16

Learning by Analyzing Differences

In this chapter, you learn how it is possible to learn by analyzing the differences that appear in a sequence of observations. Along the way, you learn about *induction heuristics* that enable procedures to learn *class descriptions* from *positive and negative examples.* These induction heuristics make it possible, for example, to learn that an arch consists of one brick that must be supported by two others that must not touch each other.† Among the heuristics needed are the *require-link* and *forbid-link* heuristics, which enable learning about classes from *near-miss* examples that miss being class members for a small number of reasons.

You also learn about *felicity conditions*, which are the implied covenants between teachers and students that make learning possible.

By way of illustration, you learn about a simple learning program that expects a cooperative teacher to present carefully chosen examples, one after another. The procedure learns whatever it can from each example as the example is presented, and then forgets the example forever.

Once you have finished this chapter, you will have accumulated an armamentarium of induction heuristics, and you will have an understanding of the covenants that must hold between a teacher and a student. You will be able to use these ideas not only to build learning programs, but also to make yourself a more perceptive student and a more effective teacher.

---

†Properly speaking, one brick supported by two others is a lintel and pair of posts; for our purpose, however, that is just a nugatory detail.

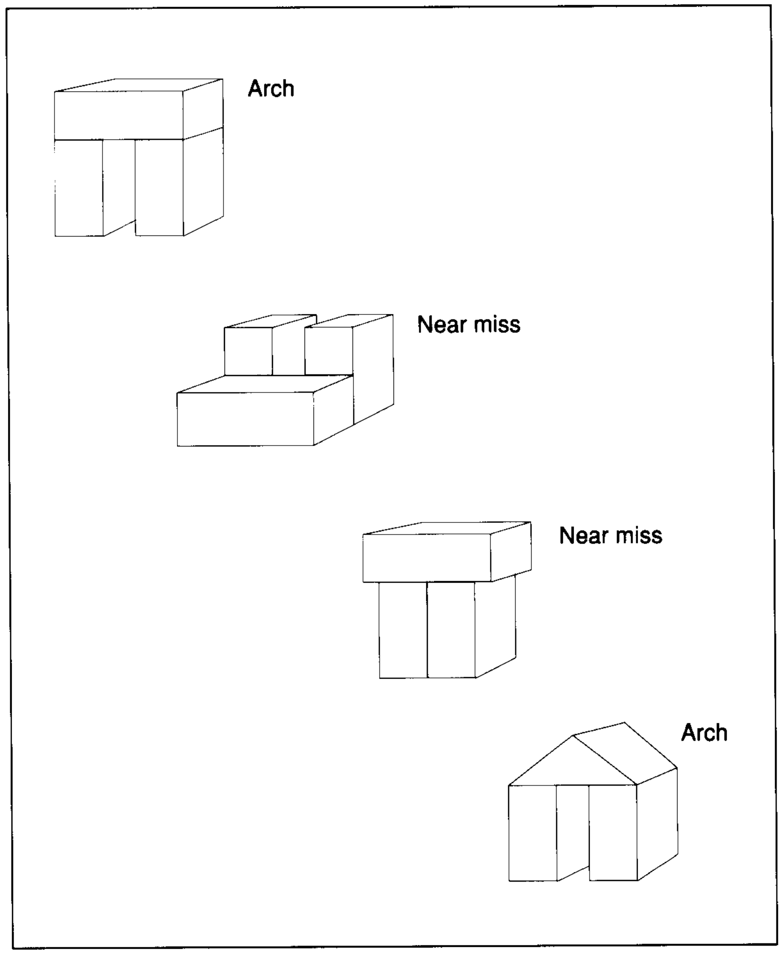


Figure 16.1 A sequence of positive examples and near-miss negative examples for learning about arches.

INDUCTION HEURISTICS

Induction occurs when you use particular examples to reach general conclusions. In this section, you learn about the **induction heuristics** used by a procedure, W, that learns about arches from the Arch and nonArch sequence shown in figure 16.1. You also learn about several powerful learning ideas that apply to human learning, as well as computer learning.

From the first example in figure 16.1, procedure W derives a general idea of what an arch is. In particular procedure W learns that an arch consists of two standing bricks that support a lying brick.

Each subsequent example drives home another point. In the second example, procedure W sees the same objects as before, but in a different configuration. Told that the pieces are not an arch, procedure W takes the example to be a **negative example**, and concludes that the support links must be an important aspect of the general arch concept. Note that the idea is conveyed by a single, well-chosen negative example, rather than by extended, tedious training exercises.

In the third example, the two standing bricks touch. Again, procedure W is told that the structure is not an arch. Nothing else is significantly different from the first arch in the example sequence. Evidently, the standing bricks must not touch if there is to be an arch. Procedure W makes progress once again by way of a good negative example.

A teacher may or may not claim the fourth example is an arch, according to personal taste. If it is given as an arch, then procedure W notes that having a brick on top is not essential. At the very least, either a brick or a wedge will do; procedure W may even guess that any simple parallelepiped is acceptable.

Responding to Near Misses Improves Models

To do its job, procedure W needs to start with a typical member of the class to be learned. From that example, procedure W constructs an **initial description**, as shown in figure 16.2(a). During learning, the initial description is augmented by information indicating which links are important. The augmented description is called the **evolving model**.

A **near miss** is a negative example that, for a small number of reasons, is not an instance of the class being taught. The description shown in figure 16.2(b) is not a description of an arch, but, because it is only a little different from the arch description in figure 16.2(a), it is a **near miss**. Its purpose is to teach the importance of the Support links.

Because the Support links are missing, comparing the two descriptions leads procedure W to the conclusion that arches require Support links. Thus, procedure W synthesizes the two descriptions into a new, refined description in which the Support links are replaced by the emphatic form, Must-support, as in figure 16.2(c). Used in this way, the near miss is said to supply information for the **require-link** heuristic. After procedure W uses the require-link heuristic, no group of blocks is identified as an arch unless Support links are in place.

Note that the two missing Support links associated with the near miss in figure 16.2(b) receive identical treatment. Generally, procedure W uses only one difference—either the only one or the one procedure W decides is most important. Sometimes, however, two or more differences are so similar, that they are handled as though they were just one difference. Procedure W's reaction is to suppose that the teacher intended the two differences to be handled in the same way. Thus, both Support links are replaced by Must-support.

The next comparison, the one between the evolving model in figure 16.3(a) and the near-miss in figure 16.3(b), also involves two similar differences because two new Touch links lie between the arch's sides. Now, however, the near miss fails to be an arch because links are present

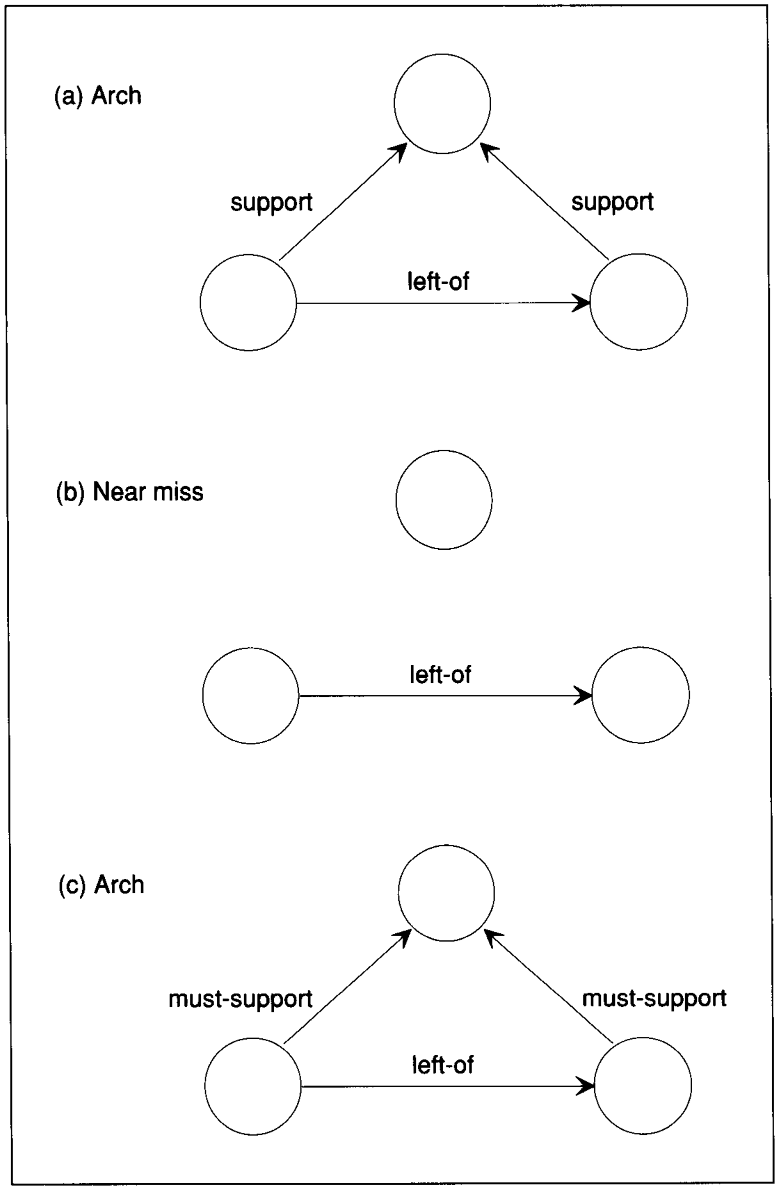
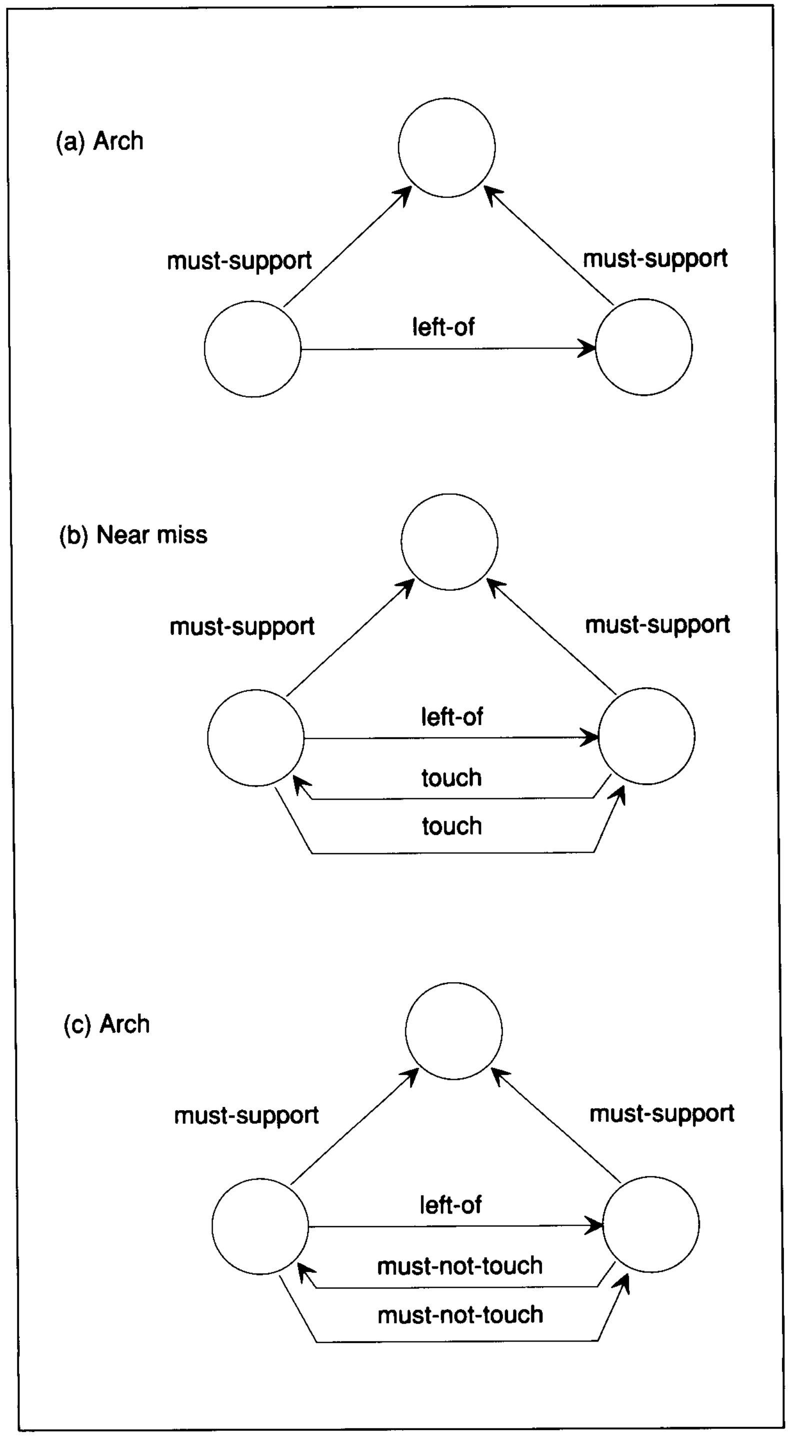


Figure 16.2 The require-link generalization rule. Compared with the Arch description in (a), the near-miss description in (b) lacks Support links. The conclusion is that Support links are essential, so the Support links in the Arch model are altered, indicating that they are required in all arches, as shown in (c). The Left-of link is shown to emphasize the need for evidence that is sufficient to establish the correct correspondence between the parts of the arch and the parts of the near miss. Many links have been omitted from the drawing to prevent distraction from those that matter.

rather than absent. Procedure W concludes that the new links should be forbidden, and converts each Touch link to the negative emphatic link, Must-not-touch, as shown in figure 16.3(c). In so doing, procedure W is said to make use of the **forbid-link** heuristic.

Note that the require-link and forbid-link heuristics work because the descriptions contain essential information and because description compar-

Figure 16.3 The forbid-link generalization rule. Compared with the Arch description in part (a), the near-miss description in part (b) differs because it has Touch links. The conclusion is that the Touch links must not be present, so Must-not-touch links are added to the Arch description as shown in part (c). Many links have been omitted from the drawing to prevent distraction from those that matter.



ison provides a way of zeroing in on the proper conclusions. These points bear elevation to principles:

You cannot learn if you cannot know.

▷ Good teachers help their students by being sure that their students acquire the necessary representations.

You cannot learn if you cannot isolate what is important.

▷ Good teachers help their students by providing not only positive examples, but also negative examples and near misses.

Responding to Examples Improves Models

So far, both near-miss examples *restrict* the model, limiting what can be an arch. Positive examples *relax* the model, expanding what can be an arch. Consider the situation of figure 16.4. Compared to the evolving model in figure 16.4(a), the example configuration in figure 16.4(b) has a wedge on top instead of a brick. If this is to be an arch, procedure W must make a change in the model that reflects a loosened constraint. At the very least, procedure W should cut the Is-a connection between the top of the arch and Brick, and replace that connection by a Must-be-a link to a more general class, as shown in figure 16.4(c). Procedure W is said to use the **climb-tree** heuristic.

Using the most specific common class is only one alternative, however. In the example, replacing Brick by Block represents a conservative position with respect to how much generalization procedure W should do, because bricks and wedges are also polyhedra, physical objects, and things. The new target for the top's Must-be-a link could be anything along the chain of Ako links, depending on how aggressive procedure W is to be.

Sometimes, however, there is no classification tree to climb. For example, if bricks and wedges were not known to be members of any common class, the climb-tree heuristic would not be of any use. In such a case, procedure W forms a new class, the Brick-or-wedge class, and joins the top part of the arch to this new class with Must-be-a, thereby using the **enlarge-set** heuristic.

If there are no objects other than bricks and wedges, however, procedure W gets rid of the Is-a link completely, and is said to use the **drop-link** heuristic.

Procedure W also uses the drop-link heuristic when a link in the evolving model is not in the example. If the initiating example has color information for some blocks and the other examples do not, procedure W ignores color, dropping all color references from the evolving model.

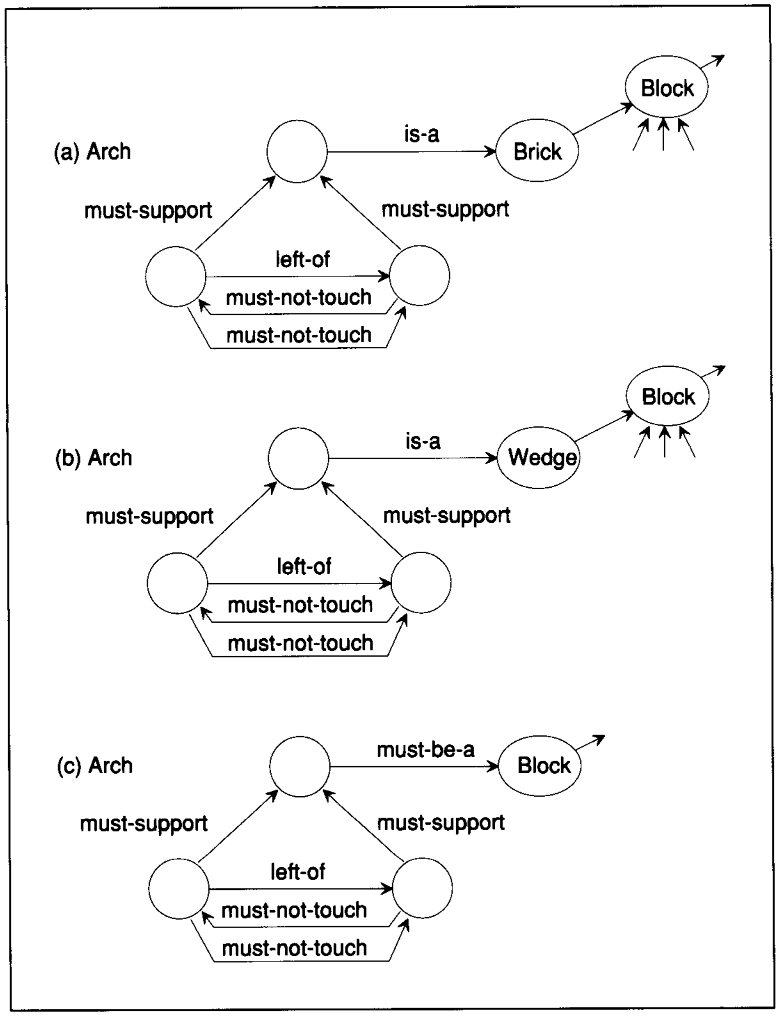


Figure 16.4 The climb-tree heuristic. The top of the Arch description in part (a) is a brick, while the corresponding object in the example description in part (b) is a wedge. Evidently, the difference does not matter. The Is-a link in the Arch description is changed to a Must-be-a link and redirected from Brick to Block, as shown in part (c), which is the most specific common generalization of Brick and Wedge. Many links have been omitted from the drawing to prevent distraction from those that matter.

Finally, procedure W uses another heuristic if a difference involves numbers. If one example exhibits a 10-centimeter brick, and another exhibits a 15-centimeter brick, then procedure W supposes that bricks of any length between 10 centimeters and 15 centimeters will do, thus using the **close-interval** heuristic.

Near-Miss Heuristics Specialize; Example Heuristics Generalize

Having seen how procedure W uses induction heuristics, it is time to summarize. Note that the near-miss heuristics, require link and forbid link, both specialize the model, making it more restrictive. The positive-example heuristics all generalize the model, making it more permissive.

- The **require-link** heuristic is used when an evolving model has a link in a place where a near miss does not. The model link is converted to a Must form.
- The **forbid-link** heuristic is used when a near miss has a link in a place where an evolving model does not. A Must-not form is installed in the evolving model.
- The **climb-tree** heuristic is used when an object in an evolving model corresponds to a different object in an example. Must-be-a links are routed to the most specific common class in the classification tree above the model object and the example object.
- The **enlarge-set** heuristic is used when an object in an evolving model corresponds to a different object in an example and the two objects are not related to each other through a classification tree. Must-be-a links are routed to a new class composed of the union of the objects' classes.
- The **drop-link** heuristic is used when the objects that are different in an evolving model and in an example form an exhaustive set. The drop-link heuristic is also used when an evolving model has a link that is not in the example. The link is dropped from the model.
- The **close-interval** heuristic is used when a number or interval in an evolving model corresponds to a number in an example. If the model uses a number, the number is replaced by an interval spanning the model's number and the example's number. If the model uses an interval, the interval is enlarged to reach the example's number.

Here then are the procedures that use these heuristics:

To use SPECIALIZE to make a model more restrictive,

- ▷ Match the evolving model to the example to establish correspondences among parts.
- ▷ Determine whether there is a single, most important difference between the evolving model and the near miss.
  - ▷ If there is a single, most important difference,
    - ▷ If the evolving model has a link that is not in the near miss, use the require-link heuristic.
    - ▷ If the near miss has a link that is not in the model, use the forbid-link heuristic.
  - ▷ Otherwise, ignore the example.

To use GENERALIZE to make a model more permissive,

- ▷ Match the evolving model to the example to establish correspondences among parts.
- ▷ For each difference, determine the difference type:
  - ▷ If a link points to a class in the evolving model different from the class to which the link points in the example,
    - ▷ If the classes are part of a classification tree, use the climb-tree heuristic.
    - ▷ If the classes form an exhaustive set, use the drop-link heuristic.
    - ▷ Otherwise, use the enlarge-set heuristic.
  - ▷ If a link is missing in the example, use the drop-link heuristic.
  - ▷ If the difference is that different numbers, or an interval and a number outside the interval, are involved, use the close-interval heuristic.
  - ▷ Otherwise, ignore the difference.

Note that SPECIALIZE does nothing if it cannot identify a most important difference. One way to identify the most important difference is to use a procedure that ranks all differences by difference type and by link type. Another way is described in Chapter 18.

Note also that both SPECIALIZE and GENERALIZE involve matching. For now, be assured that there are matching procedures that tie together the appropriate nodes. One such matching procedure is described in Chapter 17.

Learning Procedures Should Avoid Guesses

As described, procedure W uses examples supplied by a teacher in an order decided on by that teacher. The learner analyzes each example as it is given; the learner does not retain examples once they are analyzed:

To learn using procedure W,

- ▷ Let the description of the first example, which must be an example, be the initial description.
- ▷ For all subsequent examples,
  - ▷ If the example is a near miss, use procedure SPECIALIZE.
  - ▷ If the example is an example, use procedure GENERALIZE.

As given, procedure W never unlearns something it has learned once. In principle, procedure W could unlearn, but deciding exactly what to unlearn, such that nothing breaks, is hard. Consequently, it is better not to learn something that may have to be unlearned:

The **wait-and-see principle**:

▷ When there is doubt about what to do, do nothing.

It may seem excessively conservative to refuse to act because no act is absolutely safe. There is a point, however, where risk taking becomes foolhardiness. Honoring the wait-and-see principle, a learner is not condemned to eternal stupidity; the learner is merely expecting to encounter difficult situations again, later, when the learner is better prepared.

Procedure W honors the wait-and-see principle when it ignores negative examples for which it cannot identify a single or most-important difference.

Procedure W's teacher can help procedure W to avoid the need to ignore negative examples by ensuring that the negative examples are bona fide near misses. Alternatively, the teacher and the student can agree on how difference types should be ranked so that the difference that seems most important to the student actually is important from the perspective of the teacher. Learning-facilitating teacher–student agreements are called **felicity conditions**, especially if they are implied, rather than expressed.

Even with elaborate felicity conditions, however, there will be situations when a model is not consistent with an example, even though the model is basically correct. Penguins, for example, are birds, even though penguins cannot fly. In such situations, the way out is to honor another principle:

The **no-altering principle**:

▷ When an object or situation known to be an example fails to match a general model, create a special-case exception model.

Thus, the wait-and-see principle says to avoid building a model that will be wrong, and the no-altering principle says to avoid changing a model, even if it is wrong, again because fixing a general model in one way is likely to break it in another.

Learning Usually Must Be Done in Small Steps

Procedure W works because it exploits the knowledge it has, adding to that knowledge in small steps using new examples.

Skillful teachers know that people learn mostly in small steps, too. If there is too much to figure out, there is too much room for confusion and error:

Martin's law:

▷ You cannot learn anything unless you almost know it already.

IDENTIFICATION

In the previous section, you saw what procedure W can learn about objects. In this section, you learn how identification methods can use what has been learned by matching unknown objects to appropriate models.

Must Links and Must-Not Links Dominate Matching

One way to determine whether an unknown matches a model adequately is to see whether the unknown is compatible with the model's emphatic links. Any links with names prefixed by Must must be in the unknown; and links prefixed by Must-not must not be in the unknown.

More flexible match evaluation requires a procedure that can judge the degree of similarity between an unknown and a model. To implement such a procedure, you have to translate the abstract notion of *similarity* between an *unknown* and a *model*, $s(U, M)$, into a concrete measurement. One simple way of doing this translation, by a weighted counting of corresponding links, was described in Chapter 2 in connection with a geometric-analogy procedure. Note, however, that any counting scheme for combining evidence is limited, because all information is compressed into a singularly inexpressive number.

Models May Be Arranged in Lists or in Nets

Given a mechanism for matching an unknown with a model, the next issue is how to arrange the models for testing. We consider two of many possibilities: model lists and similarity nets.

Matching the unknown with the models in a model list was called the describe-and-match method in Chapter 2. It is a reasonable approach only if the number of models is small.

Another approach to arranging models is to use a **similarity net**. Imagine a set of models organized into a net in which the links connect model pairs that are very similar. Now suppose that an unknown object is to be identified. What should be done when the first comparison with a particular model in the net fails, as it ordinarily will? If the match does not fail by much—that is, if the unknown seems like the model in many respects—then surely other similar models should be tried next. These new similar models are precisely the ones connected by similarity links to the just-tried model.

APPLICATION

ARIEL Learns about Proteins

Increasingly, an impressive demonstration on a real problem is the sine qua non of successful research in artificial intelligence. Sometimes, the demonstration involves a program and a human expert working in tandem to do some task that neither could do independently.

In molecular biology, for example, the ARIEL program acts as a partner to human biologists, helping them to improve patterns that predict protein function. Before you learn how ARIEL works, you may find it helpful to review a little elementary protein biology.

First, **proteins** consist of one or a few long chains called **polypeptides**. Each link in a polypeptide chain is one of the 20 **amino acids**.

The **primary structure** of a protein is a specification of how the various amino acids are arranged in the polypeptide chain.

Here, for example, is a fragment a polypeptide produced by an AIDS virus:

-a-g-k-k-k-k-s-v-t-v-l-d-v-g-d-a-y-f-s-v-p-l-d-k-d-f-r-k-y-t-a-f-t-i-p-

The **secondary structure** is a specification of how various short segments in the chain fold up into small configurations, which have names such as alpha helix, beta strand, and beta turn. The example polypeptide fragment happens to contain several alpha helixes, beta strands, and beta turns.

The enzymatic activity of a protein is determined by its **tertiary structure**, a complete description of how it folds up in space—which ultimately depends, of course, on the primary structure of the polypeptides in the protein. As yet, however, no one knows how to use the primary structure to predict exactly how a protein will fold up. Nevertheless, a molecular biologist can predict that a protein will have certain functional properties by looking for characteristic patterns in the primary and secondary structure. One such characteristic pattern, refined with help from ARIEL, determines that a matching protein is likely to help duplicate **DNA molecules**, the ones that carry the genetic code:

DNA polymerase rule

If	There is a small amino acid followed by
	a beta strand followed by
	a hydrophobic amino acid followed by
	an aspartic acid followed by
	a hydrophobic amino acid followed by
	an aromatic amino acid followed by
	a beta strand followed by
	a beta strand
then	the protein is likely to be a DNA polymerase

Note that the pattern involves primary structure, in that it specifies amino-acid classes, as well as secondary structure, in that it specifies beta strands, which are identified, in turn, by an analysis of primary structure. The pattern happens to match the example primary-structure fragment from the AIDS virus.

ARIEL is able to improve such recognition patterns by artfully deploying induction heuristics, beam search, and parallel testing. The induction heuristics are used to perturb an existing pattern in the hope of producing a more reliable pattern. Typically, a human biologist asks ARIEL to try a variation on the climb-tree heuristic on a particular amino acid specified in the pattern, producing one new pattern for each of the many possible generalizations in the amino-acid classification tree.

None of these new patterns is perfect when tested on a database of about 50 examples and several hundred nonexamples. Instead, each pattern recognizes some fraction of the examples and rejects some fraction of the nonexamples. In the diagram that follows this text, you see that the seed pattern, indicated by the black dot, recognizes about 89 percent of the examples, and rejects about 80 percent of the nonexamples.

Also shown, by open circles, are the places occupied by new patterns derived from the seed pattern. Among these, the best are the ones that are closest to the upper-right corner, where all examples are recognized and all nonexamples are rejected.

Of course, it can take a lot of time to locate all the patterns on the recognize–reject diagram, for each induction heuristic can produce tens of possible patterns, each of which has to be tested on hundreds of examples and near-miss negative examples. In practice, given today's technology, this kind of pattern testing is best done on a parallel computer, such as a Connection Machine™.

Once all the patterns are evaluated, a few of the best are kept for further analysis, and the rest are rejected. Then the human biologist specifies another induction heuristic for ARIEL to try on the surviving patterns. Thus, ARIEL and the human biologist work their way through the space of derivative patterns using beam search.

After moving through a few layers of the beam search, ARIEL and the human biologist usually can do no more, halting with a family of patterns that are better than the seed pattern originally supplied by the human biologist.

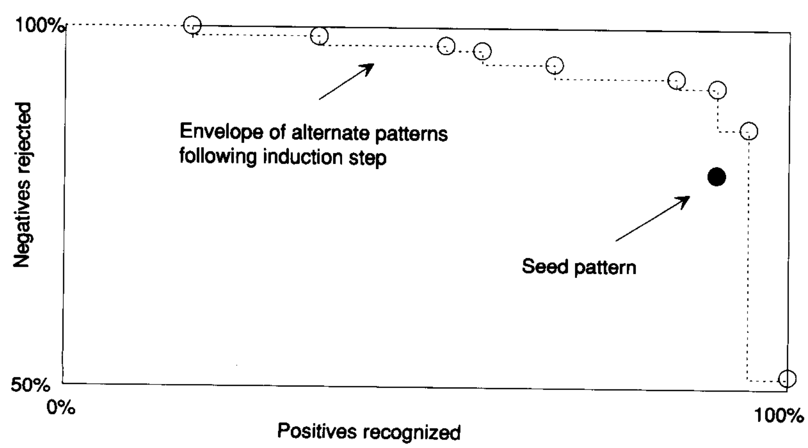
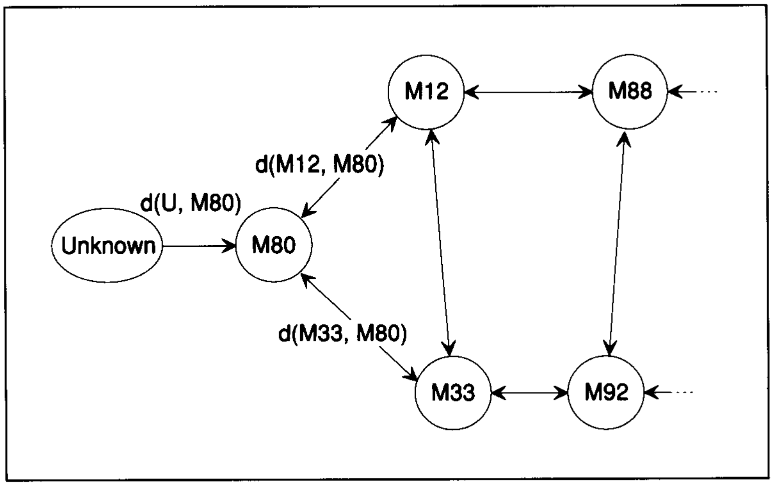


Figure 16.5 Identification using a similarity net. Progress from hypothesis to hypothesis is guided by comparison of difference descriptions. M80 is presumed to be the first hypothesis tried. M12 is next if the difference between M12 and M80 is much like the difference between the unknown, U, and M80.



In an obvious improvement, the similarity links between models can not only convey similarity, but also describe the difference, as stipulated in the following representation specification:

A **similarity net** is a representation
That is a semantic net
In which
▷ Nodes denote models.
▷ Links connect similar models.
▷ Links are tied to difference descriptions.

Figure 16.5 illustrates a similarity net. If an unknown differs from a test model in the same way that a neighbor of the test model differs from the test model, then that neighbor is a particularly good model to test next. Thus, attention moves not only to a family of likely candidates, but also to the particular member of that family that is most likely to lead toward success. Accordingly, the initial match is not so much a failure as it is an enlightening **knowledge probe**.

Finally, note that the procedure for moving through a similarity net is a hill-climbing procedure because movement is to the immediate neighbor that seems most likely to yield an improved match with the unknown.

SUMMARY

- One way to learn is to declare an initial example to be your initial model. Then, you improve the initial model incrementally using a series of examples.

- Some examples should be negative, near-miss examples. These examples enable you to zero in on just what it is about the evolving model that is essential, thus specializing the model.
- Require link and forbid link are specialization heuristics.
- Some examples should be positive examples. These enable you to generalize the model.
- Climb tree, enlarge set, drop link, and close interval are generalization heuristics.
- You cannot learn if you cannot isolate what is important. Good teachers help you by providing not only positive examples, but also negative examples and near misses.
- You should avoid guessing when you learn, because a bad guess may be hard to root out later on. One way to avoid guessing is to create a special-case exception when an object or idea known to be an example fails to match a general model.
- Martin's law says that you cannot learn anything unless you almost know it already.

BACKGROUND

The work described in this chapter is based on early work by Patrick H. Winston that introduced many induction heuristics, along with the near-miss idea [Winston 1970].

Subsequently, other researchers have offered improved procedures for using specializing and generalizing induction heuristics. In particular, most of the induction-heuristic names are adapted from the work of Ryszard S. Michalski [1980]. Michalski's INDUCE system includes several additional induction heuristics, many of which deal with chains of links and properties of groups.

The *no-altering principle* is my name for one of Marvin Minsky's **laws of noncompromise** discussed in *The Society of Mind*, Minsky's seminal book on artificial intelligence [1985].

Martin's law is an idea that was expressed by William A. Martin in Dubrovnik, Yugoslavia, in 1979.

The discussion of similarity nets is based on an idea by Winston [1970], subsequently developed by David L. Bailey [1986].

ARIEL is the work of Richard H. Lathrop [1990].