. The work of UHR[14] is also relevant, and QUILLIAN[11] is currently working on a "teachable" language comprehender.

Another area needing investigation is machine realization of the type of pattern preprocessing we have assumed for this program. There may well be significant problems in decomposing an arbitrary line drawing into the network of non-overlapping strokes that the program requires.

In our experience with the program to date, the particular inference and relation representation used has been quite natural, and not at all restrictive. The attributes of slope, length, and curvature, rectangle occupied by the stroke, and the touching relation seem to be both sufficient for effective pattern recognition and natural for describing patterns. One area which needs further exploration, however, is the difficulties that might arise in training the program on a substantial environment of complex drawings.

One possible limitation is the fact that characterizers in memory are limited to only two objects, linked by one *S*-relation. It seems plausible that some pattern characterizers would not fall naturally into this format, either because they involve more than two objects, or because the crucial relating function cannot be expressed in the *S*-relation format. As an example of the latter difficulty, consider the intuitive notion of an angle: two straight lines (of any length) meeting at a point and having different slopes. The qualities of zero curvature and arbitrary line length can be represented in an object type, but not the simple fact that the strokes can have any slope at all as long as the two are different. This quality would currently have to be represented by a large set of inferences, one for each possible combination of different slope numbers. Thus a graceful generalization of the intuitive notion of angle could not be achieved without using an inordinate amount of memory.

We believe, however, that a useful class of pattern types can be generalized without resorting to such a procedure. In the examples given, there has been only one inference leading to a given object type. In the general case, there will be more than one, and in fact the ability to represent a concept as the disjunction of several different lower level patterns is very valuable. It should be noted also that conjunction of lower level patterns can be

handled, though not elegantly, by introducing an intermediate artificial object type:

OBJECT1 R1 OBJECT2 IS OBJECT#
OBJECT# R2 OBJECT3 IS OBJECT4

so that a subpattern is of type OBJECT4 if it is both in R1 to an OBJECT2 *and* in R2 to an OBJECT3.

It would certainly be desirable to improve the relation system. The program currently has no means of narrowing, or making more precise, the definition of a relation, and cannot learn or use *properties* of relations—as does, for instance, the fact retrieval program of ELLIOT.[3]

It should also be noted that, though some of the most impressive features of the program are related to learning, learning can occur only on the level of data acquisition and erasure. Much more subtle and powerful kinds of learning are conceivable, among them reorganization and reassociation of data, and adaptation of the learning strategy itself, i.e. learning to learn. Little work has yet been done on endowing programs with these abilities.

In conclusion, many writers (e.g. NARASIMHAN,[10] MINSKY[9] have discussed the importance, in the processing of visual data, of the ability to *describe* patterns, in addition to categorizing them. This program has demonstrated ability to produce structural descriptions of patterns of moderate complexity, to learn a set of names and relations from which to build descriptions, to vary the detail of those descriptions, and to do an effective job of abstracting and storing "chunks" of visual information, and using them in subsequent pattern recognition and description problems.

SUMMARY

This paper discusses the structure and operation of a computer program, written in SNOBOL3, which learns to recognize and describe two-dimensional line drawings.

Input to the program is a coded version of the drawing, in which the drawing is decomposed into elementary "strokes", each described by its slope, length, and curvature. The program attempts to find, in the coded input pattern, previously learned subpatterns of the form stroke–relation–stroke. The recognition procedure is dependent on both the prior experience of the program, and the structure of the pattern currently being examined. The relations used are essentially classes of relative positions of two strokes, or two groups of strokes.

A structural description of the pattern is produced, using learned names for familiar subpatterns and familiar spatial relations, and elementary stroke codes in unfamiliar parts of the pattern. A human trainer responds, offering corrections, and naming new subpatterns. By analyzing this feedback, the program adds to and revises its memory of significant subpatterns and useful spatial relationships, and then makes another attempt at describing the pattern.

Included in the paper are detailed descriptions of the input coding system, the internal representation of subpatterns and relations among subpatterns, and of the feedback procedures. Samples of actual output are shown, and a short "training sequence," showing effects of feedback on the program's memory, is presented.

The paper points out some similarities between the program's behavior and the perceptual process in humans, and concludes with a short discussion of possible extensions and relationships to other areas of pattern recognition and artificial intelligence.

Acknowledgement—This research was partially supported by NIH grants MH 05254 and MH 12266, and by NSF grant GP-7069.

REFERENCES

1. S. AMAREL, On the Automatic Formulation of a Computer Program which Represents a Theory, in *Self-Organizing Systems* 1962 (Edited by YOVITS), Spartan Books (1962).
2. H. B. BARLOW, *J. Physiol.* **119**, 69 (1953).
3. R. ELLIOT, A Model for a Fact Retrieval System, TNN-42. University of Texas Computation Center, May, 1965.
4. D. H. HUBEL, *Scient. Am.* **209**(5), 54 (1963).
5. H. KAZMIERCZAK, The Potential Field as an Aid to Character Recognition. *Information Processing*, p. 244. Paris: UNESCO (1960).
6. J. Y. LETTVIN, H. R. MATURANA, W. S. MCCULLOCH and W. H. PITTS. *Proc. IRE* **47**, 1940 (1959).
7. D. L. LONDE and R. F. SIMMONS, NAMER: A Pattern Recognition System for Generating Sentences about Line Drawings, TM 1798. Systems Development Corporation, Santa Monica, California, 1964.
8. G. A. MILLER, *Psych. Rev.* **63**, 81 (1956).
9. M. MINSKY, Steps toward artificial intelligence, *Proc. IRE*, **49**(1) 8 (1961).
10. R. NARASIMHAN, Syntax directed interpretation of classes of pictures, *Communications of the ACM*, **9**, 166, March, 1966.
11. R. QUILLIAN, The Teachable Language Comprehender, to appear in *Communications of the ACM*. In press.
12. H. SHERMAN, A Quasi-topological Method for the Recognition of Line Patterns. *Information Processing*, p. 232. Paris: UNESCO (1960).
13. L. UHR and C. VOSSLER, *Proc. WJCC*, **19**, 555 (1961).
14. L. UHR, *Behav. Sci.*, **9**(2), 258 (1964).