

A HEURISTIC PROGRAM TO SOLVE GEOMETRIC-ANALOGY PROBLEMS

Thomas G. Evans

*Air Force Cambridge Research Laboratories (OAR)
Bedford, Massachusetts*

INTRODUCTION

The purpose of this paper is to describe a program now in existence which is capable of solving a wide class of the so-called 'geometric-analogy' problems frequently encountered on intelligence tests. Each member of this class of problems consists of a set of labeled line drawings. The task to be performed can be concisely described by the question: 'figure A is to figure B as figure C is to which of the given answer figures?' For example, given the problem illus-

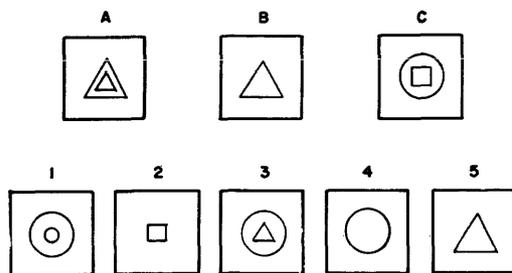


Figure 1.

trated as Fig. 1, the geometric-analogy program (which we shall subsequently call ANALOGY, for brevity) selected the problem figure labeled 4 as its answer. It seems safe to say that most people would agree with ANALOGY's answer to this problem (which, incidentally, is taken from the 1942 edition of the *Psychological Test*

for College Freshmen of the American Council on Education). Furthermore, if one were required to make explicit the reasoning by which he arrived at his answer, prospects are good that the results would correspond closely to the description of its 'reasoning' produced by

ANALOGY.

At this point, a large number of questions might reasonably be asked by the reader. Four, in particular, are:

- (i) Why were problems of this type chosen as subject matter?
- (ii) How does ANALOGY go about solving these problems?
- (iii) How competent is ANALOGY at its subject matter, especially in comparison to human performance?
- (iv) What has been learned in the construction of ANALOGY and what implications might this study have for the further development of problem-solving programs in general?

The remainder of this paper constitutes an attempt to answer these questions in some detail. We first deal with a variety of motivations for this investigation and attempt to place it in the context of other work in related areas. Next we turn to detailed consideration of the problem type and of the mechanism of the ANALOGY program. Finally, we present some answers to

the remaining two questions raised above. (A more detailed discussion of all these issues can be found in Ref. 1).

Motivations and Background

In our opinion ample general justification for the development and study of large heuristic problem-solving programs has been provided (both through argument and through example) by previous workers in this area. We shall not attempt to add to it. Given that one is interested in the construction of such programs, a number of reasons can be advanced for the choice of geometric-analogy problems as a suitable subject matter. Some of these are:

(i) Problems of this type require elaborate processing of complex line drawings: in particular, they require an analysis of each picture into parts and the determination and use of various relationships among these parts. This is an interesting problem *per se* and one which can reasonably be expected to be of great practical importance in the near future.

(ii) The form of the problems requires one to find a transformation that takes figure A into figure B and takes figure C into exactly one of the answer figures. This situation provides a natural opportunity for trying out certain ideas about the use of explicit internal 'descriptions' (here, of both figures and transformations) in a problem-solving program. Furthermore, more speculatively, it presents an interesting paradigm of 'reasoning by analogy,' a capacity which may play a large role in far more sophisticated problem-solving programs in the future. (In Section 5 we discuss the possible relevance of ANALOGY to the introduction into problem-solving programs of more powerful learning mechanisms than have yet been achieved.)

(iii) Problems of this type are widely regarded as requiring a considerable degree of intelligence for their solution and in fact are used as a touchstone of intelligence in various general intelligence tests used for college admission and other purposes. This suggests a non-trivial aspect of any attempt to mechanize their solution.

We shall now attempt very briefly to place ANALOGY in the context of earlier work in re-

lated areas. Two aspects of ANALOGY must be considered:

(i) ANALOGY contains a substantial amount of machinery for the processing of representations of line drawings, including decomposition into subfigures, calculation of relations between figures, and 'pattern-matching' computations. Thus we must relate it to other work in picture processing and pattern recognition.

(ii) ANALOGY is a complex heuristic problem-solving program, containing an elaborate mechanism for finding and 'generalizing' transformation rules. Thus we must relate it to other work on the development of problem-solving programs.

We turn first to the picture-processing aspect. The essential feature of the treatment of line drawings by ANALOGY is the construction, from relatively primitive input descriptions, of more 'abstract' descriptions of the problem figures in a form suitable for input to the rule-finding program. The fundamental programming technique underlying this method is the use of a list-processing language, in this case LISP,^{2,3} to represent and process the figures in question. Work in picture processing, for pattern-recognition purposes, involving some elements of description, is found in Grimsdale *et al.*,⁴ Marill *et al.*,⁵ and Sherman,⁶ among others. Sutherland⁷ and Roberts⁸ have used, for quite different purposes, internal representations of line drawings similar in some respects to those used in ANALOGY. Kirsch⁹ has worked with complex line drawings primarily as a vehicle for programs involving the analysis of English-language sentences pertaining to such pictures. Hodes¹⁰ and Canaday¹¹ have used LISP expressions for figure description in much the same way that we have, though the development of machinery for manipulating such descriptions was, of necessity, carried much further in ANALOGY. Evidently the first advocacy of 'scene description' ideas (for use in pattern recognition) occurs in Minsky.¹²

To place ANALOGY with respect to other work with problem-solving programs, we shall simply list a number of developments in the construction of problem-solving programs which have influenced, in a general way, our approach to the design of ANALOGY. These include LT

(the Logic Theorist)¹³ and, more recently, GPS (the General Problem Solver)¹⁴ of Newell, Simon, and Shaw, the plane-geometry theorem-prover¹⁵ of Gelernter and Rochester, and SAINT, the formal integration program of Slagle.¹⁶

Summary of the Solution Process, with Example

To exhibit as clearly as possible the entire process carried out by ANALOGY, we now sketch this process, then examine its operation on an example. The sample problem we shall be considering is shown as Fig. 2 (where the P_i's are not part of the problem figures but labels keying the corresponding parts of the figures to

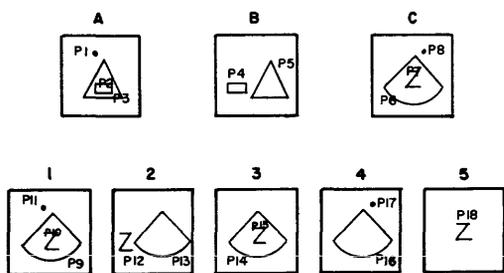


Figure 2.

expressions we shall give below). Before treating the example, we shall summarize the entire solution process. Given a problem such as that above, ANALOGY proceeds as follows: First, the input descriptions of the figures are read. Currently these descriptions, given as LISP expressions in a format to be illustrated below, are hand-made; however, they could well be mechanically generated from scanner or light-pen input by a relatively straightforward, quite 'unintelligent' program embodying line-tracing techniques already described in the literature. The descriptions represent the figures in terms of straight line segments and arcs of circles (to any desired accuracy, at the cost of longer and longer expressions). Examples of the descriptions are given below.

The first step taken by ANALOGY is to decompose each problem figure into 'objects' (sub-figures). The decomposition program originally written, which was sufficient to handle many

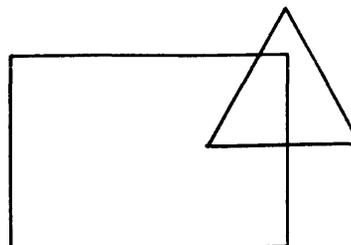


Figure 3a.

cases, including the example to be discussed below, was quite simple. It merely separated a problem figure into its connected subfigures; e.g., figure A of the above example consists of the three objects labeled P₁, P₂, and P₃. It later became desirable to have a more sophisti-

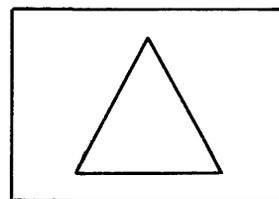


Figure 3b.

cated decomposition program with, in particular, the capability of separating overlapped objects on appropriate cues. For example, suppose problem figure A is as in Fig. 3a and figure B is as in Fig. 3b. The decomposition program should be able to separate the single object of figure A into the triangle and rectangle on the basis that they appear in figure B, from which point the remaining mechanism of parts I and II could proceed with the problem. While a decomposition program of the full generality desirable has not yet been constructed, the most recent version of the program is capable, in particular, of finding all occurrences of an arbitrary simple closed figure x in an arbitrary connected figure y; for each such occurrence the program can, if required, separate y into two objects: that occurrence of x and the rest of y (described in the standard figure format—note that this 'editing' can be rather complex: connected figures can be split into non-connected parts, etc.).

The type of decomposition illustrated above might be called 'environmental,' in that, e.g., figure A is separated into subfigures on the information that these subfigures are present, already separated, in figure B. An interesting extension to the present part I of ANALOGY might be to incorporate some form of 'intrinsic'

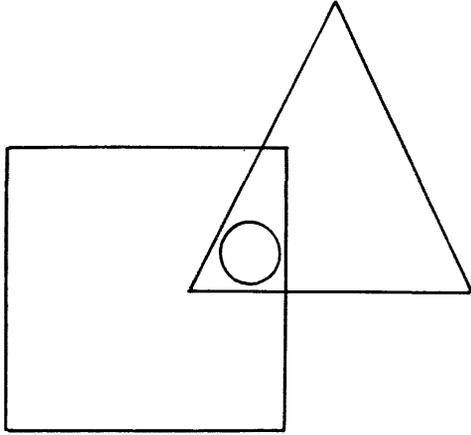


Figure 4a.

decomposition in which 'most plausible' decompositions are generated according to Gestalt-like criteria of 'good figure.' Such an extension could widen the problem-solving scope of ANALOGY considerably to include many cases where the appropriate subfigures do not appear already 'decomposed' among the problem figures. For example, suppose problem figures A and B are as shown in Figs. 4a and 4b, respec-

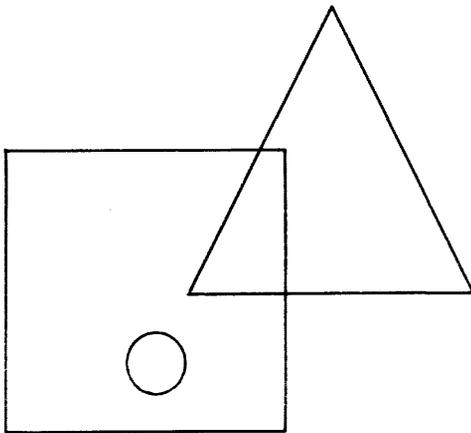


Figure 4b.

tively. A decomposition into the square, triangle, and circle seems necessary to state a reasonable transformation rule. This example, incidentally, illustrates one potentially useful 'intrinsic' decomposition heuristic: roughly, choose decompositions into subfigures which have as much internal symmetry (in some precise sense) as possible.

Next, the 'objects' generated from the decomposition process are given to a routine which calculates a specified set of properties of these objects and relations among them. The program is designed so that this set can be changed easily. As a sample of a relation-calculating subroutine, we cite one that calculates, for figure A of our example, that the object labeled P2 lies inside that labeled P3 and generates a corresponding expression (INSIDE P2 P3) to be added to the part I output description of figure A. The method used involves calculating all intersections with P3 of a line segment drawn from a point on P2 to the edge of the field (all figures are considered as drawn on a unit square). In this case P2 lies inside P3 since the number of such intersections is odd, namely one (and P3 is known to be a simple closed curve—if it were not, the calculation just described would be performed for each closed curve contained in P3). To do this, a substantial repertoire of 'analytic geometry' routines is required for part I, to determine, for example, intersections of straight line segments and arcs of circles in all cases and combinations. Other relation routines available in part I calculate, for example, that in figure A of our example P1 is above P2 and P3 and in figure B that P4 is to the left of P5.

The principal business of part I, aside from decomposition and the property and relation calculations, is a set of 'similarity' calculations. Here, part I determines, for each appropriate pair of objects, all members from a certain class T of transformations which carry one object of the pair into the other. The elements of T are compositions of Euclidean similarity transformations (rotation and uniform scale change) with horizontal and vertical reflections. Given descriptions of virtually any pair of arbitrary line-drawings x and y, the routines of part I will calculate the parameters of all instances of transformations from T that 'map' x into y.

More precisely, an acceptable 'map' is a member of T for which $T(x)$ is congruent to y up to certain metric tolerances which are parameters in the corresponding programs.

This routine is, in effect, a pattern-recognition program with built-in invariance under scale changes, rotations, and certain types of reflections. It consists essentially of a topological matching process, with metric comparisons being made between pairs of lines selected by the topological process. In Ref. 6 Sherman introduced some topological classification into a sequential decision tree program for the recognition of hand-printed letters, but the notion of systematically using the topological information to determine which metric comparisons are to be made seems to be new. This type of organization for pattern recognition has its own advantages (e.g., flexibility—the metric parts can be changed easily with no effect on the overall structure) and difficulties (e.g., sensitivity to metrically small changes in a figure which affect the connectivity—but this sensitivity can be largely removed by suitable pre-processing). Incidentally, it may be worth noting that if we suppress the metric comparisons entirely we get a general, and reasonably efficient, topological equivalence algorithm for graphs (networks).

The set of techniques we have just been describing, based on the use of a list-processing language to perform processing of line drawings by manipulating their list-structured descriptions, is by no means limited in applicability to the uses to which we have put it in part I of ANALOGY. To the contrary, it is our view that the representation of line drawings used here and the corresponding processing routines form a suitable basis for the development of a quite powerful 'line-drawing-manipulation language' with potential usefulness in a wide variety of applications. Regardless of whether the present investigation turns out to have a measurable influence on the art of designing problem-solving programs, it seems probable that the principal short-range contribution of ANALOGY is in the picture-processing by-products just described. (Incidentally, these techniques were discussed briefly from an ANALOGY-independent point of view in Ref. 17.)

After the similarity information is computed for every required pair of objects, both within a problem figure and between figures, this information, together with the decomposition and property and relation information, is punched out on cards in a standard format for input to part II. (For a typical set of figures, the total output of part I, punched at up to 72 columns/card, might come to 15 to 20 cards.)

Part II is given these cards as input. Its final output is either the number of the solution figure or a statement that it failed to find an answer. Its first step is to generate a rule (or, more frequently, a number of alternate rules) transforming figure A into figure B. Such a rule specifies how the objects of figure A are removed, added to, or altered in their properties and their relations to other objects to generate figure B. Once this set of rule possibilities has been generated, the next task is to 'generalize' each rule just enough so that the resulting rules still take figure A into figure B and now take figure C into exactly one of the answer figures. More precisely, for each 'figure A \rightarrow figure B' rule and for each answer figure, part II attempts to construct a 'common generalization' rule which both takes figure A into figure B and figure C into the answer figure in question. This process may produce a number of rules, some very weak in that virtually all the distinguishing detail has been 'washed out' by 'generalization.' Hence it is necessary at this point to pick the 'strongest' rule by some means. This entire process requires a complex mechanism for manipulating and testing the rules and deciding which of the several rule candidates, the results of different initial rules or of different 'generalizations,' is to be chosen.

The principal method embodied in part II at present is able to deal quite generally with problems in which the numbers of parts added, removed, and matched in taking figure A into figure B are the same as the numbers of parts added, removed, and matched, respectively, in taking figure C into the answer figure. A substantial majority of the questions on the tests we have used are of this type, as is our present example; virtually all would be under a sufficiently elaborate decomposition process in part I; this restriction still permits a wide variety of transformation rules. It should be mentioned

that the methods of part II have been kept subject-matter free; no use is made of any geometric meaning of the properties and relations appearing in the input to part II.

The more detailed workings of both parts I and II are best introduced through examining the process sketched above at work on our example. To convey some further feeling for the nature of the input to part I, we exhibit part of it, namely, the input description of figure A. The LISP expressions look like:

```
(
(DOT (0.4 . 0.8))
(SCC ((0.3 . 0.2) 0.0 (0.7 . 0.2) 0.0 (0.5 .
0.7) 0.0 (0.3 . 0.2)))
(SCC ((0.4 . 0.3) 0.0 (0.6 . 0.3) 0.0 (0.6 .
0.4) 0.0 (0.4 . 0.4) 0.0 (0.4 . 0.3)))
)
```

The first line above corresponds to the dot (at coordinates $x = 0.4$ and $y = 0.8$ on the unit square; the coordinate pairs in the other expressions are interpreted analogously). The next two lines correspond to the triangle (SCC stands for simple closed curve. All connected figures are divided into three classes—dots (DOT), simple closed curves (SCC), and all the rest (REG). This is solely for reasons of programming convenience; no other use is made of this three-way classification). Each non-connected figure is represented simply by a list of descriptions of its connected parts.

A curve (which may consist of an arbitrary sequence of elements chosen from straight line segments and arcs of circles) is represented by a list in which coordinate pairs alternate with the curvatures of the line elements between (all curvatures are zero here since the lines in question are all straight). Similarly, the next two lines above correspond to the rectangle; the entire description of figure A is then a list of the descriptions of these three parts. The format corresponding to the non-SCC figures like the Z-shaped subfigure of figure C is similar though somewhat more complex; it looks like:

```
(REG ((VI V2 (0.0 (0.55 . 0.5) 0.0 (0.45 .
0.3) 0.0))
(V2 V1 (0.0 (0.45 . 0.3) 0.0 (0.55 . 0.5)
0.0))))
```

where V1 and V2 are the two vertices (here, endpoints) of the figure. The coordinates of V1 and V2 are given to part I in a separate list. They are $V1 = (0.45 . 0.5)$, $V2 = (0.55 . 0.3)$. Here, the top-level list describes the connectivity by stating which vertices are connected to which and how often—sublists describe in detail the curves making these connections. (By vertex we mean either an endpoint of a curve or a point at which three or more curves come together.) The complete details of the input format are given in Ref. 1, along with many examples.

When the input shown above corresponding to problem figure A and the corresponding inputs for the other seven figures are processed, the output from part I is, in its entirety, the ten LISP expressions shown below. For brevity, all similarity information concerning non-null reflections has been deleted. Also, we have replaced the actual arbitrary symbols generated internally by ANALOGY as names for the parts found by the decomposition program by the names P1, P2, etc., which appear as labels on our example figures above. The ten output expressions are:

- (1) ((P1 P2 P3) . ((INSIDE P2 P3) (ABOVE P1 P3) (ABOVE P1 P2)))
- (2) ((P4 P5) . ((LEFT P4 P5)))
- (3) ((P6 P7 P8) . ((INSIDE P7 P6) (ABOVE P8 P6) (ABOVE P8 P7)))
- (4) ((P2 P4 (((1.0 . 0.0) . (N.N)) ((1.0 . 3.14) . (N.N)))) (P3 P5 (((1.0 . 0.0) . (N.N)))))
- (5) ((P1 P8 (((1.0 . 0.0) . (N.N)))))
- (6) NIL
- (7) ((P9 P10 P11) (P12 P13) (P14 P15) (P16 P17) (P18))
- (8) (((INSIDE P10 P9) ABOVE P11 P9) (ABOVE P11 P10) ((LEFT P12 P13)) ((INSIDE P15 P14)) ((ABOVE P17 P16)) NIL)
- (9) (((P6 P9 (((1.0 . 0.0) . (N.N)))) (P7 P10 (((1.0 . 0.0) . (N.N)) ((1.0 . -3.14) . (N.N)))) (P8 P11 (((1.0 . 0.0) . (N.N)))))

```

((P6 P13 (((1.0 . 0.0) . (N.N))))
 (P7 P12 (((1.0 . 0.0) . (N.N))
 ((1.0 . -3.14) . (N.N))))))
((P6 P14 (((1.0 . 0.0) . (N.N))))
 (P7 P15 (((1.0 . 0.0) . (N.N))
 ((1.0 . -3.14) . (N.N))))))
((P6 P16 (((1.0 . 0.0) . (N.N))))
 (P8 P17 (((1.0 . 0.0) . (N.N))))))
((P7 P18 (((1.0 . 0.0) . (N.N)) ((1.0 .
 -3.14) . (N.N))))))
(10) ( ( ((P1 P11 (((1.0 . 0.0) . (N.N))))))
      NIL NIL
      ((P1 P17 (((1.0 . 0.0) . (N.N))))
      NIL) . (NIL NIL NIL NIL NIL) )

```

To explain some of this: The first expression corresponds to figure A. It says figure A has been decomposed into three parts, which have been given the names P1, P2, and P3. Then we have a list of properties and relations and similarity information internal to figure A, namely, here, that P2 is inside P3, P1 is above P2, and P1 is above P3. The next two expressions give the corresponding information for figures B and C. The fourth expression gives information about Euclidean similarities between figure A and figure B. For example, P3 goes into P5 under a 'scale factor = 1, rotation angle = 0, and both reflections null' transformation. The next two expressions contain the corresponding information between figure A and figure C and between figure B and figure C, respectively. The seventh list is a five-element list of lists of the parts of the five answer figures; the eighth a five-element list of lists, one for each answer figure, giving their property, relation, and similarity information. The ninth is again a five-element list, each a 'similarity' list from figure C to one of the answer figures. The tenth, and last, expression is a dotted pair of expressions, the first again a five-element list, a 'similarity' list from figure A to each of the answer figures, the second the same from figure B to each of the answer figures. This brief description leaves certain loose ends, but it should provide a reasonably adequate notion of what is done by part I in processing our sample problem.

The ten expressions above are given as arguments to the top-level function of part II

(optimistically called *solve*). The basic method employed by *solve*, which suffices to do this problem, begins by matching the parts of figure A and those of figure B in all possible ways compatible with the similarity information. From this process, it concludes, in the case in question, that $P2 \rightarrow P4$, $P3 \rightarrow P5$, and P1 is removed in going from A to B. (The machinery provided can also handle far more complicated cases, in which alternate matchings are possible and parts are both added and removed.) On the basis of this matching, a statement of a rule taking figure A into figure B is generated. It looks like:

```

(
(REMOVE A1 ((ABOVE A1 A3) (ABOVE
  A1 A2) (SIM OB3 A1 (((1.0 . 0.0) .
  (N.N))))))
(MATCH A2 (((INSIDE A2 A3) (ABOVE
  A1 A2) (SIM OB2 A2 (((1.0 . 0.0) .
  (N.N)))))) . ((LEFT A2 A3) (SIM
  OB2 A2 (((1.0 . 0.0) . (N.N)) ((1.0 .
  3.14) . (N.N)))) (SIMTRAN (((1.0 .
  0.0) . (N.N)) ((1.0 . 3.14) . (N.N)
  ))))))
(MATCH A3 (((INSIDE A2 A3) (ABOVE
  A1 A3) (SIM OB1 A3 (((1.0 . 0.0) .
  (N.N)))))) . ((LEFT A2 A3) (SIM
  OB1 A3 (((1.0 . 0.0) . (N.N))))
  (SIMTRAN (((1.0 . 0.0) . (N.N)
  ))))))
)

```

The A's are used as 'variables' representing objects. The format is rather simple. For each object added, removed, or matched, there is a list of the properties, relations and similarity information pertaining to it. (In the case of a matched object, there are two such lists, one pertaining to its occurrence in figure A and the other to its occurrence in figure B.) There are two special devices; the (SIM OB1 . . .) — form expressions give a means of comparing types of objects between, say, figure A and figure C; the other device is the use of the SIMTRAN expressions in the figure-B list for each matched object. This enables us to handle conveniently some additional situations which we shall omit from consideration, for brevity. They are treated in detail in Ref. 1.

The above rule expresses everything about figures A and B and their relationship that is used in the rest of the process. (The reader may verify that the rule does, in some sense, describe the transformation of figure A into figure B of our example.)

Next, a similarity matching is carried out between figure C and each of the five answer figures. Matchings which do not correspond to the ones between figure A and figure B in numbers of parts added, removed, and matched, are discarded. If all are rejected this method has failed and *solve* goes on to try a further method. In the present case, figures 1 and 5 are rejected on this basis. However, figures 2, 3, and 4 pass this test and are examined further, as follows. Choose an answer figure. For a given matching of figure C to the answer figure in question (and *solve* goes through all possible matchings compatible with similarity) we take each 'figure A → figure B' rule and attempt to fit it to the new case, making all matchings between the A's of the rule statement and the objects of figure C and the answer figures which are compatible with preserving add, remove, and match categories, then testing to see which information is preserved, thus getting a new, 'generalized' rule which fits both 'figure A → figure B' and 'figure C → the answer figure in question.' In our case, for each of the three possible answer figures we get two reduced rules in this way (since there are two possible pairings between A and C, namely, $P1 \leftrightarrow P8$, $P2 \leftrightarrow P6$, and $P3 \leftrightarrow P7$, or $P1 \leftrightarrow P8$, $P2 \leftrightarrow P7$, and $P3 \leftrightarrow P6$).

In some sense, each of these rules provides an answer. However, as pointed out earlier, we want a 'best' or 'strongest' rule, that is, the one that says the most or is the least alteration in the original 'figure A → figure B' rule and that still maps C onto exactly one answer figure. A simple device seems to approximate human opinion on this question rather well; we define a rather crude 'strength' function on the rules and sort them by this. If a rule is a clear winner in this test, the corresponding answer figure is chosen; if the test results in a tie, the entire method has failed and *solve* goes on to try something else. In our case, when the values for the six rules are computed, the winner is one

of the rules corresponding to figure 2, so the program, like all humans so far consulted, chooses it as the answer. The rule chosen looks like this:

```
(
  (REMOVE A1 ((ABOVE A1 A3) (ABOVE
    A1 A2) (SIM OB3 A1 (((1.0 . 0.0) .
      (N.N))))))
  (MATCH A2 (((INSIDE A2 A3) (ABOVE
    A1 A2)) . ((LEFT A2 A3) (SIMTRAN
      (((1.0 . 0.0) . (N.N)) ((1.0 . 3.14) .
        (N.N))))))
  (MATCH A3 (((INSIDE A2 A3) (ABOVE
    A1 A3)) . ((LEFT A2 A3) (SIMTRAN
      (((1.0 . 0.0) . (N.N))))))
)
```

Again, it is easy to check that this rule both takes figure A into figure B and figure C into figure 2, but not into any of the other answer figures.

Further Examples and Comments

(a) Examples

We first exhibit several additional examples of problems given to ANALOGY:

(i) (See Fig. 5)

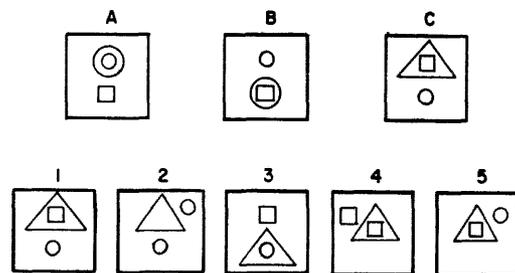


Figure 5.

Here the rule involves changes in the relations of the three parts. ANALOGY chose answer figure 3.

(ii) (See Fig. 6)

This case involves both addition and removal of objects. ANALOGY chose answer figure 2.

(iii) (See Fig. 7)

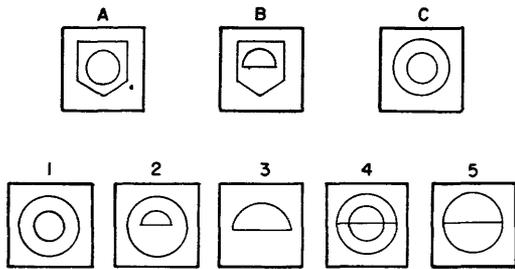


Figure 6.

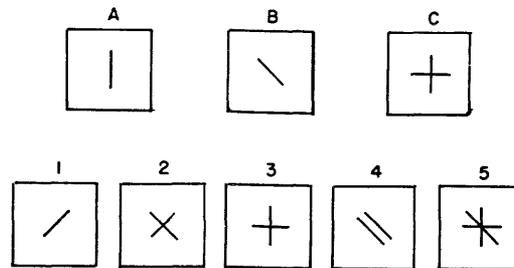


Figure 8.

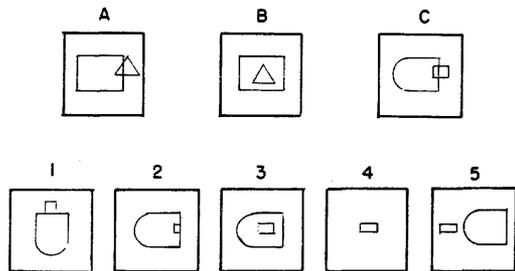


Figure 7.

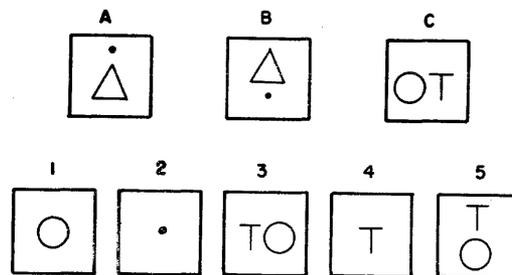


Figure 9.

Note that this case required the more powerful decomposition program. Here ANALOGY chose figure 3.

(iv) (See Fig. 8)

The rule here simply involved a rotation. ANALOGY chose figure 2.

(v) (See Fig. 9)

Here ANALOGY chose figure 3, using an extension of the part II techniques discussed above. This extension, employed after failure of the basic process, involves systematic substitution of certain specified relations (e.g., LEFT for ABOVE) for others in the part II input descriptions, thus making it possible for ANALOGY to relate the 'vertical' transformation taking A into B to the 'horizontal' transformation taking C into 3.

(vi) In the problem of Fig. 1, the large circle of answer figure 4 was replaced by a large square and the problem rerun. Again figure 4 was chosen but by a different rule. Now, instead of the inner object being removed, as be-

fore, the outer object is removed and the inner one enlarged. This illustrates some of the flexibility of the procedure and the dependence of the answer choice on the range of allowed answers as well as on A, B, and C.

(vii) (See Fig. 10)

Here is an example of a failure by ANALOGY to agree with the human consensus which favors figure 5. ANALOGY chose figure 3.

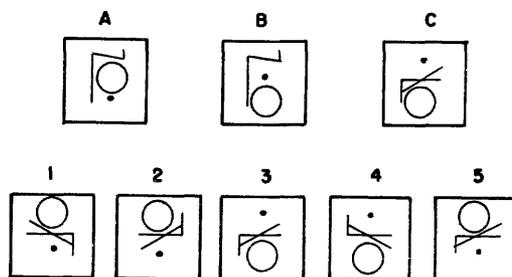


Figure 10.

(b) *Comparison with Human Performance*

We can only roughly compare the performance of ANALOGY with that of humans on geometric-analogy problems, since ANALOGY has not yet been given the complete set of such problems from any test for which scores are available. However, as some indication, we cite scores on the ACE tests based on a period of years including those editions of the test from which most of the problems on which ANALOGY was tested were selected. These scores are for a large population of college-preparatory students; the median score, on a test consisting of 30 such questions, ranged from 17 for 9th grade to 20 for 12th grade. Our estimate is that, on the same tests, ANALOGY, as it currently exists, could solve between 15 and 20 problems. Given, in addition, certain changes (mostly in part I, e.g., a more powerful decomposition program and additional properties and relations) for which we have reasonably well-worked-out implementations in mind, ANALOGY should be capable of perhaps 25 successful solutions.

(c) *The Use of LISP*

The use of a list-processing language to construct the ANALOGY program appears to have been a suitable choice; most notably, its capability at handling intermediate expressions of unpredictable size and 'shape' (such as our figure descriptions and transformation rules) is of great value. We especially wish to praise LISP as a convenient and elegant language in which to write and debug complex programs. The ease of composition of routines, the highly mnemonic nature of the language, and the good tracing facilities all contribute greatly to effective program construction. In return for the use of such a language one pays a certain price in speed and storage space, which, in the case of ANALOGY, at least, was a very acceptable bargain, since the necessity of machine-language coding would have made the entire project unfeasible. Incidentally, the ANALOGY program (apparently the largest program written in LISP to date) is so large that parts I and II must occupy core separately. The consequent limited (and one-way) communication between the parts was a serious design constraint but proved to have some compensating advantages in simplicity.

ANALOGY and Pattern-Recognition in Problem-Solving Programs

In this section we shall consider certain aspects of the design of problem-solving machines. To aid this discussion we shall specify (rather loosely) a subclass of problem-solving machines and carry out our discussion in terms of these though the ideas involved are by no means limited in applicability to this class. The machines we have in mind are typified by GPS¹⁴ in that the problem to be solved by the machine is to transform one specified 'object' or 'situation' (whatever this may mean in a particular subject-matter context) into another by applying an appropriate sequence of transformations chosen from a class available to the machine. A wide variety of problems may be cast in this form (again see Ref. 14 or other discussions of GPS by the same authors). As in GPS, subgoals may be generated and attacked by such a machine and elaborate schemes of resource allocation may be required. However, these aspects do not concern us here. Our interest lies in the basic task of the machine; given a pair of 'objects,' it must choose an 'appropriate' transformation, i.e., one contributing to the goal of transforming one of the given 'objects' into the other.

It is a widely-held view, with which we agree completely, that for a machine to be capable of highly intelligent behavior on a task of this kind, in a rich environment of objects and transformations (and, in particular, to be capable of learning at a level more advanced than that of present machines), the critical factor is that it have a good internal representation of both its subject matter ('objects') and its methods ('transformations'), as well as an elaborate set of 'pattern-recognition' techniques for matching transformations to object pairs. Probably this means a quite 'verbal' representation of both objects and transformations as expressions in suitable 'description languages.' Furthermore, these matching techniques must be represented in a form in which they themselves are capable of being improved as the machine gains experience. The central role which 'pattern-recognition' techniques must play in sophisticated problem-solving programs and the corresponding importance for effective learning of autonomous improvement in the perform-

ance of these techniques are well expressed in Minsky.¹² There we find:

In order not to try all possibilities a resourceful program must classify problem situations into categories associated with the domains of effectiveness of the machine's different methods. These pattern-recognition methods must extract the heuristically significant features of the objects in question. Again from Ref. 12 we have:

Again from ¹² we have:

In order to solve a new problem one uses what might be called the basic learning heuristic—first try using methods similar to those which have worked, in the past, on similar problems.

Here, the problem is, of course, to have pattern-recognition techniques possessing, or able themselves to learn, criteria of 'similarity' appropriate to the subject matter in question.

The 'fixed-length property-list' schemes (see Ref. 12) which characteristically have been used to perform this pattern-recognition task in current problem-solving programs have two principal defects which limit their extension to harder problems:

(i) While, in principle, given enough sufficiently elaborate properties, one can make arbitrarily fine discriminations, in practice a given set of properties will begin to fail rapidly as situations become more complex. In particular, for 'situations' which must be treated as consisting of interrelated parts, the 'global' nature of the scheme in question leaves it helpless.

(ii) Such a scheme is very limited in its learning capabilities, since it has access to very little information about its component properties; in particular, it is incapable of "knowledgeably" modifying its tests or adding new ones—it can only modify the weightings given to the results of these tests in its 'decisions.'

In view of the limitations of the 'property-list' pattern-recognition scheme just mentioned, we can formulate some requirements for a pattern-recognition scheme suitable to replace it as a 'transformation-selecting' mechanism. First, the scheme must have access to a representation of each 'object' in terms of a 'descriptive framework' for the subject matter in

question which is suitable in that useful relationships between 'objects' can be extracted relatively simply from the corresponding representations. Furthermore, the transformation-selecting rules of the pattern-recognition apparatus should themselves be expressed in a representation suitable for a 'learning mechanism' to revise the set of rules (i) by adding new rules and deleting those old ones which prove least useful as experience associates certain object pairs with certain transformations and (ii) by replacing a set of particular rules by a 'common generalization' rule again represented in the same language. Such facilities could go far toward removing the limitations of which we have spoken and providing both a powerful rule language (the rules can be stated in terms of the 'descriptive framework' we have postulated for the 'objects') and a learning mode more sophisticated than any yet incorporated in such a general problem-solving program.

So far we have been enumerating desirable features in a 'pattern-recognition' mechanism to be used as a transformation-selection device within a large problem-solver. What has all this to do with ANALOGY, which is not even a problem-solving program of the class we have been considering? We suggest that ANALOGY can, under a suitable (rather drastic) reinterpretation, be to some extent viewed as a pattern-recognition program having, to the limited degree appropriate for its particular environment, all the features we have listed. First, the 'objects' are the problem figures of ANALOGY and the suitable 'descriptive framework' appropriate to these objects is the 'subfigure and relation' representation used as the input part I generates for part II of ANALOGY. (Thus part I of ANALOGY corresponds to the apparatus that generates this representation for each object; that is, it goes from a representation of the 'problem objects' which is convenient for input to the problem-solver to one which is in a form suitable for internal use.) The generation in ANALOGY of a transformation rule taking one answer figure into another can be thought of as corresponding to the first kind of learning we listed above, namely, the adding of rules as, with experience, the machine associates certain object pairs with certain simple or com-

posite transformations. Finally, the common generalization of two rules in ANALOGY corresponds to the second kind of learning we mentioned, namely, the generation of a common generalization of several rules associating 'objects' and 'transformations.' Furthermore, ANALOGY's process of choosing between 'common generalizations' of different rule pairs mirrors a process of selectively incorporating only those generalizations with the greatest discriminatory power. Under this interpretation, ANALOGY appears as a model for a pattern-recognition process with all the characteristics mentioned. The potential value of ANALOGY, viewed in this way, as a suggestive model for the construction of such pattern-recognition mechanisms for use within problem-solving programs may prove to be the chief product of our work with ANALOGY and the best justification for having carried it out.

References

1. T. G. EVANS, PH.D. Thesis, Department of Mathematics, MIT, June, 1963 (soon to be available as an AFCRL Technical Report).
2. J. MCCARTHY, "Recursive functions of symbolic expressions," *Comm. ACM*, Vol. 3, April, 1960.
3. J. MCCARTHY *et al.*, *LISP 1.5 Programmer's Manual*, MIT, revised edition, August, 1962.
4. R. L. GRIMSDALE, F. H. SUMNER, C. J. TUNIS, and T. KILBURN *et al.*, "A system for the automatic recognition of patterns," *Proc. IEE*, March, 1959, Vol. 106, pt. B, pp. 210-221.
5. T. MARILL, A. K. HARTLEY, T. G. EVANS, B. H. BLOOM, D. M. R. PARK, T. P. HART, and D. L. DARLEY, "CYCLOPS-1: a second-generation recognition system," *FJCC*, Las Vegas, Nevada, November, 1963.
6. H. SHERMAN, "A quasi-topological method for the recognition of line patterns," *Proc. ICIP*, Paris, France, June, 1959, pp. 232-238.
7. I. SUTHERLAND, "Sketchpad: a man-machine graphical communication system," *SJCC*, Detroit, Michigan, May, 1963.
8. L. ROBERTS, PH.D. Thesis, Department of Electrical Engineering, MIT, June, 1963.
9. R. KIRSCH, personal communication.
10. L. HODES, "Machine processing of line drawings," *Lincoln Laboratory Technical Memorandum*, March, 1961.
11. R. CANADAY, M.S. Thesis, Department of Electrical Engineering, MIT, February, 1962.
12. M. L. MINSKY, "Steps toward artificial intelligence," *Proc. IRE*, January, 1961, pp. 8-30.
13. A. NEWELL and H. A. SIMON, "The logic theory machine," *IRE Trans. on Information Theory*, Vol. IT-2, #3, September, 1956, pp. 61-79.
14. A. NEWELL, J. C. SHAW, and H. A. SIMON, "Report on a general problem-solving program," *Proc. ICIP*, Paris, France, June, 1959, pp. 256-264.
15. H. GELERENTER and N. ROCHESTER, "Intelligent behavior in problem-solving machines," *IBM J. Res. and Dev.*, Vol. 2, #4, October, 1958, pp. 336-345.
16. J. SLAGLE, PH.D. Thesis, Department of Mathematics, MIT, June, 1961.
17. T. G. EVANS, "The use of list-structured descriptions for programming manipulations on line drawings," *ACM National Conference*, Denver, Colorado, August, 1963.

Acknowledgements

The assistance of the Cooperative Test Division of the Educational Testing Service, Princeton, New Jersey, in providing a large set of geometric-analogy questions from its files is gratefully acknowledged.

Thanks are also due to the Educational Records Bureau, New York, N.Y., for the statistics on human performance on geometrical-analogy questions cited in Sec. 4b.

Most of the computation associated with the development and testing of ANALOGY was performed at the MIT Computation Center.