which I can give no answers. There has as yet been no complete agreement on what a frame really is, or on how to implement frames in AI programs. I make my own stab at discussing some of these questions in the following section, where I talk about some puzzles in visual pattern recognition, which I call "Bongard problems".

Bongard Problems

Bongard problems (BP's) are problems of the general type given by the Russian scientist M. Bongard in his book *Pattern Recognition.* A typical BP—number 51 in his collection of one hundred—is shown in Figure 119.

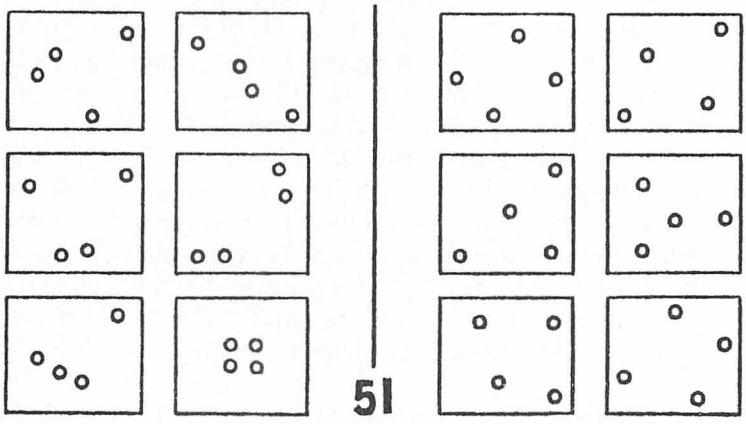


FIGURE 119. Bongard problem 51. [*From M. Bongard,* Pattern Recognition *(Rochelle Park, N. J.: Hayden Book Co., Spartan Books, 1970).*]

These fascinating problems are intended for pattern-recognizers, whether human or machine. (One might also throw in ETI's—extraterrestrial intelligences.) Each problem consists of twelve boxed figures (henceforth called *boxes*): six on the left, forming *Class I,* and six on the right, forming *Class II.* The boxes may be indexed this way:

I-A I-B	II-A II-B
I-C I-D	II-C II-D
I-E I-F	II-E II-F

The problem is "How do Class I boxes differ from Class II boxes?"

A Bongard problem-solving program would have several stages, in which raw data gradually get converted into descriptions. The early stages are relatively inflexible, and higher stages become gradually more flexible. The final stages have a property which I call *tentativity,* which means simply that the way a picture is represented is always tentative. Upon the drop of a hat, a high-level description can be restructured, using all the devices of the

later stages. The ideas presented below also have a tentative quality to them. I will try to convey overall ideas first, glossing over significant difficulties. Then I will go back and try to explain subtleties and tricks and so forth. So your notion of how it all works may also undergo some revisions as you read. But that is in the spirit of the discussion.

Preprocessing Selects a Mini-vocabulary

Suppose, then, that we have some Bongard problem which we want to solve. The problem is presented to a TV camera and the raw data are read in. Then the raw data are *preprocessed.* This means that some salient features are detected. The *names* of these features constitute a "mini-vocabulary" for the problem; they are drawn from a general "salient-feature vocabulary". Some typical terms of the salient-feature vocabulary are:

> line segment, curve, horizontal, vertical, black, white, big, small, pointy, round . . .

In a second stage of preprocessing, some knowledge about elementary *shapes* is used; and if any are found, their names are also made available. Thus, terms such as

> triangle, circle, square, indentation, protrusion, right angle, vertex, cusp, arrow . . .

may be selected. This is roughly the point at which the conscious and the unconscious meet, in humans. This discussion is primarily concerned with describing what happens from here on out.

High-Level Descriptions

Now that the picture is "understood", to some extent, in terms of familiar concepts, some looking around is done. Tentative descriptions are made for one or a few of the twelve boxes. They will typically use simple descriptors such as

> above, below, to the right of, to the left of, inside, outside of, close to, far from, parallel to, perpendicular to, in a row, scattered, evenly spaced, irregularly spaced, etc.

Also, definite and indefinite numerical descriptors can be used:

> 1, 2, 3, 4, 5, . . . many, few, etc.

More complicated descriptors may be built up, such as

> further to the right of, less close to, almost parallel to, etc.

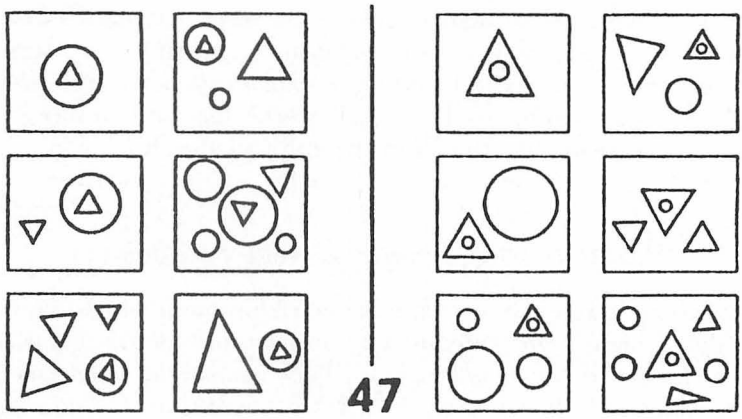


FIGURE 120. Bongard problem 47. [*From M. Bongard,* Pattern Recognition.]

Thus, a typical box—say I-F of BP 47 (Fig. 120)—could be variously described as having:

three shapes
or
three white shapes
or
a circle on the right
or
two triangles and a circle
or
two upwards-pointing triangles
or
one large shape and two small shapes
or
one curved shape and two straight-edged shapes
or
a circle with the same kind of shape on the inside and outside.

Each of these descriptions sees the box through a "filter". Out of context, any of them might be a useful description. As it turns out, though, all of them are "wrong", in the context of the particular Bongard problem they are part of. In other words, if you knew the distinction between Classes I and II in BP 47, and were given one of the preceding lines as a description of an unseen drawing, that information would not allow you to tell to which Class the drawing belonged. The essential feature of this box, in context, is that it includes

a circle containing a triangle.

Note that someone who heard such a description would not be able to *reconstruct* the original drawing, but would be able to *recognize* drawings

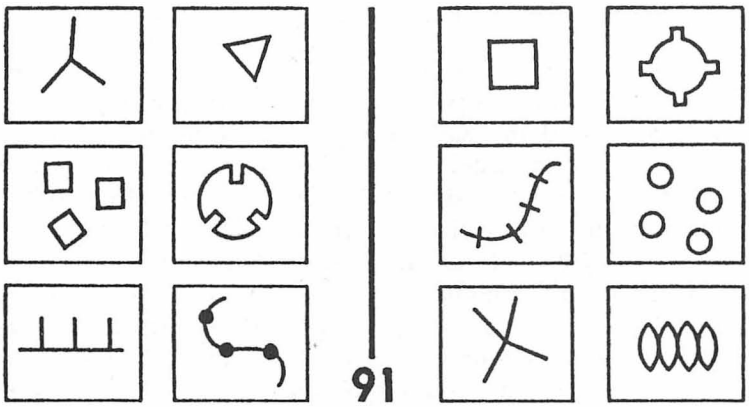


FIGURE 121. Bongard problem 91. [*From M. Bongard,* Pattern Recognition.]

which have this property. It is a little like musical style: you may be an infallible recognizer of Mozart, but at the same time unable to write anything which would fool anybody into thinking it was by Mozart.

Now consider box I-D of BP 91 (Fig. 121). An overloaded but "right" description in the context of BP 91 is

a circle with three rectangular intrusions.

Notice the sophistication of such a description, in which the word "with" functions as a disclaimer, implying that the "circle" is not really a circle: it is *almost* a circle, except that . . . Furthermore, the intrusions are not full rectangles. There is a lot of "play" in the way we use language to describe things. Clearly, a lot of information has been thrown away, and even more could be thrown away. A priori, it is very hard to know what it would be smart to throw away and what to keep. So some sort of method for an intelligent compromise has to be encoded, via heuristics. Of course, there is always recourse to lower levels of description (i.e., less chunked descriptions) if discarded information has to be retrieved, just as people can constantly look at the puzzle for help in restructuring their ideas about it. The trick, then, is to devise explicit rules that say how to

make tentative descriptions for each box;
compare them with tentative descriptions for other boxes of either Class;
restructure the descriptions, by
(i) adding information,
(ii) discarding information,
or *(iii)* viewing the same information from another angle;
iterate this process until finding out what makes the two Classes differ.

Templates and Sameness-Detectors

One good strategy would be to try to make descriptions *structurally similar to each other,* to the extent this is possible. Any structure they have in common will make comparing them that much easier. Two important elements of this theory deal with this strategy. One is the idea of "description-schemas", or *templates;* the other is the idea of *Sam*—a "sameness detector".

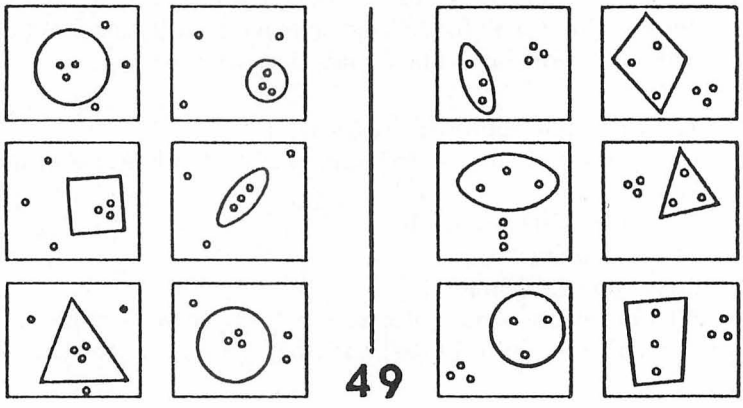
First Sam. Sam is a special agent present on all levels of the program. (Actually there may be different kinds of Sams on different levels.) Sam constantly runs around within individual descriptions and within different descriptions, looking for descriptors or other things which are repeated. When some sameness is found, various restructuring operations can be triggered, either on the single-description level or on the level of several descriptions at once.

Now templates. The first thing that happens after preprocessing is an attempt to manufacture a template, or description-schema—a *uniform format* for the descriptions of all the boxes in a problem. The idea is that a description can often be broken up in a natural way into subdescriptions, and those in turn into subsubdescriptions, if need be. The bottom is hit when you come to primitive concepts which belong to the level of the preprocessor. Now it is important to choose the way of breaking descriptions into parts so as to reflect commonality among all the boxes; otherwise you are introducing a superfluous and meaningless kind of "pseudo-order" into the world.

On the basis of what information is a template built? It is best to look at an example. Take BP 49 (Fig. 122). Preprocessing yields the information that each box consists of several little o's, and one large closed curve. This is a valuable observation, and deserves to be incorporated in the template. Thus a first stab at a template would be:

large closed curve: ——
small o's: ——

FIGURE 122. Bongard problem 49. [*From M. Bongard,* Pattern Recognition.]



It is very simple: the description-template has two explicit *slots* where subdescriptions are to be attached.

A Heterarchical Program

Now an interesting thing happens, triggered by the term "closed curve". One of the most important modules in the program is a kind of semantic net—the *concept network*—in which all the known nouns, adjectives, etc., are linked in ways which indicate their interrelations. For instance, "closed curve" is strongly linked with the terms "interior" and "exterior". The concept net is just brimming with information about relations between terms, such as what is the opposite of what, what is similar to what, what often occurs with what, and so on. A little portion of a concept network, to be explained shortly, is shown in Figure 123. But let us follow what happens now, in the solution of problem 49. The concepts "interior" and "exterior" are activated by their proximity in the net to "closed curve". This suggests to the template-builder that it might be a good idea to make distinct slots for the interior and exterior of the curve. Thus, in the spirit of tentativity, the template is tentatively restructured to be this:

large closed curve: ——
little o's in interior: ——
little o's in exterior: ——

Now when subdescriptions are sought, the terms "interior" and "exterior" will cause procedures to inspect those specific regions of the box. What is found in BP 49, box I-A is this:

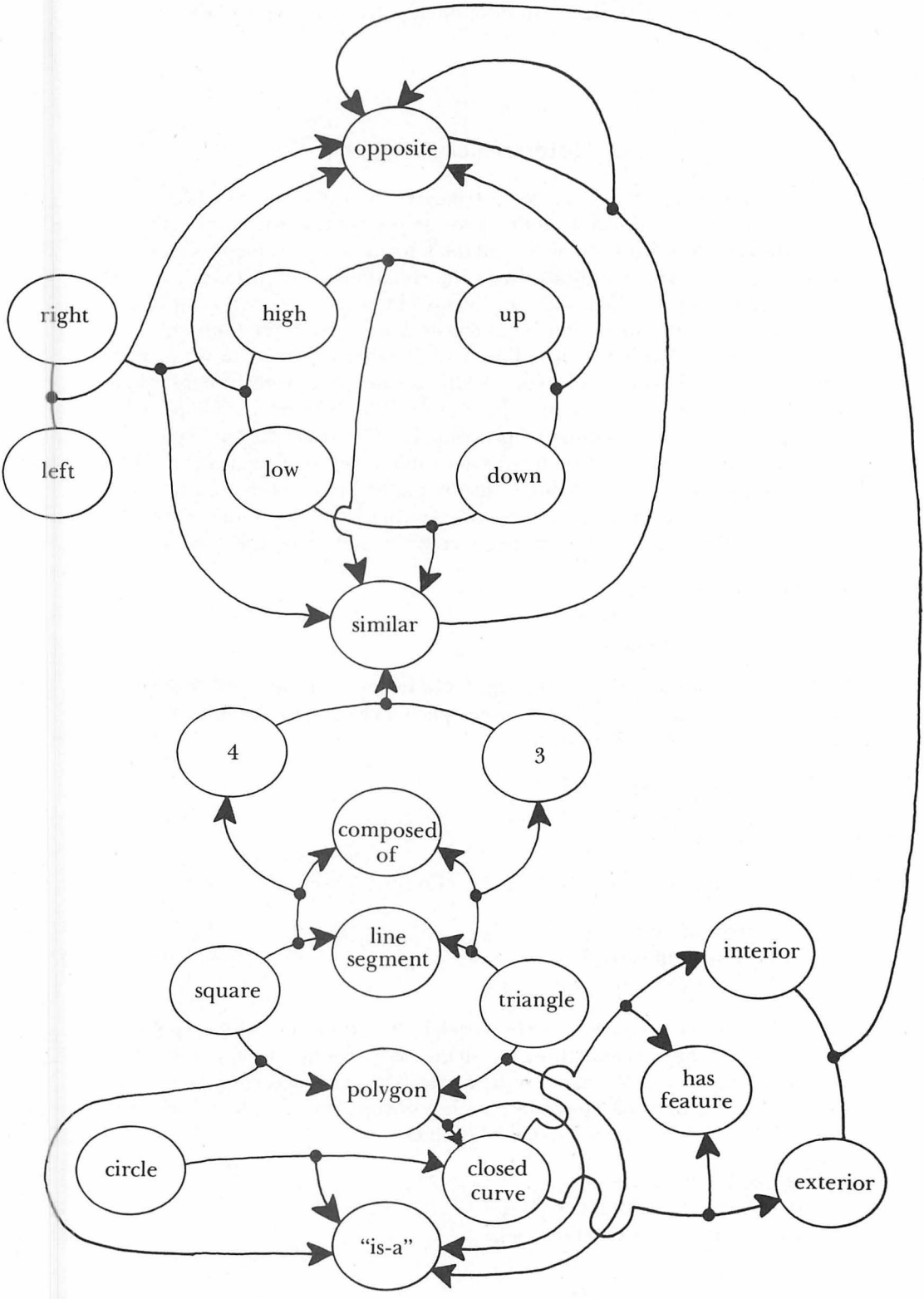
large closed curve: *circle*
little o's in interior: *three*
little o's in exterior: *three*

And a description of box II-A of the same BP might be

large closed curve: *cigar*
little o's in interior: *three*
little o's in exterior: *three*

Now Sam, constantly active in parallel with other operations, spots the recurrence of the concept "three" in all the slots dealing with o's, and this is strong reason to undertake a second template-restructuring operation. Notice that the first was suggested by the concept net, the second by Sam. Now our template for problem 49 becomes:

large closed curve: ——
three little o's in interior: ——
three little o's in exterior: ——


opposite
right
high
up
left
low
down
similar
4
3
composed of
line segment
square
triangle
interior
has feature
polygon
circle
closed curve
exterior
"is-a"

Now that "three" has risen one level of generality—namely, into the template—it becomes worthwhile to explore its neighbors in the concept network. One of them is "triangle", which suggests that triangles of o's may be important. As it happens, this leads down a blind alley—but how could you know in advance? It is a typical blind alley that a human would explore, so it is good if our program finds it too! For box II-E, a description such as the following might get generated:

large closed curve: *circle*
three little o's in interior: *equilateral triangle*
three little o's in exterior: *equilateral triangle*

Of course an enormous amount of information has been thrown away concerning the sizes, positions, and orientations of these triangles, and many other things as well. But that is the whole point of making descriptions instead of just using the raw data! It is the same idea as funneling, which we discussed in Chapter XI.

The Concept Network

We need not run through the entire solution of problem 49; this suffices to show the constant back-and-forth interaction of individual descriptions, templates, the sameness-detector Sam, and the concept network. We should now look a little more in detail at the concept network and its function. A simplified portion shown in the figure codes the following ideas:

"High" and "low" are opposites.
"Up" and "down" are opposites.
"High" and "up" are similar.
"Low" and "down" are similar.
"Right" and "left" are opposites.
The "right-left" distinction is similar to the "high-low" distinction.
"Opposite" and "similar" are opposites.

Note how everything in the net—both nodes and links—can be talked about. In that sense nothing in the net is on a higher level than anything else. Another portion of the net is shown; it codes for the ideas that

A square is a polygon.
A triangle is a polygon.
A polygon is a closed curve.

FIGURE 123. A small portion of a concept network for a program to solve Bongard problems. "Nodes" are joined by "links", which in turn can be linked. By considering a link as a verb and the nodes it joins as subject and object, you can pull out some English sentences from this diagram.

The difference between a triangle and a square is that one has 3 sides and the other has 4.
4 is similar to 3.
A circle is a closed curve.
A closed curve has an interior and an exterior.
"Interior" and "exterior" are opposites.

The network of concepts is necessarily very vast. It seems to store knowledge only statically, or declaratively, but that is only half the story. Actually, its knowledge borders on being procedural as well, by the fact that the proximities in the net act as guides, or "programs", telling the main program how to develop its understanding of the drawings in the boxes.

For instance, some early hunch may turn out to be wrong and yet have the germ of the right answer in it. In BP 33 (Fig. 124), one might at first

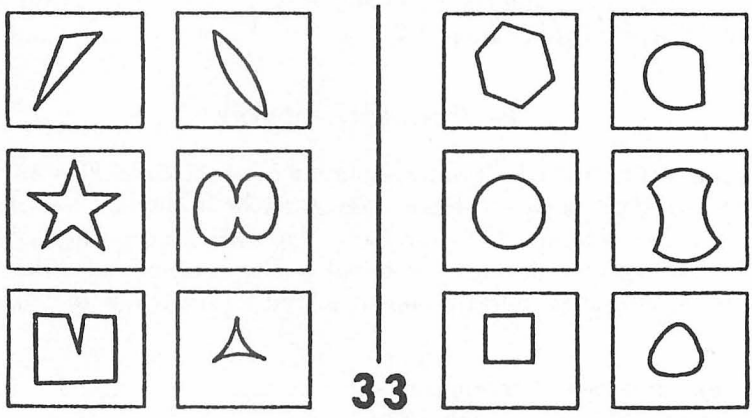


FIGURE 124. Bongard problem 33. [*From M. Bongard,* Pattern Recognition.]

jump to the idea that Class I boxes contain "pointy" shapes, Class II boxes contain "smooth" ones. But on closer inspection, this is wrong. Nevertheless, there is a worthwhile insight here, and one can try to push it further, by sliding around in the network of concepts beginning at "pointy". It is close to the concept "acute", which is precisely the distinguishing feature of Class I. Thus one of the main functions of the concept network is to allow early wrong ideas to be modified slightly to slip into variations which may be correct.

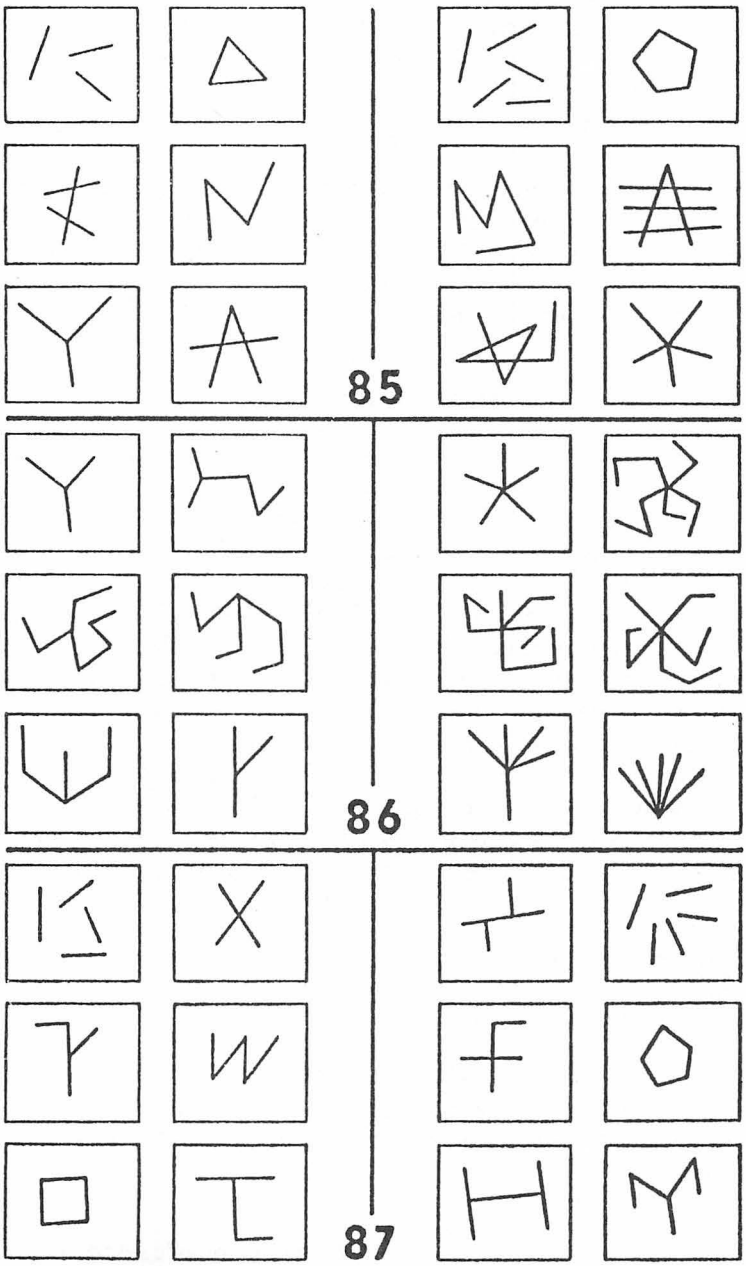
Slippage and Tentativity

Related to this notion of slipping between closely related terms is the notion of seeing a given object as a variation on another object. An excellent example has been mentioned already—that of the "circle with three indentations", where in fact there is no circle at all. One has to be able to bend concepts, when it is appropriate. Nothing should be absolutely rigid. On

the other hand, things shouldn't be so wishy-washy that nothing has any meaning at all, either. The trick is to know when and how to slip one concept into another.

An extremely interesting set of examples where slipping from one description to another is the crux of the matter is given in Bongard problems 85-87 (Fig. 125). BP 85 is rather trivial. Let us assume that our program identifies "line segment" in its preprocessing stage. It is relatively simple for it then to count line segments and arrive at the difference

FIGURE 125. Bongard problems 85-87. [*From M. Bongard,* Pattern Recognition.]



between Class I and Class II in BP 85. Now it goes on to BP 86. A general heuristic which it uses is to *try out recent ideas which have worked.* Successful repetition of recent methods is very common in the real world, and Bongard does not try to outwit this kind of heuristic in his collection—in fact, he reinforces it, fortunately. So we plunge right into problem 86 with two ideas ("count" and "line segment") fused into one: "count line segments". But as it happens, the trick of BP 86 is to count line *trains* rather than line *segments,* where "line train" means an end-to-end concatenation of (one or more) line segments. One way the program might figure this out is if the concepts "line train" and "line segment" are both known, and are close in the concept network. Another way is if it can *invent* the concept of "line train"—a tricky proposition, to say the least.

Then comes BP 87, in which the notion of "line segment" is further played with. When is a line segment three line segments? (See box II-A.) The program must be sufficiently flexible that it can go back and forth between such different representations for a given part of a drawing. It is wise to store old representations, rather than forgetting them and perhaps having to reconstruct them, for there is no guarantee that a newer representation is better than an old one. Thus, along with each old representation should be stored some of the reasons for liking it and disliking it. (This begins to sound rather complex, doesn't it?)

Meta-Descriptions

Now we come to another vital part of the recognition process, and that has to do with levels of abstraction and meta-descriptions. For this let us consider BP 91 (Fig. 121) again. What kind of template could be constructed here? There is such an amount of variety that it is hard to know where to begin. But this is in itself a clue! The clue says, namely, that the class distinction very likely exists on a higher level of abstraction than that of geometrical description. This observation clues the program that it should construct *descriptions of descriptions*—that is, *meta-descriptions.* Perhaps on this second level some common feature will emerge; and if we are lucky, we will discover enough commonality to guide us towards the formulation of a template for the meta-descriptions! So we plunge ahead without a template, and manufacture descriptions for various boxes; then, once these descriptions have been made, we describe *them.* What kinds of slot will our template for meta-descriptions have? Perhaps these, among others:

concepts used: ——
recurring concepts: ——
names of slots: ——
filters used: ——

There are many other kinds of slots which might be needed in meta-descriptions, but this is a sample. Now suppose we have described box I-E of BP 91. Its (template-less) description might look like this:

horizontal line segment
vertical line segment mounted on the horizontal line segment
vertical line segment mounted on the horizontal line segment
vertical line segment mounted on the horizontal line segment

Of course much information has been thrown out: the fact that the three vertical lines are of the same length, are spaced equidistantly, etc. But it is plausible that the above description would be made. So the meta-description might look like this:

concepts used: *vertical-horizontal, line segment, mounted on*
repetitions in description: *3 copies of "vertical line segment mounted on the horizontal line segment"*
names of slots: ——
filters used: ——

Not all slots of the meta-description need be filled in; information can be thrown away on this level as well as on the "just-plain-description" level.

Now if we were to make a description for any of the other boxes of Class I, and then a meta-description of it, we would wind up filling the slot "repetitions in description" each time with the phrase "3 copies of . . ." The sameness-detector would notice this, and pick up *three-ness* as a salient feature, on quite a high level of abstraction, of the boxes of Class I. Similarly, *four-ness* would be recognized, via the method of meta-descriptions, as the mark of Class II.

Flexibility Is Important

Now you might object that in this case, resorting to the method of meta-descriptions is like shooting a fly with an elephant gun, for the three-ness versus four-ness might as easily have shown up on the lower level if we had constructed our descriptions slightly differently. Yes, true—but it is important to have the possibility of solving these problems by different routes. There should be a large amount of flexibility in the program; it should not be doomed if, malaphorically speaking, it "barks up the wrong alley" for a while. (The amusing term "malaphor" was coined by the newspaper columnist Lawrence Harrison; it means a cross between a malapropism and a metaphor. It is a good example of "recombinant ideas".) In any case, I wanted to illustrate the general principle that says: When it is hard to build a template because the preprocessor finds too much diversity, that should serve as a clue that concepts on a higher level of abstraction are involved than the preprocessor knows about.

Focusing and Filtering

Now let us deal with another question: ways to throw information out. This involves two related notions, which I call "focusing" and "filtering". *Focus-*

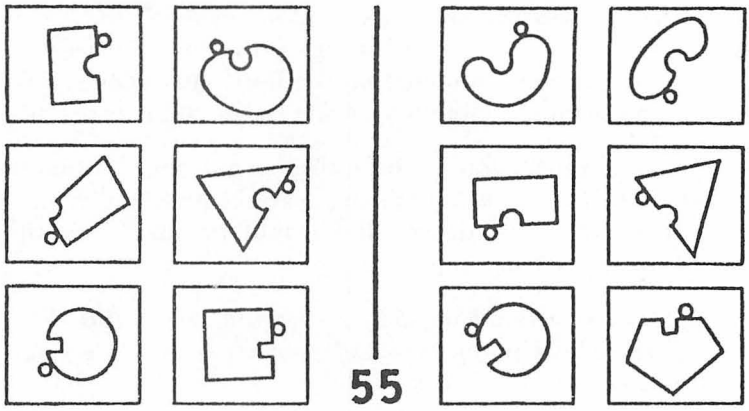


FIGURE 126. Bongard problem 55. [*From M. Bongard,* Pattern Recognition.]

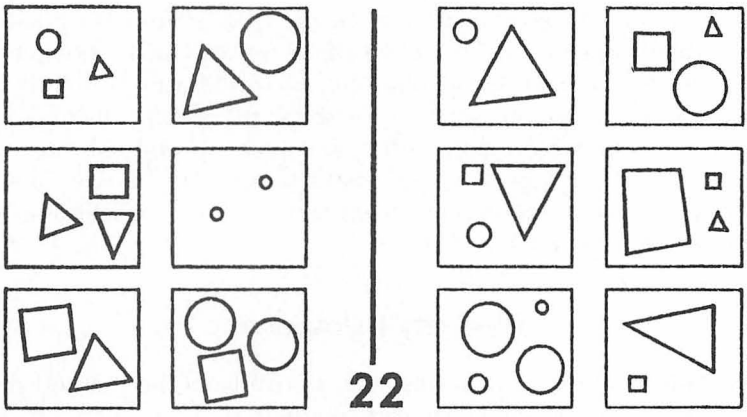


FIGURE 127. Bongard problem 22. [*From M. Bongard,* Pattern Recognition.]

ing involves making a description whose focus is some part of the drawing in the box, to the exclusion of everything else. *Filtering* involves making a description which concentrates on some particular way of viewing the contents of the box, and deliberately ignores all other aspects. Thus they are complementary: focusing has to do with objects (roughly, nouns), and filtering has to do with concepts (roughly, adjectives). For an example of focusing, let's look at BP 55 (Fig. 126). Here, we focus on the indentation and the little circle next to it, to the exclusion of the everything else in the box. BP 22 (Fig. 127) presents an example of filtering. Here, we must filter out every concept but that of size. A combination of focusing and filtering is required to solve problem BP 58 (Fig. 128).

One of the most important ways to get ideas for focusing and filtering is by another sort of "focusing": namely, by inspection of a single particularly simple box—say one with as few objects in it as possible. It can be

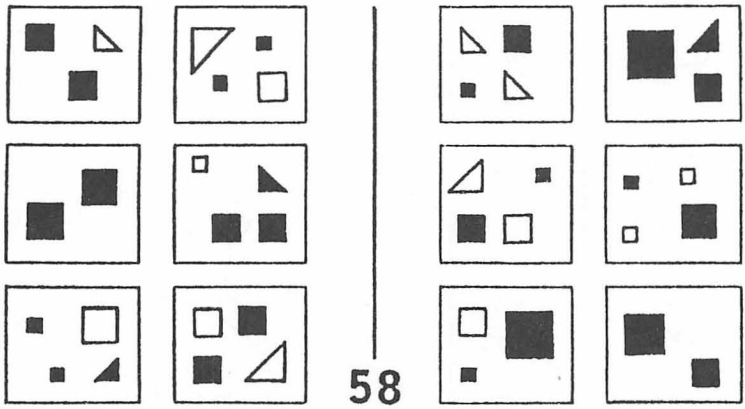


FIGURE 128. Bongard problem 58. [*From M. Bongard,* Pattern Recognition.]

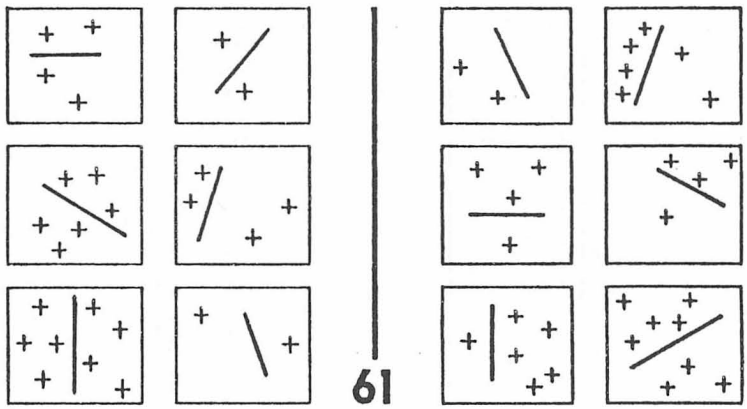


FIGURE 129. Bongard problem 61. [*From M. Bongard,* Pattern Recognition.]

extremely helpful to compare the starkest boxes from the two Classes. But how can you tell which boxes are stark until you have descriptions for them? Well, one way of detecting starkness is to look for a box with a minimum of the features provided by the preprocessor. This can be done very early, for it does not require a pre-existing template; in fact, this can be one useful way of discovering features to build into a template. BP 61 (Fig. 129) is an example where that technique might quickly lead to a solution.

Science and the World of Bongard Problems

One can think of the Bongard-problem world as a tiny place where "science" is done—that is, where the purpose is to discern patterns in the world. As patterns are sought, templates are made, unmade, and remade;

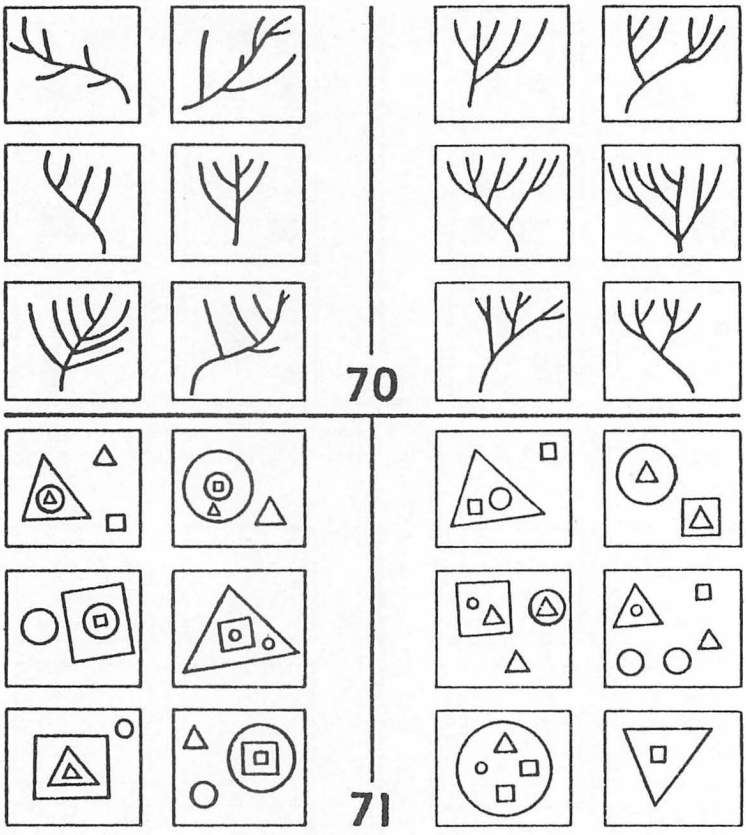


FIGURE 130. Bongard problems 70-71. [*From M. Bongard,* Pattern Recognition.]

slots are shifted from one level of generality to another; filtering and focusing are done; and so on. There are discoveries on all levels of complexity. The Kuhnian theory that certain rare events called "paradigm shifts" mark the distinction between "normal" science and "conceptual revolutions" does not seem to work, for we can see paradigm shifts happening all throughout the system, all the time. The fluidity of descriptions ensures that paradigm shifts will take place on all scales.

Of course, some discoveries are more "revolutionary" than others, because they have wider effects. For instance, one can make the discovery that problems 70 and 71 (Fig. 130) are "the same problem", when looked at on a sufficiently abstract level. The key observation is that both involve depth-2 versus depth-1 nesting. This is a new level of discovery that can be made about Bongard problems. There is an even higher level, concerning the collection as a whole. If someone has never seen the collection, it can be a good puzzle just to figure out what it is. To figure it out is a revolutionary insight, but it must be pointed out that the mechanisms of thought which allow such a discovery to be made are no different from those which operate in the solution of a single Bongard problem.

By the same token, real science does not divide up into "normal" periods versus "conceptual revolutions"; rather, paradigm shifts pervade—there are just bigger and smaller ones, paradigm shifts on different levels. The recursive plots of INT and Gplot (Figs. 32 and 34) provide a geometric model for this idea: they have the same structure full of discontinuous jumps on every level, not just the top level—only the lower the level, the smaller the jumps.

Connections to Other Types of Thought

To set this entire program somewhat in context, let me suggest two ways in which it is related to other aspects of cognition. Not only does it depend on other aspects of cognition, but also they in turn depend on it. First let me comment on how it depends on other aspects of cognition. The intuition which is required for knowing when it makes sense to blur distinctions, to try redescriptions, to backtrack, to shift levels, and so forth, is something which probably comes only with much experience in thought in general. Thus it would be very hard to define heuristics for these crucial aspects of the program. Sometimes one's experience with real objects in the world has a subtle effect on how one describes or redescribes boxes. For instance, who can say how much one's familiarity with living trees helps one to solve BP 70? It is very doubtful that in humans, the subnetwork of concepts relevant to these puzzles can be easily separated out from the whole network. Rather, it is much more likely that one's intuitions gained from seeing and handling real objects—combs, trains, strings, blocks, letters, rubber bands, etc., etc.—play an invisible but significant guiding role in the solution of these puzzles.

Conversely, it is certain that understanding real-world situations heavily depends on visual imagery and spatial intuition, so that having a powerful and flexible way of representing patterns such as these Bongard patterns can only contribute to the general efficiency of thought processes.

It seems to me that Bongard's problems were worked out with great care, and that they have a quality of universality to them, in the sense that each one has a unique correct answer. Of course one could argue with this and say that what we consider "correct" depends in some deep way on our being human, and some creatures from some other star system might disagree entirely. Not having any concrete evidence either way, I still have a certain faith that Bongard problems depend on a sense of simplicity which is not just limited to earthbound human beings. My earlier comments about the probable importance of being acquainted with such surely earth-limited objects as combs, trains, rubber bands, and so on, are not in conflict with the idea that our notion of simplicity is universal, for what matters is not any of these individual objects, but the fact that taken together they span a wide space. And it seems likely that any other civilization would have as vast a repertoire of artifacts and natural objects and varieties of experience on which to draw as we do. So I believe that the skill of solving Bongard

problems lies very close to the core of "pure" intelligence, if there is such a thing. Therefore it is a good place to begin if one wants to investigate the ability to discover "intrinsic meaning" in patterns or messages. Unfortunately we have reproduced only a small selection of his stimulating collection. I hope that many readers will acquaint themselves with the entire collection, to be found in his book (see Bibliography).

Some of the problems of visual pattern recognition which we human beings seem to have completely "flattened" into our unconscious are quite amazing. They include:

recognition of faces (invariance of faces under age change, expression change, lighting change, distance change, angle change, etc.)

recognition of hiking trails in forests and mountains—somehow this has always impressed me as one of our most subtle acts of pattern recognition—and yet animals can do it, too

reading text without hesitation in hundreds if not thousands of different typefaces

Message-Passing Languages, Frames, and Symbols

One way that has been suggested for handling the complexities of pattern recognition and other challenges to AI programs is the so-called "actor" formalism of Carl Hewitt (similar to the language "Smalltalk", developed by Alan Kay and others), in which a program is written as a collection of interacting *actors,* which can pass elaborate *messages* back and forth among themselves. In a way, this resembles a heterarchical collection of procedures which can call each other. The major difference is that where procedures usually only pass a rather small number of arguments back and forth, the messages exchanged by actors can be arbitrarily long and complex.

Actors with the ability to exchange messages become somewhat autonomous agents—in fact, even like autonomous computers, with messages being somewhat like programs. Each actor can have its own idiosyncratic way of interpreting any given message; thus a message's meaning will depend on the actor it is intercepted by. This comes about by the actor having within it a piece of program which interprets messages; so there may be as many interpreters as there are actors. Of course, there may be many actors with identical interpreters; in fact, this could be a great advantage, just as it is extremely important in the cell to have a multitude of identical ribosomes floating throughout the cytoplasm, all of which will interpret a message—in this case, messenger RNA—in one and the same way.

It is interesting to think how one might merge the frame-notion with the actor-notion. Let us call a frame with the capability of generating and interpreting complex messages a *symbol:*

frame + actor = symbol

We now have reached the point where we are talking about ways of implementing those elusive *active symbols* of Chapters XI and XII; henceforth in this Chapter, "symbol" will have that meaning. By the way, don't feel dumb if you don't immediately see just how this synthesis is to be made. It is not clear, though it is certainly one of the most fascinating directions to go in AI. Furthermore, it is quite certain that even the best synthesis of these notions will turn out to have much less power than the actual symbols of human minds. In that sense, calling these frame-actor syntheses "symbols" is premature, but it is an optimistic way of looking at things.

Let us return to some issues connected with message passing. Should each message be directed specifically at a target symbol, or should it be thrown out into the grand void, much as mRNA is thrown out into the cytoplasm, to seek its ribosome? If messages have destinations, then each symbol must have an address, and messages for it should always be sent to that address. On the other hand, there could be one central receiving dock for messages, where a message would simply sit until it got picked up by some symbol that wanted it. This is a counterpart to General Delivery. Probably the best solution is to allow both types of message to exist; also to have provisions for different classes of urgency—special delivery, first class, second class, and so on. The whole postal system provides a rich source of ideas for message-passing languages, including such curios as self-addressed stamped envelopes (messages whose senders want answers quickly), parcel post (extremely long messages which can be sent some very slow way), and more. The telephone system will give you more inspiration when you run out of postal-system ideas.

Enzymes and AI

Another rich source of ideas for message passing—indeed, for information processing in general—is, of course, the cell. Some objects in the cell are quite comparable to actors—in particular, enzymes. Each enzyme's active site acts as a filter which only recognizes certain kinds of substrates (messages). Thus an enzyme has an "address", in effect. The enzyme is "programmed" (by virtue of its tertiary structure) to carry out certain operations upon that "message", and then to release it to the world again. Now in this way, when a message is passed from enzyme to enzyme along a chemical pathway, a lot can be accomplished. We have already described the elaborate kinds of feedback mechanisms which can take place in cells (either by inhibition or repression). These kinds of mechanisms show that complicated control of processes can arise through the kind of message passing that exists in the cell.

One of the most striking things about enzymes is how they sit around idly, waiting to be triggered by an incoming substrate. Then, when the substrate arrives, suddenly the enzyme springs into action, like a Venus's-flytrap. This kind of "hair-trigger" program has been used in AI, and goes by the name of *demon*. The important thing here is the idea of having many different "species" of triggerable subroutines just lying around waiting to