treatments of these issues are readily available [Brachman 1979; Hayes* 1977; Nilsson 1980].

It is particularly helpful to have a denotion link to keep perceptual structures separate from model structures. Then if mistakes are made by the vision automaton, a correction mechanism can either sever the denotation link completely or create a new denotation link between the correct model and image structures.

When dealing with many spatial relations, it is economical to recognize that many relations are "inverses" of each other. That is, LEFT-OF(x,y) is the "inverse" of RIGHT-OF(x,y);

$$\text{LEFT-OF}(x,y) \iff \text{RIGHT-OF}(y,x)$$

and also

$$\text{ADJACENT}(x,y) \iff \text{ADJACENT}(y,x)$$

Rather than double the number of these kinds of links, one can *normalize* them. That is, only one half of the inverse pair is used, and the interpreter infers the inverse relation when necessary.

Properties have a different semantics depending on the type of object that has the property. An "abstract" node can have a property that gives one aspect or refinement of the represented concept. A property of a "concrete" node presumably means an established and quantified property of the individual.

Partitions

Partitions are a powerful notion in networks. "Partition" is not used in the sense of a mathematical partition, but in the sense of a barrier. Since the network is a graph, it contains no intrinsic method of delimiting subgraphs of nodes and arcs. Such subgraphs are useful for two reasons:

1. *Syntactic.* It is useful to delimit that part of the network which represents the results of specific inferences.
2. *Semantic.* It is useful to delimit that part of the network which represents knowledge about specific objects. Partitions may then be used to impose a hierarchy upon an otherwise "flat" structure of nodes.

The simple way of representing partitions in a net is to create an additional node to represent the partition and introduce additional arcs from that node to every node or arc in the partition. Partitions allow the nodes and relations in them to be manipulated as a unit.

Notationally, it is cleaner to draw a labeled boundary enclosing the relevant nodes (or arcs). An example is shown by Fig. 10.12 where we consider two objects each made up of several parts with one object entirely left of the other. Rather than use a separate LEFT-OF relation for each of the parts, a single relation can be used between the two partitions. Any pair of parts (one from each object) should inherit the LEFT-OF relation. Partitions may be used to implement quantification in semantic net representations of predicate calculus [Hendrix 1975, 1979]. They may be used to implement frames (Section 10.3.1).

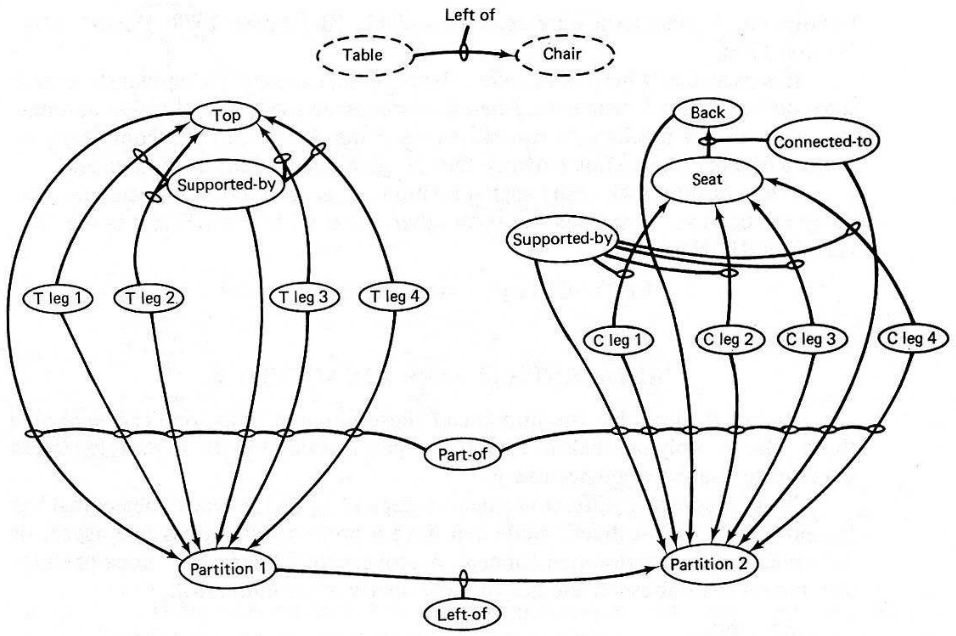


Fig. 10.12 The use of partitions. (a) Construction of a partition. (b) Two objects described by partitions.

Conversions

It is important to be able to transform from geometric (and logical) representations to propositional abstract representations and vice versa. For example, in Fig. 10.13 the problem is to find the exact location of a telephone on a previously located desk. In this case, propositional knowledge that telephones are usually on desktops, together with the desk top location and knowledge about the size of telephones, define a search area in the image.

Converting image data about a particular group of objects into relational form involves the inverse problem. The problem is to perform a level of abstraction to remove the specificity of the geometric knowledge and derive a relation that is appropriate in a larger context. For example, the following program fragment creates the relations ABOVE(A,B), where A and B are world objects.

```
Comment: assume a world coordinate system where Z is the positive vertical.

    Find $ZA_{\min}$ for $Z$ in $A$ and $ZB_{\max}$ for $Z$ in $B$.
    If $ZA_{\min} > ZB_{\max}$, then make ABOVE $(A,B)$ true.
```

Many other definitions of ABOVE, one of which compares centers of gravity, are possible. In most cases, the conversion from continuous geometric relations to discrete propositional relations involves more or less arbitrary conventions. To appreciate this further, consult Fig. 10.14 and try to determine in which of the cases

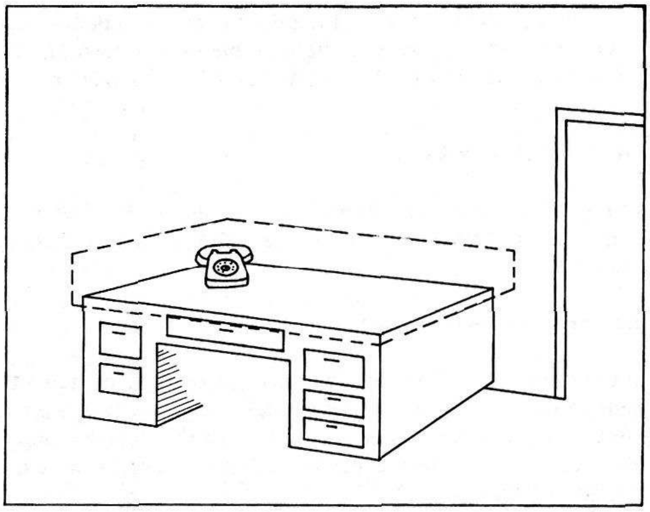


Fig. 10.13 Search area defined by relational bindings.

block *A* is LEFT-OF block *B*. Figure 10.14d shows a case where different answers are obtained depending on whether a two-dimensional or three-dimensional interpretation is used. Also, when relations are used to encode what is *usually* true of the world, it is often easy to construct a counterexample. Winston [Winston 1975] used

SUPPORTS (B,A) ABOVE (A,B)

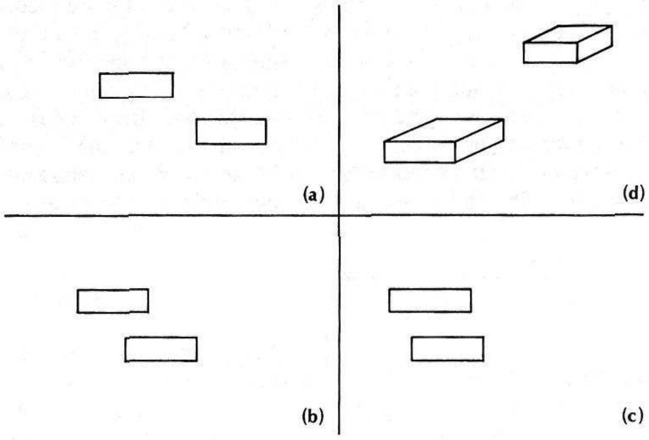


Fig. 10.14 Examples to demonstrate difficulties in encoding spatial relation LEFT-OF (see text).

which is contradicted by Fig. 10.15, given the previous definition of ABOVE.

One common way around these problems is to associate quantitative, "continuous" information with relations (section 10.3.2 and later examples).

10.3 SEMANTIC NET EXAMPLES

Examples of semantic nets abound throughout Part IV. Two more examples illustrate the power of the notions. The first example is described very generally, the second in detail.

10.3.1 Frame Implementations

Frame system theory [Minsky 1975] is a way of explaining our quick access to important aspects of a (perhaps perceptual) situation. It is a provocative and controversial idea, and the reader should consult the References for a full treatment. Implementationally, a frame may be realized by a partition; a frame is a "chunk" of related structure.

Associating related "chunks" of knowledge into manipulable units is a powerful and widespread idea in artificial intelligence [Hayes 1980; Hendrix 1979] as well as psychology. These chunks go by several names: units, frames, partitions, schemata, depictions, scripts, and so forth [Schank and Abelson 1977; Moore and Newell 1973; Roberts and Goldstein 1977; Hayes* 1977; Bobrow and Winograd 1977, 1979; Stefik 1979; Lehnert and Wilks 1979; Rumelhart et al. 1972].

Frames systems incorporate a theory of associative recall in which one selects frames from memory that are relevant to the situation in which one finds oneself. These frames include several kinds of information. Most important, frames have *slots* which contain details of the viewing situation. Frame theory dictates a strictly specific and prototypical structure for frames. That is, the number and type of slots for a particular type of frame are immutable and specified in advance. Further, frames represent specific prototype situations; many slots have default values; this is where expectations and prior knowledge come from. These default values may be disconfirmed by perceptual evidence; if they are, the frame can contain information about what actions to take to fill the slot. Some slots are to be filled in by investigation. Thus a frame is a set of expectations to be confirmed or disconfirmed

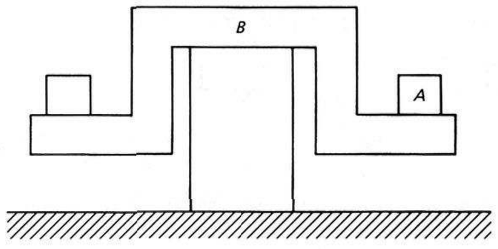


Fig. 10.15 A counterexample to SUPPORTS(B, A) => ABOVE(A, B).

and actions to pursue in various contingencies. One common action is to "bring in another frame."

The theory is that based on a partial match of a frame's defining slots, a frame can be "brought to mind." The retrieval is much like jumping to a conclusion based on partial evidence. Once the frame is proposed, its slots must be matched up with reality; thus we have the initial major hypothesis that the frame represents, which itself consists of a number of minor subhypotheses to be verified. A frame may have other frames in its slots, and so frames may be linked into "frame systems" that are themselves associatively related. (Consider, for example, the linked perceptual frames for being just outside a theater and for being just inside.) Transformations between frames correspond to the effects of relevant actions. Thus the hypotheses can suggest one another. "Thinking always begins with suggestive but imperfect plans and images; these are progressively replaced by better—but usually still imperfect—ideas" [Minsky 1975].

Frame theory is controversial and has its share of technical problems [Hinton 1977]. The most important of these are the following.

1. Multiple instances of concepts seem to call for copying frames (since the instances may have different slotfillers). Hence, one loses the economy of a preexisting structure.
2. Often, objects have variable numbers of components (wheels on a truck, runways in an airport). The natural representation seems to be a rule for constructing examples, not some specific example.
3. Default values seem inadequate to express legal ranges of slot-filling values or dependencies between their properties.
4. Property inheritance is an important capability that semantic nets can implement with "is a" or "element-of" hierarchies. However, such hierarchies raise the question of which frame to copy when a particular individual is being perceived. Should one copy the generic Mammal frame or the more specific Camel frame, for instance. Surely, it is redundant for the Camel frame to duplicate all the slots in the Mammal frame. Yet our perceptual task may call for a particular slot to be filled, and it is painful not to be able to tell where any particular slot resides.

Nevertheless, where these disadvantages can be circumvented or are irrelevant, frames are seeing increasing use. They are a natural organizing tool for complex data.

10.3.2 Location Networks

This section describes a system for associating geometric analogical data with a semantic net structure which is sometimes like a frame with special "evaluation" rules. The system is a geometrical inference mechanism that computes (or infers) two-dimensional search areas in an image [Russell 1979]. Such networks have found use in both aerial image applications [Brooks and Binford 1980; Nevatia and Price 1978] and medical image applications [Ballard et al. 1979].

The Network

A *location network* is a network representation of geometric point sets related by set-theoretic and geometric operations such as set intersection and union, distance calculation, and so forth. The operations correspond to restrictions on the location of objects in the world. These restrictions, or rules, are dictated by cultural or physical facts.

Each internal node of the location network contains a geometric *operation*, a list of *arguments* for the operation, and a *result* of the operation. For instance, a node might represent the set-theoretic union of two argument point sets, and the result would be a point set. Inference is performed by *evaluating* the net; evaluating all its operations to derive a point set for the top (root) operation.

The network thus has a hierarchy of ancestors and descendents imposed on it through the argument links. At the bottom of this hierarchy are *data nodes* which contain no operation or arguments, only geometric data. Each node is in one of three states: A node is *up-to-date* if the data attached to it are currently considered to be accurate. It is *out-of-date* if the data in it are known to be incomplete, inaccurate, or missing. It is *hypothesized* if its contents have been created by net evaluation but not verified in the image.

In a common application, the expected relative locations of features in a scene are encoded in a network, which thus models the expected structure of the image. The primitive set of geometric relations between objects is made up of four different types of operations.

1. *Directional* operations (left, reflect, north, up, down, and so on) specify a point set with the obvious locations and orientations to another.
2. *Area* operations (close-to, in-quadrilateral, in-circle and so on) create a point set with a non-directional relation to another.
3. *Set* operations (union, difference and intersection) perform the obvious set operations.
4. *Predicates* on areas allow point sets to be filtered out of consideration by measuring some characteristic of the data. For example, a predicate testing width, length, or area against some value restricts the size of sets to be those within a permissible range.

The location of the aeration tank in a sewage treatment plant provides a specific example. The aeration tank is often a rectangular tank surrounded on either end by circular sludge and sedimentation tanks (Fig. 10.16). As a general rule, sewage flows from the sedimentation tanks to aeration tanks and finally through to the sludge tanks. This design permits the use of the following types of restrictions on the location of the aeration tanks.

> *Rule 1*: "Aeration tanks are located somewhere close to both the sludge tanks and the sedimentation tanks."

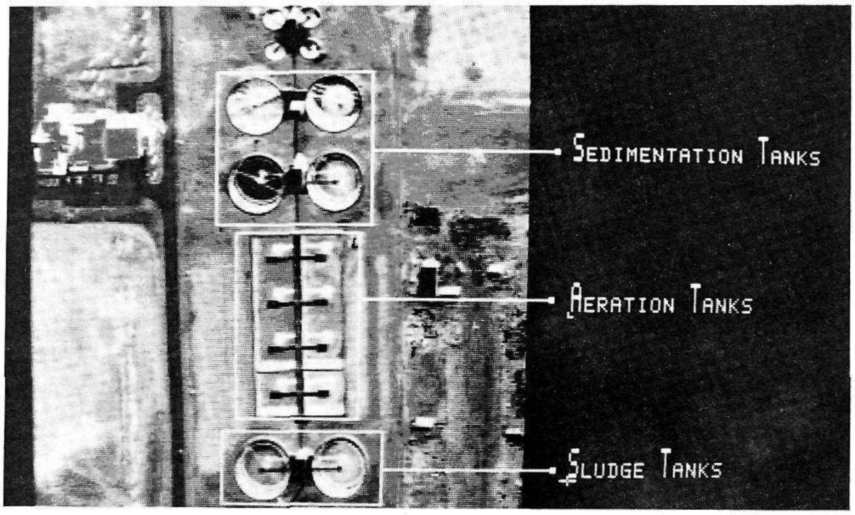


Fig. 10.16 Aerial image of a sewage plant.

The various tanks cannot occupy the same space, so:

> *Rule 2*: "Aeration tanks must not be too close to either the sludge or sedimentation tanks."

Rule 1 is translated to the following network relations.

CLOSE-TO(Union (LocSludgeTanks, LocSedTanks), Distance X)

Rule 2 is translated to

NOT-IN(Union (LocSludgeTanks,LocSedTanks), Distance Y)

The network describing the probable location of the aeration tanks embodies both of these rules. Rule 1 determines an area that is close to both groupings of tanks and Rule 2 eliminates a portion of that area. Thinking of the image as a point set, a set difference operation can remove the area given by Rule 2 from that specified by Rule 1. Figure 10.17 shows the final network that incorporates both rules.

Of course, there could be places where the aeration tanks might be located very far away or perhaps violate some other rule. It is important to note that, like the frames of Section 10.3.1, location networks give prototypical, likely locations for an object. They can work very well for stereotyped scenes, and might fail to perform in novel situations.

The Evaluation Mechanism

The network is interpreted (evaluated) by a program that works top-down in a recursive fashion, storing the partial results of each rule at the topmost node as-

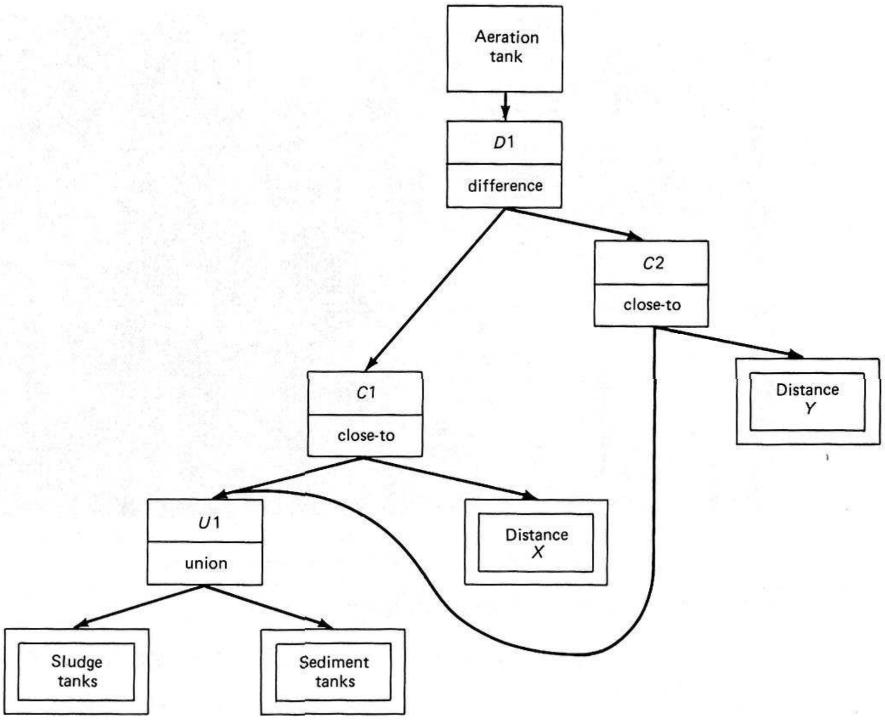


Fig. 10.17 Constraint network for aeration tank.

sociated with that rule (with a few exceptions). Evaluation starts with the root node. In most networks, this node is an operation node. An operation node is evaluated by first evaluating all its arguments, and then applying its operation to those results. Its own result is then available to the node of the network that called for its evaluation.

Data nodes may already contain results which might come from a map or from the previous application of vision operators. At some point in the course of the evaluation, the evaluator may reach a node that has already been evaluated and is marked up-to-date or hypothesized (such a node contains the results of evaluation below that point). The results of this node are returned and used exactly as if it were a data node. Out-of-date nodes cause the evaluation mechanism to execute a low-level procedure to establish the location of the feature. If the procedure is unable to establish the status of the object firmly within its resource limits, the status will remain out-of-date. At any time, out-of-date nodes may be processed without having to recompute any up-to-date nodes. A node marked hypothesized has a value, usually supplied by an inference process, and not verified by low-level image analysis. Hypothesized data may be used in inferences: the results of all inferences based on hypothesized data are marked hypothesized as well.

If a data node ever has its value changed (say, by an independent process that adds new information), all its ancestors are marked out-of-date. Thus the root node will indicate an out-of-date status, but only those nodes on the out-of-date path must be reevaluated to bring the network up to date. Figure 10.18 shows the operation of the aeration tank network of Fig. 10.17 on the input of Fig. 10.16. In this case the initial feature data were a single sludge tank and a single sedimentation tank. Suppose additional work is done to find the location of the remaining sludge and sediment tanks in the image. This causes a reevaluation of the network, and the new result more accurately reflects the actual location of the aeration tanks.

Properties of Location Networks

The location network provides a very general example of use of semantic nets in computer vision.

1. It serves as a data base of point sets and geometric information. The truth status of items in the network is explicitly maintained and depends on incoming information and operations performed on the net.
2. It is an expansion of a geometric expression into a tree, which makes the order of evaluation explicit and in which the partial results are kept for each geometric calculation. Thus it provides efficient updating when some but not all the partial results change in a reevaluation.
3. It provides a way to make geometrical inferences without losing track of the hypothetical nature of assumptions. The tree structure records dependencies among hypotheses and geometrical results, and so upon invalidation of a geometric hypothesis the consequences (here, what other nodes have their values affected) are explicit. The record of dependencies solves a major problem in automated inference systems.
4. It reflects implicit universal quantification. The network claims to represent true relations whose explicit arguments must be filled in as the network is "instantiated" with real data.
5. It has a "flat" semantics. There are no element-of hierarchies or partitions.
6. The concept of "individual" is flexible. A point set can contain multiple disconnected components corresponding to different world objects. In set operations, such an assemblage acts like an explicit set union of the components. An "individual" in the network may thus correspond to multiple individual point (sub)sets in the world.
7. The network allows use of partial knowledge. A set-theoretic semantics of existence and location allows modeling of an unknown location by the set-theoretic universe (the possible location is totally unconstrained). If something is known not to exist in a particular image, its "location" is the null set. Generally, a location is a point set.
8. The set-theoretic semantics allows useful punning on set union and the OR operation, and set intersection and the AND operation. If a dock is on the

shoreline AND near a town, the search for docks need only be carried out in the intersection of the locations.

10.4 CONTROL ISSUES IN COMPLEX VISION SYSTEMS

Computer vision involves the control of large, complex information-processing tasks. Intelligent biological systems solve this control problem. They seem to have complicated control strategies, allowing dynamic allocation of computational resources, parallelism, interrupt-driven shifts of attention, and incremental behavior modification. This section explores different strategies for controlling the complex information processing involved in vision. Appendix 2 contains specific

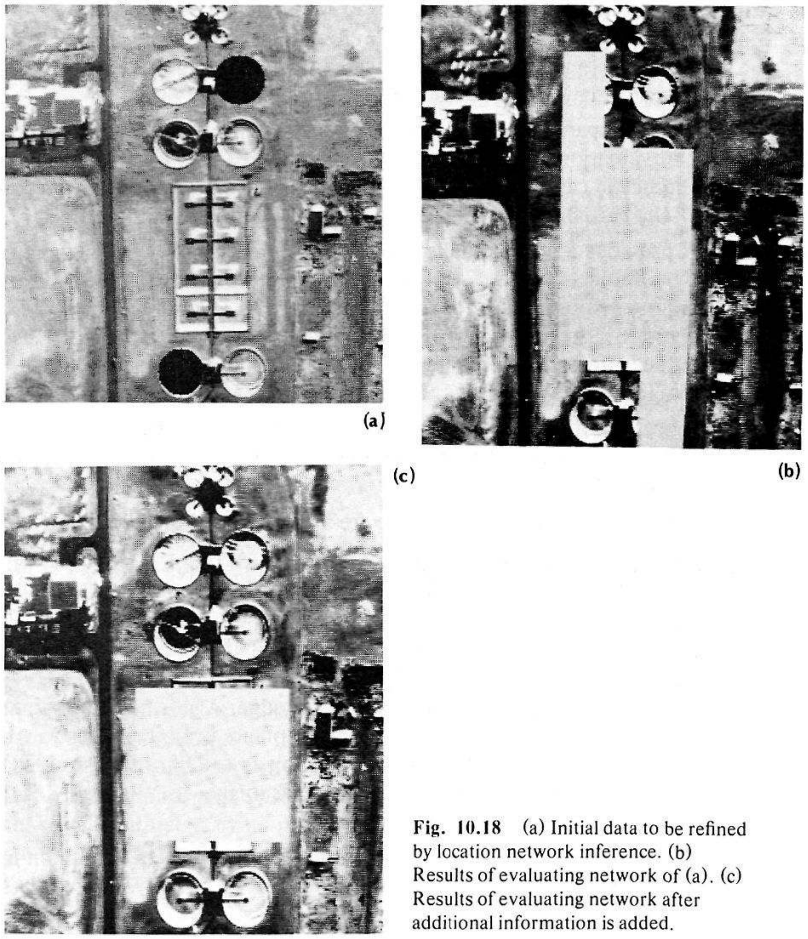


Fig. 10.18 (a) Initial data to be refined by location network inference. (b) Results of evaluating network of (a). (c) Results of evaluating network after additional information is added.

techniques and programming language constructs that have proven to be useful tools in implementing control strategies for artificial intelligence and computer vision.

10.4.1 Parallel and Serial Computation

In *parallel computation*, several computations are done at the same time. For example, different parts of an image may be processed simultaneously. One issue in parallel processing is synchronization: Is the computation such that the different parts can be done at different rates, or must they be kept in step with each other? Usually, the answer is that synchronization is important. Another issue in parallel processing is its implementation. Animal vision systems have the architecture to do parallel processing, whereas most computer systems are serial (although developing computer technologies may allow the practical realization of some parallel processing). On a serial computer parallelism must be simulated—this is not always straightforward.

In *serial computation*, operations are performed sequentially in time whether or not they depend on one another. The implied sequential control mechanism is more closely matched to a (traditional) serial computer than is a parallel mechanism. Sequential algorithms must be stingy with their resources. This fact has had many effects in computer vision. It has led to mechanisms for efficient data access, such as multiple-resolution representations. It has also led some to emphasize cognitive alternatives for low-level visual processing, in the hope that the massive parallel computations performed in biological vision systems could be circumvented. However, this trend is reversing; cheaper computation and more pervasive parallel hardware should increase the commitment of resources to low-level computations. Parallel and serial control mechanisms have both appeared in algorithms in earlier chapters. It seems clear that many low-level operations (correlation, intrinsic image computations) can be implemented with parallel algorithms. High-level operations, such as "planning" (Chapter 13) have inherently serial components. In general, in the low levels of visual processing control is predominately parallel, whereas at the more abstract levels some useful computations are necessarily serial in nature.

10.4.2 Hierarchical and Heterarchical Control

Visual control strategies dictate the flow of information and activity through the representational layers. What triggers processing: a low level input like a color patch on the retina, or a high level expectation (say, expecting to see a red car)? Different emphasis on these extremes is a basic control issue. The two extremes may be characterized as follows.

1. Image data driven. Here the control proceeds from the construction of the generalized image to segmented structures and finally to descriptions. This is also called *bottom-up* control.

2. Internal model driven. Here high-level models in the knowledge base generate expectations or predictions of geometric, segment, or generalized image structure in the input. Image understanding is the verification of these predictions. This is also called *top-down* control.

Top-down and bottom-up control are distinguished not by what they do but rather by the order in which they do it and how much of it they do. Both approaches can utilize all the basic representations—intrinsic images, features, geometric structures, and propositional representations—but the processing within these representations is done in different orders.

The division of control strategies into top-down and bottom-up is a rather simplistic one. There is evidence that attentional mechanisms may be some of the most complicated brain functions that human beings have [Geschwind 1980]. The different representational subsystems in a complex vision system influence each other in sophisticated and intricate ways; whether control flows "up" or "down" is only a broad characterization of local influence in the (loosely ordered) layers of the system.

The term "bottom-up" was originally applied to parsing algorithms for formal languages that worked their way up the parse tree, assembling the input into structures as they did so. "Top-down" parsers, on the other hand, notionally started at the top of the parse tree and worked downward, effectively generating expectations or predictions about the input based on the possibilities allowed by the grammar; the verification of these predictions confirmed a particular parsing.

These two paradigms are still basic in artificial intelligence, and provide powerful analogies and methods for reasoning about and performing many information-processing tasks. The bottom-up paradigm is comparable in spirit with "forward chaining," which derives further consequences from established results. The top-down paradigm is reflected in "backward chaining," which breaks problems up into subproblems to be solved.

These control organizations can be used not only "tactically" to accomplish specific tasks, but they can dictate the whole "strategy" of the vision campaign. We shall discover that in their pure forms the extreme strategies (top-down and bottom-up) appear inadequate to explain or implement vision. More flexible organizations which incorporate both top-down and bottom-up components seem more suited to a broad spectrum of ambitious vision tasks.

Bottom-Up Control

The general outline for bottom-up vision processing is:

1. *PREPROCESS.* Convert raw data into more usable intrinsic forms, to be interpreted by next level. This processing is automatic and domain-independent.
2. *SEGMENT.* Find visually meaningful image objects perhaps corresponding to world objects or their parts. This process is often but not always broken up into (a) the extraction of meaningful visual primitives, such as lines or regions of homogeneous composition (based on their local characteristics); and (b) the agglomeration of local image features into larger segments.

3. *UNDERSTAND.* Relate the image objects to the domain from which the image arose. For instance, identify or classify the objects. As a step in this process, or indeed as the final step in the computer vision program, the image objects and the relations between them may be described.

In pure bottom-up organization each stage yields data for the next. The progression from raw data to interpreted scene may actually proceed in many steps; the different representations at each step allow us to separate the process into the main steps mentioned above.

Bottom-up control is practical if potentially useful "domain-independent" processing is cheap. It is also practical if the input data are accurate and yield reliable and unambiguous information for the higher-level visual processes. For example, the binary images that result from careful illumination engineering and input thresholding can often be processed quite reliably and quickly in a bottom-up mode. If the data are less reliable, bottom-up styles may still work if they make only tolerably few errors on each pass.

Top-Down Control

A bottom-up, hierarchical model of perception is at first glance appealing on neurological and computational grounds, and has influenced much classical philosophical thought and psychological theory. The "classical" explanation of perception has relatively recently been augmented by a more cognition-based one involving (for instance) interaction of knowledge and expectations with the perceptual process in a more top-down manner [Neisser 1967; Bartlett 1932]. A similar evolution of the control of computer vision processing has accounted for the augmentation of the pure "pattern recognition" paradigm with more "cognitive" paradigms. The evidence seems overwhelming that there are vision processes which do not "run bottom-up," and it is one of the major themes of this book that internal models, goals, and cognitive processes must play major roles in computer vision [Gregory 1970; Buckhout 1974; Gombrich 1972]. Of course, there must be a substantial component of biological vision systems which can perform in a noncognitive mode.

There are probably no versions of top-down organization for computer vision that are as pure as the bottom-up ones. The model to keep in mind in top-down perception is that of goal-directed processing. A high-level goal spawns subgoals which are attacked, again perhaps yielding sub-subgoals, and so on, until the goals are simple enough to solve directly. A common top-down technique is "hypothesize-and-verify"; here an internal modeling process makes predictions about the way objects will act and appear. Perception becomes the verifying of predictions or hypotheses that flow from the model, and the updating of the model based on such probes into the perceptual environment [Bolles 1977]. Of course, our goal-driven processes may be interrupted and resources diverted to respond to the interrupt (as when movement in the visual periphery causes us to look toward the moving object). Normally, however, the hypothesis verification paradigm requires relatively little information from the lower levels and in principle it can control the low-level computations.

The desire to circumvent unnecessary low-level processing in computer vision is understandable. Our low-level vision system performs prodigious amounts of information processing in several cascaded parallel layers. With serial computation technology, it is very expensive to duplicate the power of our low-level visual system. Current technological developments are pointing toward making parallel, low-level processing feasible and thus lowering this price. In the past, however, the price has been so heavy that much research has been devoted to avoiding it, often by using domain knowledge to drive a more or less top-down perception paradigm. Thus there are two reasons to use a top-down control mechanism. First, it seems to be something that human beings do and to be of interest in its own right. Second, it seems to offer a chance to accomplish visual tasks without impractical expenditure of resources.

Mixed Top-Down and Bottom-Up Control

In actual computer vision practice, a judicious mixture of data-driven analysis and model-driven prediction often seems to perform better than either style in isolation. This meld of control styles can sometimes be implemented in a complex hierarchy with a simple pass-oriented control structure. An example of mixed organization is provided by a tumor-detection program which locates small nodular tumors in chest radiographs [Ballard 1976]. The data-driven component is needed because it is not known precisely where nodular tumors may be expected in the input radiograph; there is no effective model-driven location-hypothesizing scheme. On the other hand, a distinctly top-down flavor arises from the exploitation of what little is known about lung tumor location (they are found in lungs) and tumor size. The variable-resolution method using pyramids, in which data are examined in increasingly fine detail, also seems top-down. In the example, work done at 1/16 resolution in a consolidated array guides further processing at 1/4 resolution. Only when small windows of the input array are isolated for attention are they considered at full resolution.

The process proceeds in three passes which move from less to greater detail (Fig. 10.19), zooming in on interesting areas of image, and ultimately finding objects of interest (nodules). Two later passes (not shown) "understand" the nodules by classifying them as "ghosts," tumors or nontumors. Within pass II, there is a distinct data-driven (bottom-up) organization, but passes I and III have a model-directed (top-down) philosophy.

This example shows that a relatively simple, pass-oriented control structure may implement a mixture of top-down and bottom-up components which focus attention efficiently and make the computation practical. It also shows a few places where the ordering of steps is not inherently sequential, but could logically proceed in parallel. Two examples are the overlapping of high-pass filtering of pass II with pass I, and parallel exploration of candidate nodule sites in pass III.

Heterarchical Control

The word "heterarchy" seems to be due to McCulloch, who used it to describe the nonhierarchical (i.e., not partially ordered in rank) nature of neural responses implied by their connectivity in the brain. It was used in the early 1970s to characterize a particular style of nonhierarchical, non-pass-structured control

	PREPROCESS	SEGMENT	CONTROL
Pass 0			
(Digitize radiograph)	The digitizer has a hardware attechment which produces the optical density.		
Pass I			
(Find lung boundaries)	In 64 X 56 consolidated array, apply gradient at proper resolution	In 64 X 56 array, find rough lung outline; in 256 X 224 array, refine lung outline	TOP–DOWN
Pass II			
(Find candidate nodule sites and large tumors)	In 256 X 224 array, apply high-pass filter to enhance edges, then inside lung boundaries; apply gradient at proper resolution	In 256 X 224 array use gradient-directed, circular Hough method to find candidate sites; also detect large tumors	BOTTOM–UP
Pass III			
(Find nodule boundaries)	From 1024 X 896 array, extract 64 X 64 window about each candidate nodule site, then in window apply high-pass filter for edge enhancement; then apply gradient at proper resolution	In 64 X 64 full-resolution, pre-processed window, apply dynamic programming technique to find accurate nodule boundaries	TOP–DOWN

Fig. 10.19 A hierarchical tumor-detection algorithm. Technical details of the methods are found elsewhere in this volume. The processing proceeds in passes from top to bottom, and within each pass from left to right. The processing exhibits both top-down and bottom-up characteristics.

organization. Rather than a hierarchical structure (such as the military), one should imagine a community of cooperating and competing experts. They may be organized in their effort by a single executive, by a universal set of rules governing their behavior, or by an a priori system of ranking. If one can think of a task as consisting of many smaller subtasks, each requiring some expertise, and not necessarily performed globally in a fixed order, then the task could be suitable for heterarchical-like control structure.

The idea is to use, at any given time, the expert who can help *most* toward final task solution. The expert may be the most efficient, or reliable, or may give the most information; it is selected because according to some criterion its subtask is the best thing to do at that time. The criteria for selection are wide and varied, and several ideas have been tried. the experts may compute their own relevance, and the decision made on the basis of those individual local evaluations (as in PANDEMONIUM [Selfridge 1959]). They may be assigned a priori immutable rank, so that the highest-ranking expert that is applicable is always run (as in [Shirai 1975; Ambler et al. 1975]). A combination of empirically predetermined and dynamically situation-driven information can be combined to decide which expert applies.

The actual control structure of heterarchical programming can be quite simple; it can be a single iterative loop in which the best action to take is chosen, applied, and interpreted (Fig. 10.20).

10.4.3 Belief Maintenance and Goal Achievement

Belief maintenance and goal achievement are high-level processes that imply differing control styles. The former is concerned with maintaining a current state, the latter with a set of future states. Belief maintenance is an ongoing activity which can ensure that perceptions fit together in a coherent way. Goal achievement is the integration of vision into goal-directed activities such as searching for objects and navigation. There may be "unconscious" use of goal-seeking techniques (e.g., eye-movement control).

Belief Maintenance

An organism is presented with a rich visual input to interpret. Typically, it all makes sense: chairs and tables are supported by floors, objects have expected shapes and colors, objects appear to flow past as the organism moves, nearer objects obscure farther ones, and so on. However, every now and then something

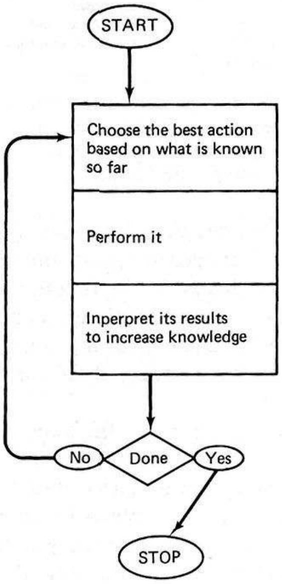

START
Choose the best action based on what is known so far
Perform it
Inperpret its results to increase knowledge
No
Done
Yes
STOP

Fig. 10.20 A main executive control loop for heterarchical vision.

enters the visual field that does not meet expectations. An unfamiliar object in a familiar environment or a sudden movement in the visual periphery can be "surprises" that do not fit in with our existing beliefs and thus have to be reckoned with.

It is sometimes impossible to ignore movements in our visual periphery, but if we are preoccupied it is easily possible to stay unconscious of small changes in our environment. How is it possible to notice some things and not others? The belief maintenance mechanism seems to be resource-limited. A certain amount of "computing resource" is allocated for the job. With this resource, only a limited amount of checking can be done. Checks to be made are ranked (somehow—responses to events in the periphery are like reflexes, or high-priority hard-wired interrupts) and those that cannot be done within the resource limit are omitted. Changes in our beliefs are often initiated in a *bottom-up* way, through unexpected inputs.

A second characteristic of belief maintenance is the almost total absence of sequential, simulation-based or "symbolic" planning or problem-solving activity. Our beliefs are "in the present"; manipulation of hypothetical worlds is not belief maintenance. "Truth maintenance" schemes have been discussed in various contexts [Doyle 1979; Stallman and Sussman 1977].

We conjecture that constraint-satisfaction (relaxation) mechanisms (Chapters 3, 7, and 12) are computationally suited to maintaining belief structures. They can operate in parallel, they seek to minimize inconsistency, they can tolerate "noise" in either input or axioms. Relaxation techniques are usually applied to low-level visual input where locally noisy parameters are combined into globally consistent intrinsic images. Chapter 12 is concerned with inference, in which constraint relaxation is applied to higher-level entities.

Characteristics of Goal Achievement

Goal achievement involves two related activities: planning and acting. Planning is a simulation of the world designed to generate a plan. A *plan* is a sequence of actions that, if carried out, should achieve a goal. *Actions* are the primitives that can modify the world. The motivation for planning is survival. By being able to simulate the effects of various actions, a human being is able to avoid dangerous situations. In an analogous fashion, planning can help machines with vision. For example, a Mars rover can plan its route so as to avoid steep inclines where it might topple over. The incline measurement is made by processing visual input. Since planning involves a sequence of actions, each of which if carried out could potentially change the world, and since planning does not involve actually making those changes, the difficult task of the planner is to keep track of all the different world states that could result from different action sequences.

Vision can clearly serve as an important information-gathering step in planning actions. Can planning techniques be of use directly to the vision process? Clearly so in "skilled vision," such as photointerpretation. Also, planning is a useful computational mechanism that need not be accompanied by conscious, cognitive behavior.

These inductive conclusions leading to the formation of our sense perceptions certainly do lack the purifying and scrutinizing work of conscious thinking. Nevertheless, in my opinion, by their particular nature they may be classed as *conclusions*, inductive conclusions unconsciously formed. [Helmholtz 1925]

The character of computations in goal achievement is related to the inference mechanisms studied in Chapter 11, only planning is distinguished by being dynamic through time. Inference (Chapter 12) is concerned with the knowledge base and deducing relations that logically follow from it. The primitives are *propositions.* In planning (Chapter 13) the primitives are *actions*, which are inherently more complex than propositions. Also, planning need not be a purely deductive mechanism; instead it can be integrated with visual "acting", or the interpretation of visual input. Often, a long deductive sequence may be obviated by using direct visual inspection. This raises a crucial point: Given the existence of plans, how does one choose between them? The solution is to have a method of scoring plans based on some measure of their effectiveness.

EXERCISES

10.1 (a) Diagram some networks for a simple dial telephone, at various levels of detail and with various complexities of relations.

(b) Now include in your network dial and pushbutton types.

(c) Embed the telephone frame into an office frame, describing where the telephone should be found.

10.2 Is a LISP vision program an analogical or propositional representation of knowledge?

10.3 Write a semantic net for the concept "leg," and use it to model human beings, tables, and spiders. Represent the fact "all tables have four legs." Can your "leg" model be shared between tables and spiders? Shared within spiders?

REFERENCES

AMBLER, A. P., H. G. BARROW, C. M. BROWN, R. M. BURSTALL, and R. J. POPPLESTONE. "A versatile system for computer controlled assembly." *Artificial Intelligence* 6, 2, 1975, 129–156.

ANDERSON, J. R. and G. H. BOWER. *Human Associative Memory.* New York: V. H. Winston & Sons, 1973.

BALLARD, D. H. *Hierarchic Recognition of Tumors in Chest Radiographs.* Basel: Birkhauser-Verlag (ISR-16), 1976.

BALLARD, D. H. "Model-directed detection of ribs in chest radiographs." TR11, Computer Science Dept., Univ. Rochester, March 1978.

BALLARD, D. H., U. SHANI, and R. B. SCHUDY. "Anatomical models for medical images." *Proc.*, 3rd COMPSAC, November 1979, 565-570.

BARROW, H. G. and J. M. TENENBAUM. "Computational vision." *Proc. IEEE* 69, 5, May 1981, 572-595.

BARTLETT, F. C. *Remembering: A Study in Experimental and Social Psychology.* Cambridge: Cambridge University Press, 1932.

BOBROW, D. G. and T. WINOGRAD. "An overview of KRL-0, a knowledge representation language." *Cognitive Science 1*, 1, 1977, 3–46.

BOBROW, D. G. and T. WINOGRAD. "KRL: another perspective." *Cognitive Science 3*, 1, 1979, 29–42.

BOLLES, R. C. "Verification vision for programmable assembly." *Proc.*, 5th IJCAI, August 1977, 569–575.

BRACHMAN, R. J. "What's in a concept? Structural foundations for semantic networks." Report 3433, Bolt, Beranek and Newman, October 1976.

BRACHMAN, R. J. "On the epistemological status of semantic networks." *In Associative Networks: Representation and Use of Knowledge by Computers*, N.V. Findler (Ed.). New York: Academic Press, 1979, 3–50.

BROOKS, R. A. and T. O. BINFORD. "Representing and reasoning about specified scenes." *Proc.*, DARPA IU Workshop, April 1980, 95–103.

BUCKHOUT, R. "Eyewitness testimony." *Scientific American*, December 1974, 23–31.

DOYLE, J. "A truth maintenance system." *Artificial Intelligence 12*, 3, 1979.

FAHLMAN, S. E. "A planning system for robot construction tasks." *Artificial Intelligence 5*, 1, 1974, 1–49.

FINDLER, N. V. (Ed.). *Associative Networks: Representation and Use of Knowledge by Computers.* New York: Academic Press, 1979.

FODOR, J. D., J. A. FODOR, and M. F. GARRETT. "The psychological unreality of semantic representations." *Linguistic Inquiry 4*, 1975, 515–531.

FREUDER, E. C. "A computer system for visual recognition using active knowledge." Ph.D. dissertation, MIT, 1975.

FUNT, B. V. "WHISPER: a problem-solving system utilizing diagrams." *Proc.*, 5th IJCAI, August 1977, 459–464.

GARVEY, J. D. "Perceptual strategies for purposive vision." Technical Note 117, AI Center, SRI International, 1976.

GELERNTER, H. "Realization of a geometry-theorem proving machine." In *Computers and Thought*, E. Feigenbaum and J. Feldman (Eds.). New York: McGraw-Hill, 1963.

GESCHWIND, N. "Neurological knowledge and complex behaviors." *Cognitive Science 4*, 2, April 1980, 185–193.

GOMBRICH, E. H. *Art and Illusion.* Princeton, NJ: Princeton University Press, 1972.

GREGORY, R. L. *The Intelligent Eye.* New York: McGraw-Hill, 1970.

HAYES, Patrick J. "The logic of frames." *In The Frame Reader.* Berlin: DeGruyter, 1980.

HAYES*, Philip J. "Some association-based techniques for lexical disambiguation by machine." Ph.D. dissertation, École polytechnique fédérale de Lausanne, 1977; also TR25, Computer Science Dept., Univ. Rochester, June 1977.

HELMHOLTZ, H. von. *Treatise on Physiological Optics* (translated by J. P. T. Sauthall). New York: Dover Publications, 1925.

HENDRIX, G. G. "Expanding the utility of semantic networks through partitions." *Proc.*, 4th IJCAI, September 1975, 115–121.

HENDRIX, G. G. "Encoding knowledge in partitioned networks." In *Associative Networks: Representation and Use of Knowledge by Computers*, N.V. Findler (Ed.). New York: Academic Press, 1979, 51–92.

HEWITT, C. "Description and theoretical analysis (using schemata) of PLANNER" (Ph.D. dissertation). AI-TR-258, AI Lab, MIT, 1972.

HINTON, G. E. "Relaxation and its role in vision." Ph.D. dissertation, Univ. Edinburgh, December 1977.

JOHNSON-LAIRD, P. N. "Mental models in cognitive science." *Cognitive Science 4*, 1, January–March 1980, 71–115.

KOSSLYN, S. M. and J. R. POMERANTZ. "Imagery, propositions and the form of internal representations." *Cognitive Psychology 9*, 1977, 52–76.

KOSSLYN, S. M. and S. P. SCHWARTZ. "A simulation of visual imagery." *Cognitive Science 1*, 3, July 1977, 265–295.

KOSSLYN, S. M. and S. P. SCHWARTZ. "Visual images as spatial representations in active memory." In *CVS*, 1978.

LEHNERT, W. and Y. WILKS. "A critical perspective on KRL." *Cognitive Science 3*, 1, 1979, 1–28.

LOZANO-PEREZ, T. and M. A. WESLEY. "An algorithm for planning collision-free paths among polyhedral obstacles." *Comm. ACM 22*, 10, October 1979, 560–570.

McDERMOTT, D. "Artificial intelligence meets natural stupidity." *SIGART Newsletter 57*, April 1976, 4–9.

MINSKY, M. L. "A framework for representing knowledge." In *PCV*, 1975.

MOORE J. and A. NEWELL. "How can MERLIN understand?" In *Knowledge and Cognition*, L. Gregg (Ed.). Hillsdale, NJ: Lawrence Erlbaum Assoc., 1973.

MOORE, R. C. "Reasoning about knowledge and action." Techical Note 191, AI Center, SRI International, 1979.

NEISSER, U. *Cognitive Psychology*. New York: Appleton-Century-Crofts, 1967.

NEVATIA, R. and K.E. PRICE. "Locating structures in aerial images." USCIPI Report 800, Image Processing Institute, Univ. Southern California, March 1978, 41–58.

NILSSON, N. J. *Problem-Solving Methods in Artificial Intelligence*. New York: McGraw-Hill, 1971.

NILSSON, N. J. *Principles of Artificial Intelligence*. Palo Alto, CA: Tioga, 1980.

PALMER, S. E. "Visual perception and world knowledge: notes on a model of sensory-cognitive interaction." In *Explorations in Cognition*, D.A. Norman, D.E. Rumelhart, and the LNR Research Group (Eds.). San Francisco: W.H. Freeman, 1975.

PYLYSHYN, Z. W. "What the mind's eye tells the mind's brain; a critique of mental imagery." *Psychological Bulletin 80*, 1973, 1–24.

QUILLIAN, M. R. "Semantic memory." In *Semantic Information Processing*, M. Minsky (Ed.). Cambridge, MA: MIT Press, 1968.

ROBERTS, R. B. and I. P. GOLDSTEIN. "The FRL primer." AI Memo 408, AI Lab, MIT, 1977.

RUMELHART, D. E., P. H. LINDSAY, and D. A. NORMAN. "A process model for long-term memory." In *Organization of Memory*, E. Tulving and J. Donaldson (Eds.). New York: Academic Press, 1972.

RUSSELL, D. M. "Where do I look now?" *Proc.*, PRIP, August 1979, 175–183.

SCHANK, R. C. *Conceptual Information Processing*. Amsterdam: North-Holland, 1975.

SCHANK, R. C. and R. P. ABELSON. *Scripts, Plans, Goals and Understanding*. Hillsdale, NJ: Lawrence Erlbaum Assoc., 1977.

SCHUBERT, L. K. "Extending the expressive power of semantic networks." *Artificial Intelligence 7*, 2, 1976, 163–198.

SELFRIDGE, O. "Pandemonium, a paradigm for learning." In *Proc.*, Symp. on the Mechanisation of Thought Processes, National Physical Laboratory, Teddington, England, 1959.

SHEPARD, R. N. "The mental image." *American Psychologist 33*, 1978, 125–137.

SHIRAI, Y. "Analyzing intensity arrays using knowledge about scenes." In *PCV*, 1975.

SLOMAN, A. "Interactions between philosophy and artificial intelligence: the role of intuition and non-logical reasoning in intelligence." *Artificial Intelligence 2*, 3/4, 1971, 209–225.

STALLMAN, R. M. and G. J. SUSSMAN. "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis." *Artificial Intelligence 9*, 2, 1977, 135–196.

STEFIK, M. "An examination of a frame-structured representation system." *Proc.*, 6th IJCAI, August 1979, 845–852.

SUSSMAN, G. J. and D. MCDERMOTT. "Why conniving is better than planning." AI Memo 255A, AI Lab, MIT, 1972.

WALTZ, D. and L. BOGGESS. "Visual analog representations for natural language understanding." *Proc.*, 6th IJCAI, August 1979, 226–234.

WINOGRAD, T. "Extended inference modes in reasoning by computer systems." *Proc.*, Conf. on Inductive Logic, Oxford Univ., August 1978.

WINOGRAD, T. "Frame representations and the declarative/procedural controversy." In *Representation and Understanding*, D. G. Bobrow and A. M. Collins (Eds.). New York: Academic Press, 1975, 185–210.

WINSTON, P. H. "Learning structural descriptions from examples." In *PCV*, 1975.

WINSTON, P. H. *Artificial Intelligence.* Reading, MA: Addison-Wesley, 1977.

WOODS, W. A. "What's in a link? Foundations for semantic networks." In *Representation and Understanding*, D. G. Bobrow and A. M. Collins (Eds.). New York: Academic Press, 1975.

Matching 11

11.1 ASPECTS OF MATCHING

11.1.1 Interpretation: Construction, Matching, and Labeling

Figure 10.1 shows a vision system organization in which there are several representations for visual entities. A complex vision system will at any time have several coexisting representations for visual inputs and other knowledge. Perception is the process of integrating the visual input with the preexisting representations, for whatever purpose. Recognition, belief maintenance, goalseeking, or building complex descriptions—all involve forming or finding relations between internal representations. These correspondences *match* (''model,'' ''re-represent,'' ''abstract,'' ''label'') entities at one level with those at another level.

Ultimately, matching ''establishes an interpretation'' of input data, where an interpretation is the correspondence between models represented in a computer and the external world of phenomena and objects. To do this, matching associates different representations, hence establishing a connection between their interpretations in the world. Figure 11.1 illustrates this point. Matching associates TOKNODE, a token for a linear geometric structure derived from image segmentation efforts with a model token NODE101 for a particular road. The token TOKNODE has the interpretation of an image entity; NODE101 has the interpretation of a particular road.

One way to relate representations is to *construct* one from the other. An example is the construction of an intrinsic image from raw visual input. Bottom-up construction in a complex visual system is for reliably useful, domain-independent, goal-independent processing steps. Such steps rely only on ''compiled-in'' (''hard-wired,'' ''innate'') knowledge supplied by the designer of the system. Matching becomes more important as the needed processing becomes more diverse and idiosyncratic to an individual's experience, goals, and

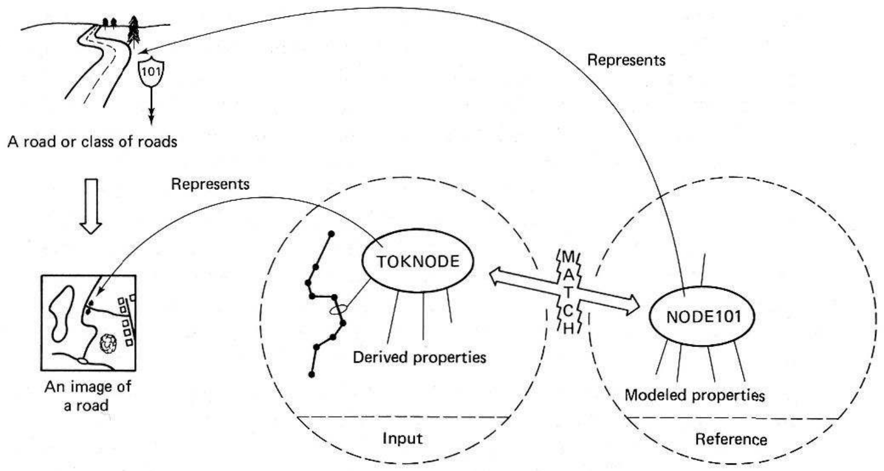


Fig. 11.1 Matching and interpretation.

knowledge. Thus as processing moves from "early" to "late," control shifts from bottom-up toward top-down, and existing knowledge begins to dominate perception.

This chapter deals with some aspects of matching, in which two already existing representations are put into correspondence. When the two representations are similar (both are images or relational structures, say), "matching" can be used in its familiar sense. When the representations are different (one image and one geometric structure, say), we use "matching" in an extended sense; perhaps "fitting" would be better. This second sort of matching usually has a top-down or expectation-driven flavor; a representation is being related to a preexisting one.

As a final extension to the meaning of matching, matching might include the process of checking a structure with a set of rules describing structural legality, consistency, or likelihood. In this sense a scene can be matched against rules to see if it is nonsense or to assign an interpretation. One such interpretation process (called *labeling*) assigns consistent or optimally likely interpretations (labels) at one level to entities of another level. Labeling is like matching a given structure with a possibly infinite set of acceptable structures to find the best fit. However, we (fairly arbitrarily) treat labeling in Chapter 12 as extended inference rather than here as extended matching.

11.1.2 Matching Iconic, Geometric, and Relational Structures

Chapter 3 presented various correlation techniques for matching *iconic* (image-like) structures with each other. The bulk of this chapter, starting in Section 11.2, deals with matching *relational* (semantic net) structures. Another important sort of matching between two dissimilar representations fits data to parameterized models (usually geometric). This kind of matching is an important part of computer vision. A typical example is shown in Fig. 11.2. A preexisting representation (here a straight line) is to be used to interpret a set of input data. The line that best "explains" the data is (by definition) the line of "best fit." Notice that the decision to use a line (rather than a cubic, or a piecewise linear template) is made at a higher level. Given the model, the fitting or matching means determining the *parameters* of the model that tailor it into a useful abstraction of the data.

Sometimes there is no parameterized mathematical model to fit, but rather a given geometric structure, such as a piecewise linear curve representing a shoreline in a map which is to be matched to a piece of shoreline in an image, or to another piecewise linear structure derived from such a shoreline. These geometric matching problems are not traditional mathematical applications, but they are similar in that the best match is defined as the one minimizing a measure of disagreement.

Often, the computational solutions to such geometric matching problems exhibit considerable ingenuity. For example, the shore-matching example above may proceed by finding that position for the segment of shore to be matched that minimizes some function (perhaps the square) of a distance metric (perhaps Euclidean) between input points on the iconic image shoreline and the nearest point on the reference geometric map shoreline. To compute the smallest distance between an arbitrary point and a piecewise linear point set is not a trivial task, and this calculation may have to be performed often to find the best match. The computation may be reduced to a simple table lookup by precomputing the metric in a "chamfer array," that contains the metric of disagreement for any point around the geometric reference shoreline [Barrow et al. 1978]. The array may be computed efficiently by symmetric axis transform techniques (Chapter 8) that "grow" the linear structure outward in contours of equal disagreement (distance) until a value has been computed for each point of the chamfer array.

Parameter optimization techniques can relate geometrical structures to lower-level representations and to each other through the use of a merit function measuring how well the relations match. The models are described by a vector of parameters $\mathbf{a} = (a_1, \ldots, a_n)$. The merit function M must rate each set of those parameters in terms of a real number. For example, M could be a function of both a, the parameters, and $f(x)$, the image. The problem is to find a such that

$$M(\mathbf{a}, f(\mathbf{x}))$$

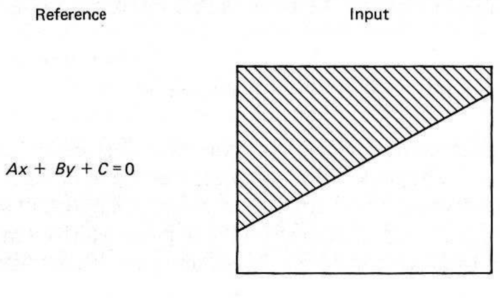


Fig. 11.2 Matching or fitting a straight line model to data.

is maximized. Note that if **a** were some form of template function rather than a vector of parameters, the problem statement would encompass the iconic correlation techniques just covered. There is a vast literature on optimization techniques and we cannot do more than provide a cursory discussion of a few cases with examples.

Formally, the different techniques have to do with the form of the merit function M. A fundamental result from calculus is that if M is sufficiently well behaved (i.e., has continuous derivatives), then a condition for a local maximum (or minimum) is that

$$M_{a_j} = \frac{\partial M}{\partial a_j} = 0 \qquad \text{for } j = 1, \ldots, n \tag{11.1}$$

This condition can be exploited in many different ways.

- Sometimes Eqs. (11.1) are sufficiently simple so that the a can be determined analytically, as in the least squares fitting, described in Appendix 1.
- An approximate solution a^0 can be iteratively adjusted by moving in the gradient direction or direction of maximum improvement:

  $$a_j^k = a_j^{k-1} + cM_{a_j} \tag{11.2}$$

  where c is a constant. This is the most elementary of several kinds of *gradient (hill-climbing) techniques*. Here the gradient is defined with respect to M and does not mean edge strength.
- If the partial derivatives are expensive to calculate, the coefficients can be perturbed (either randomly or in a structured way) and the perturbations kept if they improve M:

  (1) $\mathbf{a}' := \mathbf{a} + \Delta\mathbf{a}$

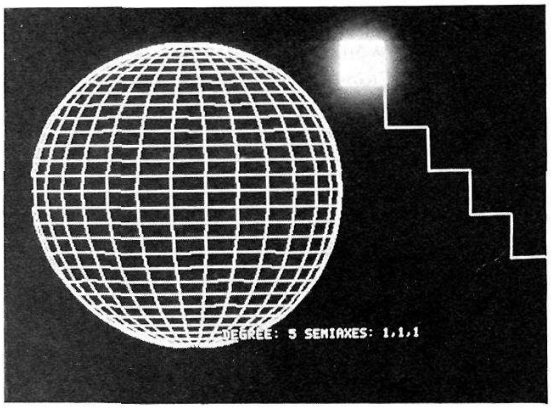
  (2) $\mathbf{a} := \mathbf{a}'$ if $M(\mathbf{a}') > M(\mathbf{a})$

A program to fit three-dimensional image data with shapes described by spherical harmonics used these techniques [Schudy and Ballard 1978]. The details of the spherical harmonics shape representation appear in Chapter 9. The fitting proceeded by the third method above. A nominal expected shape was matched to boundaries in image data. If a subsequent perturbation in one of its parameters results in an improvement in fit it was kept; otherwise, a different perturbation was made. Figure 11.3 shows this fitting process for a cross section of the shape.

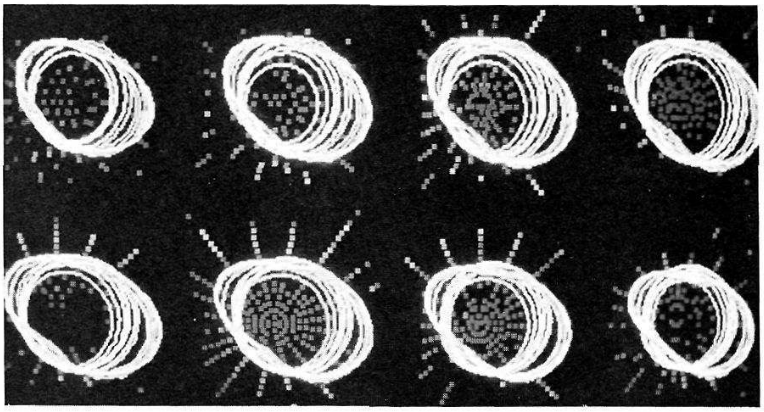
Though parameter optimization is an important aspect of matching, we shall not pursue it further here in view of the extensive literature on the subject.

11.2 GRAPH-THEORETIC ALGORITHMS

The remainder of this chapter deals with methods of matching relational structures. Chapter 10 showed how to represent a relational structure containing n-ary relations as a graph with labeled arcs. Recall that the labels can have values from a



(a)



(b)

(c)

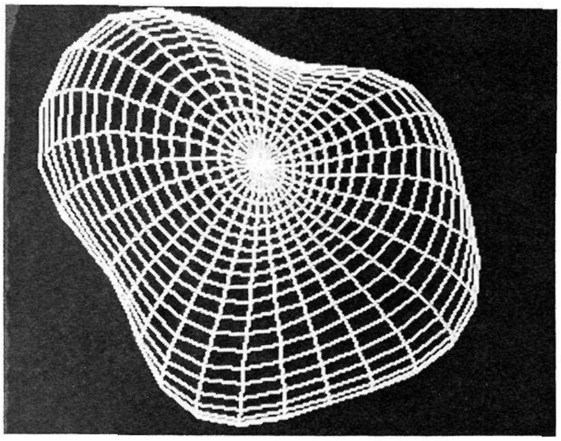


Fig. 11.3 An example of matching as parameter optimization. (a) Initial parameter set (displayed at left as three-dimensional surface (see Fig. 9.8) (b) Fitting process: iteratively adjust *a* based on M (see text). (c) Final parameter set yields this three-dimensional surface. (*See color inserts.*)

continuum, and that labeled arcs could be replaced by nodes to yield a directed graph with labeled nodes.

Depending on the attributes of the relational structure and of the correspondence desired, the definition of a match may be more or less elegant. It is always possible to translate powerful representations such as labeled graphs or n-ary relations into computational representations which are amenable to formal treatment (such as undirected graphs). However, when graph algorithms are to be implemented with computer data structures, the freedom and power of programming languages often tempts the implementer away from pure graph theory. He can replace elegant (but occasionally restrictive and impractical) graph-theoretic concepts and operations with arbitrarily complex data structures and algorithms.

One example is the "graph isomorphism" problem, a very pure version of relational structure matching. In it, all graph nodes and arcs are unlabeled, and graphs match if there is a 1:1 and onto correspondence between the arcs and nodes of the two graphs. The lack of expressive power in these graphs and the requirement that a match be "perfect" limits the usefulness of this pure model of matching in the context of noisy input and imprecise reference structures. In practice, graph nodes may have properties with continuous ranges of values, and an arbitrarily complex algorithm determines whether nodes or arcs match. The algorithm may even access information outside the graphs themselves, as long as it returns the answer "match" or "no match." Generalizing the graph-theoretic notions in this way can obscure issues of their efficiency, power, and properties; one must steer a course between the "elegant and unusable" and the "general and uncontrollable." This section introduces some "pure" graph-theoretic algorithms that form the basis for techniques in Sections 11.3 and 11.4.

11.2.1 The Algorithms

The following are several definitions of matching between graphs [Harary 1969; Berge 1976].

- *Graph isomorphism.* Given two graphs (V_1, E_1) and (V_2, E_2), find a 1:1 and onto mapping (an isomorphism) f between V_1 and V_2 such that for $v_1, v_2 \in V_1, V_2, f(v_1) = v_2$ and for each edge of E_1 connecting any pair of nodes v_1 and $v'_1 \in V_1$, there is an edge of E_2 connecting $f(v_1)$ and $f(v_1')$.
- *Subgraph isomorphism.* Find isomorphisms between a graph $(V_{1,} E_1)$ and *subgraphs* of another graph $(V_{2,} E_2)$. This is computationally harder than isomorphism because one does not know in advance which subsets of V_2 are involved in isomorphisms.
- *"Double" subgraph isomorphisms.* Find all isomorphisms between *subgraphs* of a graph $(V_{1,} E_1)$ and *subgraphs* of another graph $(V_{2,} E_2)$. This sounds harder than the subgraph isomorphism problem, but is equivalent.
- A match may not conform to strict rules of correspondence between arcs and nodes (some nodes and arcs may be "unimportant"). Such a matching criterion may well be implemented as a "computational" (impure) version of one of the pure graph isomorphisms.

Figure 11.4 shows examples of these kinds of matches.

One algorithm for finding graph isomorphism [Corneil and Gotlieb 1970] is based on the idea of separately putting each graph into a canonical form, from which isomorphism may easily be determined. For directed graphs (i.e., nonsymmetric relations) a backtrack search algorithm [Berztiss 1973] works on both graphs at once.

Two solutions to the subgraph isomorphism problem appear in [Ullman 1976]: The first is a simple enumerative search of the tree of possible matches between nodes. The second is more interesting; in it a process of "parallel-iterative" refinement is applied at each stage of the search. This process is a way of rejecting node pairs from the isomorphism and of propagating the effects of such rejections; one rejected match can lead to more matches being rejected. When the iteration converges (i.e., when no more matches can be rejected at the current stage), another step in the tree search is performed (one more matching pair is hypothesized). This mixing of parallel-iterative processes with tree search is useful in a variety of applications (Section 11.4.4, Chapter 12).

"Double" subgraph isomorphism is easily reduced to subgraph isomorphism via another well-known graph problem, the "clique problem." A *clique* of size *N* is a totally connected subgraph of size *N* (each node is connected to every other node in the clique by an arc). Finding isomorphisms between subgraphs of a graph A and subgraphs of a graph B is accomplished by forming an *association graph G* from the graphs A and B and finding cliques in G (for details, see Section 11.3.3). Clique

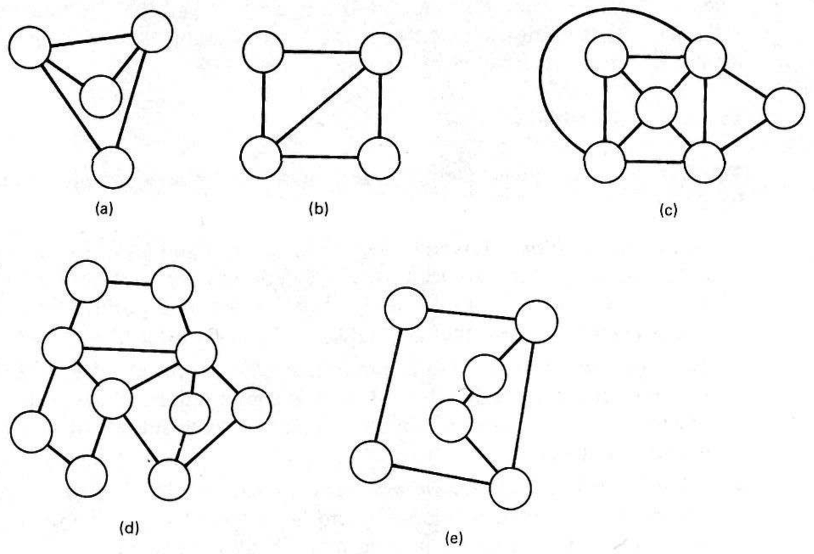


Fig. 11.4 Isomorphisms and matches. The graph (a) has an isomorphism with (b), various subgraph isomorphisms with (c), and several "double" subgraph isomorphisms with (d). Several partial matches with (e) (and also (b), (c), and (d)), depending on which missing or extra nodes are ignored.

finding may be done with a subgraph isomorphism algorithm; hence the reduction. Several other clique-finding algorithms exist [Ambler et al. 1975; Knodel 1968; Bron and Kerbosch 1973; Osteen and Tou 1973].

11.2.2 Complexity

It is of some practical importance to be aware of the computational complexity of the matching algorithms proposed here; they may take surprising amounts of computer time. There are many accessible treatments of computational complexity of graph-theoretic algorithms [Reingold et al. 1977; Aho, Hopcroft and Ullman 1974]. Theoretical results usually describe worst-case or average time complexity. The state of knowledge in graph algorithms is still improving; some interesting worst-case bounds have not been established.

A "hard" combinatorial problem is one that takes time (in a usual model of computation based on a serial computer) proportional to an exponential function of the length of the input. "Polynomial-time" solutions are desirable because they do not grow as fast with the size of the problem. The time to find all the cliques of a graph is in the worst case inherently exponential in the size of the input graphs, because the output is an exponential number of graphs. Both the single subgraph isomorphism problem and the "clique problem" (does there exist a clique of size k?) are *NP-complete*; all known deterministic algorithms run (in the worst case) in time exponential in the length of the description of the graphs involved (which must specify the nodes and arcs). Not only this, but if either of these problems (or a host of other NP complete problems) could be solved deterministically in time polynomially related to the length of the input, it could be used to solve all the other NP problems in polynomial time.

Graph isomorphism, both directed and undirected, is at this writing in a netherworld (along with many other combinatorial problems). No polynomial-time deterministic algorithms are known to exist, but the relation of these problems to each other is not as clear-cut as it is between the NP-complete problem. In particular, finding a polynomial-time deterministic solution to one of them would not necessarily indicate anything about how to solve the other problems deterministically in polynomial time. These problems are not mutually reducible. Certain restrictions on the graphs, for instance that they are planar (can be arranged with their nodes in a plane and with no arcs crossing), can make graph isomorphism an "easy" (polynomial-time) problem.

The average-case complexity is often of more practical interest than the worst case. Typically, such a measure is impossible to determine analytically and must be approximated through simulation. For instance, one algorithm to find isomorphisms of randomly generated graphs yields an average time that seems not exponential, but proportional to N^3, with N the number of nodes in the graph [Ullman 1976]. Another algorithm seems to run in average time proportional to N^2 [Corneil and Gotlieb 1970].

All the graph problems of this section are in NP. That is, a *non*deterministic algorithm can solve them in polynomial time. There are various ways of visualizing nondeterministic algorithms; one is that the algorithm makes certain significant "good guesses" from a range of possibilities (such as correctly guessing which subset of nodes from graph *B* are isomorphic with graph *A* and then only having to worry about the arcs). Another way is to imagine *parallel* computation; in the clique problem, for example, imagine multiple machines running in parallel, each with a different subset of nodes from the input graph. If any machine discovers a totally connected subset, it has, of course, discovered a clique. Checking whether *N* nodes are all pairwise connected is at most a polynomial-time problem, so all the machines will terminate in polynomial time, either with success or not. Several interesting processes can be implemented with parallel computations. Ullman's algorithm uses a refinement procedure which may run in parallel between stages of his tree search, and which he explains how to implement in parallel hardware [Ullman 1976].

11.3 IMPLEMENTING GRAPH-THEORETIC ALGORITHMS

11.3.1 Matching Metrics

Matching involves *quantifiable similarities.* A match is not merely a correspondence, but a correspondence that has been quantified according to its "goodness." This measure of goodness is the *matching metric.* Similarity measures for correlation matching are lumped together as one number. In relational matching they must take into account a relational, structured form of data [Shapiro and Haralick 1979].

Most of the structural matching metrics may be explained with the physical analogy of "templates and springs" [Fischler and Elschlager 1973]. Imagine that the reference data comprise a structure on a transparent rubber sheet. The matching process moves this sheet over the input data structure, distorting the sheet so as to get the best match. The final goodness of fit depends on the individual matches between elements of the input and reference data, and on the amount of work it takes to distort the sheet. The continuous deformation process is a pretty abstraction which most matching algorithms do not implement. A computationally more tractable form of the idea is to consider the model as a set of rigid "templates" connected by "springs" (see Fig. 11.5). The templates are connected by "springs" whose "tension" is also a function of the relations between elements. A spring function can be arbitrarily complex and nonlinear; for example the "tension" in the spring can attain very high or infinite values for configurations of templates which cannot be allowed. Nonlinearity is good for such constraints as: in a picture of a face the two eyes must be essentially in a horizontal line and must be within fixed limits of distance. The quality of the match is a function of the goodness of fit of the templates locally and the amount of "energy" needed to stretch the springs to force the input onto the reference data. Costs may be imposed for missing or extra elements.

The template match functions and spring functions are general procedures, thus the templates may be more general than pure iconic templates. Further,

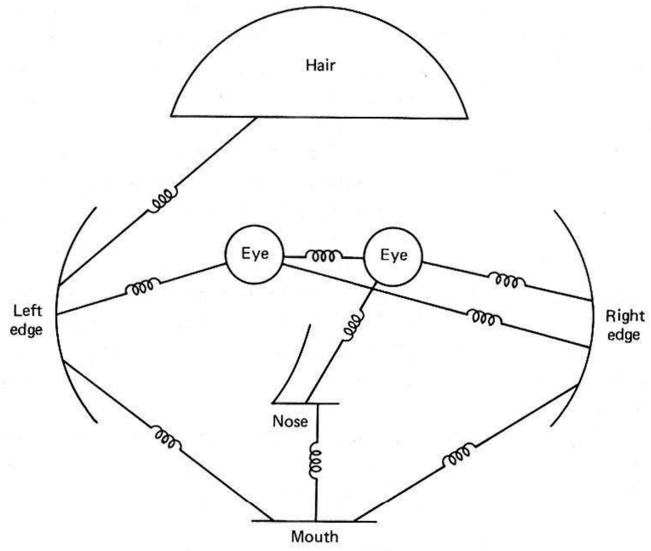


Fig. 11.5 A templates and springs model of a face.

matches may be defined not only between nodes and other nodes, but between nodes and image data directly. Thus the template and springs formalism is workable for "cross-representational" matching. The mechanism of minimizing the total cost of the match can take several forms; more detailed examples follow in Section 11.4.

Equation 11.3 a general form of the template-and-springs metric. TemplateCost measures dissimilarity between the input and the templates, and SpringCost measures the dissimilarity between the matched input elements' relations and the reference relations between the templates. MissingCost measures the penalties for missing elements. F(·) is the mapping from templates of the reference to elements of the input data. F partitions the reference templates into two classes, those found {FoundinRefer} and those not found {MissinginRefer} in the input data. If the input data are symbolic they may be similarly partitioned. The general metric is

$$\begin{aligned} \text{Cost} = & \sum_{d \in \{\text{FoundinRefer}\}} \text{TemplateCost}(d, F(d)) \\ & + \sum_{(d,e) \in \{\text{FoundinRefer} \times \text{FoundinInput}\}} \text{SpringCost}(F(d), F(e)) \\ & + \sum_{c \in \{\text{MissinginRefer}\} \cup \{\text{MissinginInput}\}} \text{MissingCost}(c) \end{aligned} \tag{11.3}$$

Equation 11.3 may be written as one sum of generalized SpringCosts in which the template properties are included (as 1-ary relations), as are "springs" involving missing elements.

As with correlation metrics, there are normalization issues involved with structural matching metrics. The number of elements matched may affect the ultimate magnitude of the metric. For instance, if springs always have a finite cost, then the more elements that are matched, the higher the total spring energy must be; this should probably not be taken to imply that a match of many elements is worse than a match of a few. Conversely, suppose that relations which agree are given positive "goodness" measures, and a match is chosen on the basis of the total "goodness." Then unless one is careful, the sheer number of possibly mediocre relational matches induced by matching many elements may outweigh the "goodness" of an elegant match involving only a few elements. On the other hand, a small, elegant match of a part of the input structure with one particular reference object may leave much of the search structure unexplained. This good "submatch" may be less helpful than a match that explains more of the input. To some extent the general metric (Eq.11.3) copes with this by acknowledging the "missing" category of elements.

If the reference templates actually contain iconic representations of what the input elements should look like in the image, a TemplateCost can be associated with a template and a location in the image by

$$\begin{aligned}&\text{TemplateCost(Template, Location)}\\&\qquad = (1 - \text{normalized correlation metric between}\\&\qquad\quad \text{template shape and input image at the location}).\end{aligned}$$

If the match is, for instance, to match reference descriptions of a chair with an input data structure, a typical "spring" might be that the chair seat must be supported by its legs. Thus if F is the association function mapping reference elements such as LEG or TABLETOP to input elements,

$$\text{SpringCost}_1(F(\text{LEG}), F(\text{TABLETOP})$$
$$= \begin{cases} 0 & \text{if } F(\text{LEG}) \text{ appears to support } F(\text{TABLETOP}), \\ 1 & \text{if } F(\text{LEG}) \text{ does not appear to support } F(\text{TABLETOP}). \end{cases}$$

For quantified relations, one might have

$$\text{SpringCost}_2 = \text{number of standard deviations from the canonical mean value for this relation.}$$

Another version of SpringCost_2 is the following [Barrow and Popplestone 1971].

$$\text{Cost of Match} = \frac{\text{SpringCosts of properties (unary) and binary relations}}{\text{total number of unary and binary springs}} + \frac{\text{Empirical Constant}}{\text{Total number of reference elements matched}} \tag{11.4}$$

The first term measures the average badness of matches between properties (unary relations) and relations between regions. The second term is inversely proportional to the number of regions that are matched, effectively increasing the cost of matches that explain less of the input.

11.3.2 Backtrack Search

Backtrack search is a generic name for a type of potentially exhaustive search organized in stages; each processing stage attempts to extend a partial solution derived in the previous stage. Should the attempt fail, the search "backtracks" to the most recent partial solution, from which a new extension is attempted. The technique is basic, amounting to a depth-first search through a tree of partial solutions (Fig. 11.6). Backtracking is a pervasive control structure in artificial intelli-

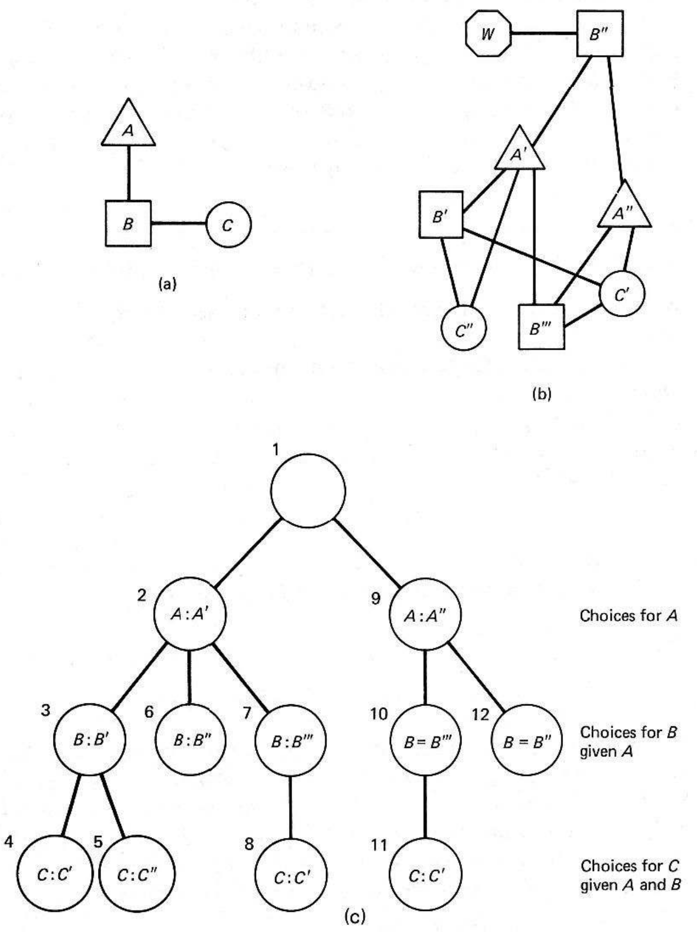


Fig. 11.6 The graph of (a) is to be matched in (b) with arcs all being unlabeled but nodes having properties indicated by their shapes, (c) is the tree of solutions built by a backtrack algorithm.

gence, and through the years several general classes of techniques have evolved to make the basic, brute-force backtrack search more efficient.

Example: Graph Isomorphisms

Given two graphs,

$$X = (V_X, E_X)$$
$$Y = (V_Y, E_Y),$$

without loss of generality, let $V_X = V_Y = \{1, 2, \ldots, n\}$, and let X be the reference graph, Y the input graph. The isomorphism is given by: If $i \in V_X$, the corresponding node under the isomorphism is $F(i) \in V_Y$.

In the algorithm, S is the set of nodes accounted for in Y by a partial solution. k gives the current level of the search in the tree of partial solutions, the number of nodes in the current partial solution, and the node of X whose match in Y is currently being sought. v is a node of Y currently being considered to extend the current partial solution. As written, the algorithm finds all isomorphisms. It is easily modified to quit after finding the first.

Algorithm 11.1 Backtrack Search for Directed Graph Isomorphism

```
Recursive Procedure DirectedGraphIsomorphisms($S,k$);
begin
 if $S=V_Y$ then ReportAsIsomorphism(F)
 else
  forall $v \in (V_Y-S)$
   do
   if Match($k,v$)
   then
    begin
     $F(k) := v$;
     DirectedGraphIsomorphisms ($S \in \{v\}, k+1$);
    end;
end;
```

ReportAsIsomorphism could print or save the current value of F, the global structure recording the current solution. Match(k,v) is a procedure that tests whether $v \in V_Y$ can correspond to $k \in V_X$ under the isomorphism so far defined by F. Let X_k be the subgraph of X with vertices $\{1, 2, \ldots, k\}$. The procedure "Match" must check for $i < k$, whether (i, k) is an edge of X_k iff $(F(i), v)$ is an edge of Y *and* whether (k, i) is an edge of X_k iff $(v, F(i))$ is an edge of Y.

Improving Backtrack Search

Several techniques are useful in improving the efficiency of backtrack search [Bittner and Reingold 1975]:

1. *Branch pruning.* All techniques of this variety examine the current partial solution and prune away descendents that are not viable continuations of the solution. Should none exist, backtracking can take place immediately.
2. *Branch merging.* Do not search branches of the solution tree isomorphic with those already searched.
3. *Tree rearrangement and reordering.* Given pruning capabilities, more nodes are likely to be eliminated by pruning if there are fewer choices to make early in the search (partial solution nodes of low degree should be high in the search tree). Similarly, search first those extensions to the current solution that have the fewest alternatives.
4. *Branch and bound.* If a cost may be assigned to solutions, standard techniques such as heuristic search and the A* search algorithm [Nilsson 1980] (Section 4.4) may be employed to allow the search to proceed on a "best-first" rather than a "depth-first" basis.

For extensions of these techniques, see [Haralick and Elliott 1979].

11.3.3 Association Graph Techniques

Generalized Structure Matching

A general relational structure "best match" is less restricted than graph isomorphism, because nodes or arcs may be missing from one or the other graph. Also, it is more general than subgraph isomorphism because one structure may not be exactly isomorphic to a substructure of the other. A more general match consists of a set of nodes from one structure and a set of nodes from the other and a 1:1 mapping between them which preserves the compatibilities of properties and relations. In other words, corresponding nodes (under the node mapping) have sufficiently similar properties, and corresponding sets under the mapping have compatible relations.

The two relational structures may have a complex makeup that falls outside the normal purview of graph theory. For instance, they may have parameterized properties attached to their nodes and edges. The definition of whether a node matches another node and whether two such node matches are mutually compatible can be determined by arbitrary procedures, unlike the much simpler criteria used in pure graph isomorphism or subgraph isomorphism, for example. Recall that the various graph and subgraph isomorphisms rely heavily on a 1:1 match, at least locally, between arcs and nodes of the structures to be matched. However, the idea of a "best match" may make sense even in the absence of such perfect correspondences.

The *association graph* defined in this section is an auxiliary data structure produced from two relational structures to be matched. The beauty of the association graph is that it *is* a simple, pure graph-theoretic structure which is amenable to pure graph-theoretic algorithms such as clique finding. This is useful for several reasons.

- It takes relational structure matching from the ad hoc to the classical domain.
- It broadens the base of people who are producing useful algorithms for structure matching. If the rather specialized relational structure matching enterprise is reducible to a classical graph-theoretical problem, then everyone working on the classical problem is also working indirectly on structure matching.
- Knowledge about the computational complexity of classical graph algorithms illuminates the difficulty of structure matching.

Clique Finding for Generalized Matching

Let a relational structure be a set of elements V, a set of properties (or more simply unary predicates) P defined over the elements, and a set of binary relations (or binary predicates) R defined over pairs of the elements. An example of a graph representation of such a structure is given in Fig. 11.7.

Given two structures defined by (V_1, P, R) and (V_2, P, R), say that "similar" and "compatible" actually mean "the same." Then we construct an association graph G as follows [Ambler et al. 1975]. For each v_1 in V_1 and v_2 in V_2, construct a node of G labeled (v_1, v_2) if v_1 and v_2 have the same properties [$p(v_1)$ iff $p(v_2)$ for each p in P]. Thus the nodes of G denote assignments, or pairs of nodes, one each from V_1 and V_2, which have similar properties. Now connect two nodes (v_1, v_2) and (v'_1, v'_2) of G if they represent *compatible* assignments according to R, that is, if the pairs satisfy the same binary predicates [$r(v_1, v'_1)$ iff $r(v_2, v'_2)$ for each r in R].

A match between (V_1, P, R) and (V_2, P, R), the two relational structures, is just a set of assignments that are all mutually compatible. The "best match" could well be taken to be the largest set of assignments (node correspondences) that were all mutually compatible under the relations. But this in the association graph G is just the largest totally connected (completely mutually compatible)—set of nodes. It is a *clique*. A clique to which no new nodes may be added without destroying the clique properties is a *maximal* clique. In this formulation of matching, larger cliques are taken to indicate better matches, since they account for more nodes.

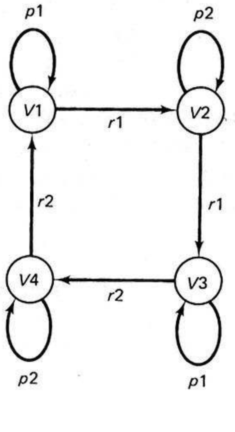


Fig. 11.7 A graph representation of a relational structure. Elements (nodes) v_1 and v_3 have property p1, v_2 and v_4 have property p2, and the arcs between nodes indicate that the relation r1 holds between v_1 and v_2 and between v_2 and v_3, and r2 holds between v_3 and v_4 and between v_4 and v_1.

Thus the best matches are determined by the largest maximal cliques in the association graph. Figure 11.8 shows an example: Certain subfeatures of the objects have been selected as "primitive elements" of the objects, and appear as nodes (elements) in the relational structures. To these nodes are attached properties, and between them can exist relations. The choice of primitives, properties, and relations is up to the designer of the representation. Here the primitives of the representation correspond to edges and corners of the shape.

The association graph is shown in 11.8e. Its nodes correspond to pairs of nodes, one each from A and B, whose properties are similar. [Notice that there is no node in the association graph for (6,6′)]. The arcs of the association graph indicate that the endpoints of the arc represent compatible associations. Maximal cliques in the association graph (shown as sets of nodes with the same shape) indicate sets of consistent associations. The largest maximal clique provides the node pairings of the "best match."

In the example construction, the association graph is formed by associating nodes with exactly the same properties (actually unary predicates), and by allowing as compatible associations only those with exactly the same relations (actually binary predicates). These conditions are easy to state, but they may not be exactly what is needed. In particular, if the properties and relations may take on ranges of values greater than the binary "exists" and "does not exist," then a measure of similarity must be introduced to define when node properties are similar enough for association, and when relations are similar enough for compatibility. Arbitrarily complex functions can decide whether properties and relations are similar. As long as the function answers "yes" or "no," the complexity of its computations is irrelevant to the matching algorithm.

The following recursive clique-finding algorithm builds up cliques a node at a time [Ambler et al. 1975]. The search tree it generates has states that are ordered pairs (set of nodes chosen for a clique, set of nodes available for inclusion in the clique). The root of the tree is the state (∅, all graph nodes), and at each branch a choice is made whether to include or not to include an eligible node in the clique. (If a node is eligible for inclusion in clique X, then *each* clique including X must either include the node or exclude it).

Algorithm 11.2: Clique-Finding Algorithm

```
Comment Nodes is the set of nodes in the input graph.
Comment
  Cliques (X, Y)  takes as arguments a clique X, and Y, a set of nodes that includes
                  X. It returns all cliques that include X and are included in Y.
  Cliques (∅,Nodes) finds all cliques in the graph.
Cliques(X, Y) :=
  if no node in Y−X is connected to all elements of X
    then {X}
    else
      Cliques(X ⋃ {y}, Y) ⋃ Cliques (X, Y−{y})
      where y is connected to all elements of X.
```

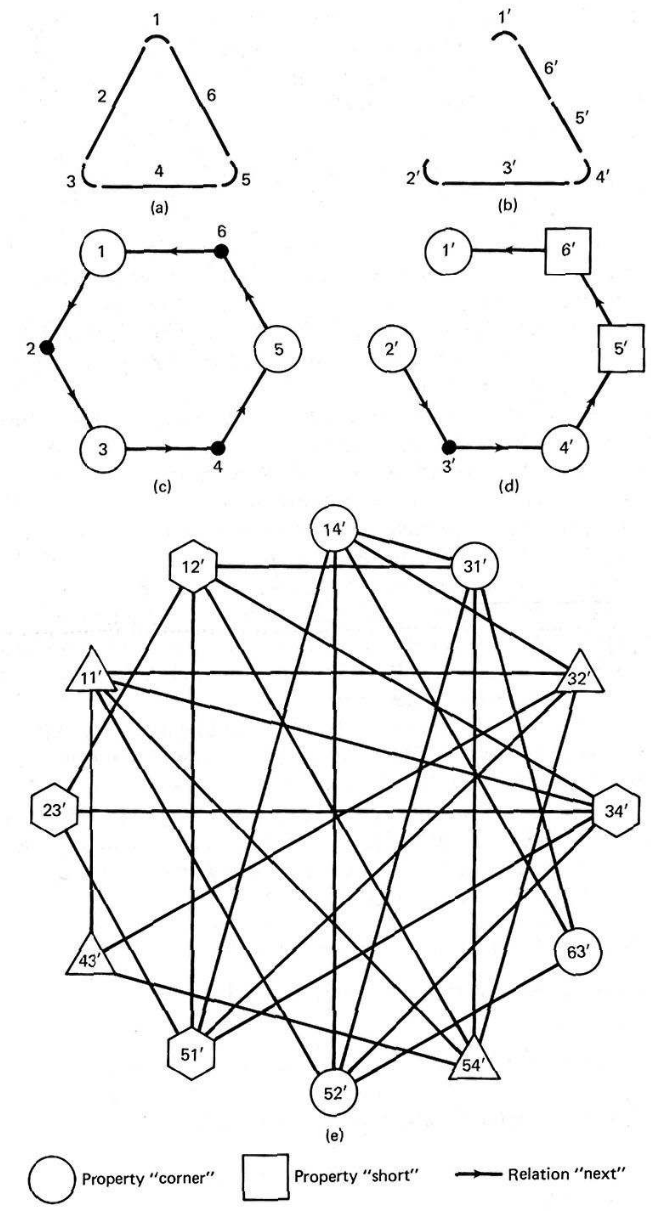


Fig. 11.8 Clique-finding example. Entities to be matched are given in (a) (reference) and (b) (input). The relational structures corresponding to them are shown in (c) and (d). The resulting association graph is shown in (e) with its largest cliques indicated by node shapes.

Extensions

Modifications to the clique-finding algorithm extend it to finding maximal cliques and finding largest cliques. To find largest cliques, perform an additional test to stop the recursion in *Cliques* if the size of X plus the number of nodes in $Y-X$ connected to all of X becomes less than k, which is initially set to the size of the largest possible clique. If no cliques of size k are found, decrement k and run *Cliques* with the new k.

To find maximal cliques, at each stage of *Cliques*, compute the set

$$Y' = \{z \in \text{Nodes: } z \text{ is connected to each node of Y}\}.$$

Since any maximal clique must include Y', searching a branch may be terminated should Y' not be contained in Y, since Y can then contain no maximal cliques.

The association graph may be searched not for cliques, but for r-connected components. An *r-connected* component is a set of nodes such that each node is connected to at least r other nodes of the set. A clique of size n is an $n-1$-connected component. Fig. 11.9 shows some examples.

The r-connected components generalize the notion of clique. An r-connected component of N nodes in the association graph indicates a match of N pairs of nodes from the input and reference structures, as does an N-clique. Each matching pair has similar properties, and each pair is compatible with at least r other matches in the component.

Whether or not the r-connected component definition of a match between two structures is useful depends on the semantics of "compatibility." For instance, if all relations are either compulsory or prohibited, clearly a clique is called for. If the relations merely give some degree of mutual support, perhaps an r-connected component is the better definition of a match.

11.4 MATCHING IN PRACTICE

This section illustrates some principles of matching with examples from the computer vision literature.

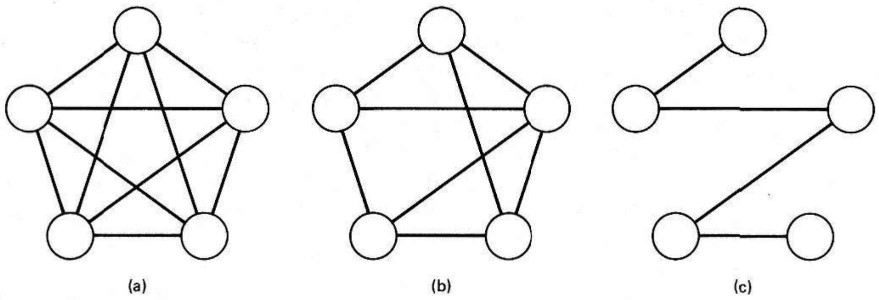


Fig. 11.9 r-connected components. (a) A 5-clique (which is 4-connected). (b) A 3-connected set of 5 nodes. (c) A 1-connected set of 5 nodes.

11.4.1 Decision Trees

Hierarchical decision-tree matching with ad hoc metrics is a popular way to identify input data structures as instances of reference models and thus classify the input instances [Nevatia 1974; Ambler et al. 1975; Winston 1975]. Decision trees are indicated when it is predictable that certain features are more reliably extracted than others and that certain relations are either easier to sense or more necessary to the success of a match.

Winston and Nevatia both compare matches with a "weighted sums of difference" metric that reflects cumulative differences between the parameters of corresponding elements and relations in the reference and input data. In addition, Nevatia does parameter fitting; his reference information includes geometrical information.

Matching Structural Descriptions

Winston is interested in matching such structures as appear in Fig. 11.10B. The idea is to recognize instances of structural concepts such as "arch" or "house," which are relational structures of primitive blocks (Fig.11.10A) [Winston 1975]. An important part of the program learns the concept in the first place—only the matching aspect of the program is discussed here. His system has the pleasant property of representational uniqueness: reference and input data structures that are identical up to the resolution of the descriptors used by the program have identical representations. Matching is easy in this case. Reflections of block structures can be recognized because the information available about relations (such as LEFT-OF and IN-FRONT-OF) includes their OPPOSITE (i.e., RIGHT-OF and BEHIND). The program thus can recognize various sorts of symmetry by replacing all input data structure relations by their relevant opposite, then comparing with the reference.

The next most complicated matching task after exact or symmetric matches is to match structures in isolation. Here the method is sequentially to match the input data against the whole reference data catalog of structures and determine which match is best (which difference description is most inconsequential). Easily computed scene characteristics can rule out some candidate models immediately.

The models contain arcs such as MUST-BE and MUST-NOT, expressing relations mandatory or forbidden relations. A match is not allowed between a description and a model if one of the strictures is violated. For instance, the program may reject a "house" immediately as not being a "pedestal," "tent," or "arch," since the pedestal top must be a parallelepiped, both tent components must be wedges, and the house is missing a component to support the top piece that is needed in the arch. These outright rejections are in a sense easy cases; it can also happen that more than one model matches some scene description. To determine the best match in this case, a weighted sum of differences is made to express the amount of difference.

The next harder case is to match structures in a complex scene. The issue here is what to do about evidence that is missing through obscuration. Two heuristics help:

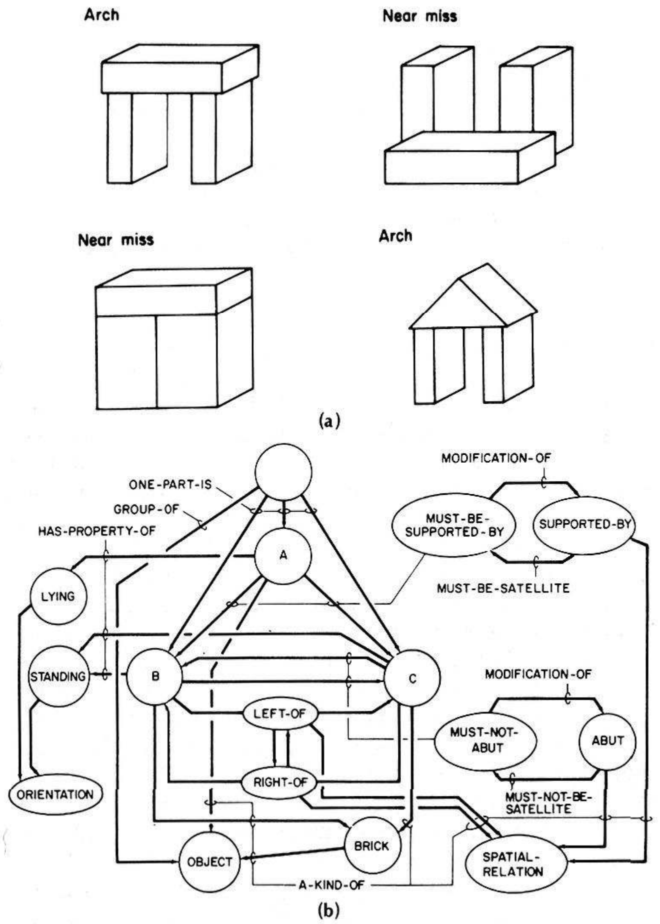


Fig. 11.10 (a) Several arches and non-arches. (b) The computer-generated arch description to be used for matching.

1. Objects that seem to have been stacked and could be the same type are of the same type.
2. Essential model properties may be hidden in the scene, so the match should not be aborted because of missing essential properties (though the presence of forbidden properties is enough to abort a match).

This latter rule is equivalent to Nevatia's rules about connectivity difference and missing input instance parts (see below). In terms of the general structure metric introduced earlier, neither Winston or Nevatia penalize the match for missing elements or relations in the reference data. One result of this is that the best match is sometimes missed in favor of the first possible match. Winston suggests that complex scenes be analyzed by identifying subscenes and subtracting out the identified parts, as was done by Roberts.

Winston's program can learn shortcuts in matching strategy by itself; it builds for itself a similarity network relating models whose differences are slight. If a reference model does not quite fit an input structure, the program can make an intelligent choice of the next model to try. A good choice is a model that has only minor differences with the first. This self-organization and cataloging of the models according to their mutual differences is a powerful way to use matching work that is already performed to guide further search for a good match.

Backtrack Search

Nevatia addresses a domain of complex articulated biological-like forms (hands, horses, dolls, snakes) [Nevatia 1974]. His strategy is to segment the objects into parts with central axes and "cross section" (not unlike generalized cylinders, except that they are largely treated in two dimensions). The derived descriptions of objects contain the connectivity of subparts, and descriptions of the shape and joint types of the parts. Matching is needed to compare descriptions and find differences, which can then be explained or used to abort the match. Partial matches are important (as in most real-world domains) because of occlusions, noise, and so on.

A priori ideas as to the relative importance of different aspects of structures are used to impose a hierarchical order on the matching decision tree. Nevatia finds this heuristic approach more appealing than a uniform, domain-independent one such as clique finding. His system knows that certain parts of a structure are more important than others, and uses them to index into the reference data catalog containing known structures. Thus relevant models for matching may be retrieved efficiently on the basis of easily-computed functions of the input data. The models are generated by the machine by the same process that later extracts descriptions of the image for recognition. Several different models are stored for the same view of the same object, because his program has no idea of model equivalence, and cannot always extract the same description.

The matching process is basically a depth-first tree search, with initial choices being constrained by "distinguished pieces." These are important pieces of image which first dictate the models to be tried in the match, and then constrain the possible other matches of other parts.

There is a topological and a geometrical component to the match. The topological part is based on the connectivity of the "stick figure" that underlies the representation. The geometrical part matches the more local characteristics of individual pieces. Consider Nevatia's example of matching a doll with stored reference descriptions, including those of a doll and a horse.

By a process not concerning us here, the doll image is segmented into pieces as shown in Fig. 11.11. From this, before any matching is done, a connection graph of pieces is formed, as shown in Fig. 11.12.

This connection graph is topologically the same as the reference connection graph for the doll, which looks as one would expect. In both reference and input, "distinguished pieces" are identified by their large size. During reference forma-

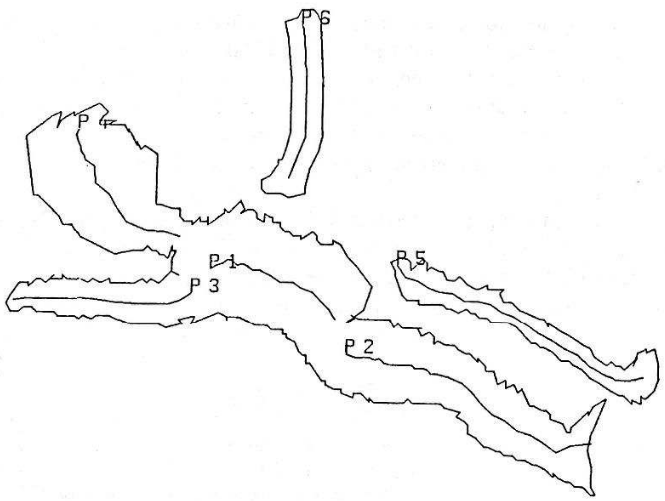


Fig. 11.11 A view of a doll, with derived structure.

tion time, the two largest pieces were the head and the trunk, and these are the distinguished pieces in the reference. There are the same pieces picked as distinguished in the instance to be matched consistent with the hierarchical decision-tree style, distinguished pieces are matched first.

Because of noise, connections at joints may be missed; because of the nature of the objects, extra joints are hardly ever produced. Thus there is a domain-dependent rule that an input piece with two other pieces connected at a single joint (a "two-ended piece") cannot match a one-ended reference piece, although the reverse is possible.

On the basis of the distinguished pieces in the input instance, the program decides that the instance could be a doll or a horse. Both these possibilities are evaluated carefully; Fig. 11.13 shows a schematic view of the process. Piece-match evaluation must be performed at the nodes of the tree to determine which pieces at a joint should be made to correspond.

The final best match between the doll input and the horse reference model is diagrammed in Fig. 11.14. This match is not as good as the match between the doll input and the doll reference.

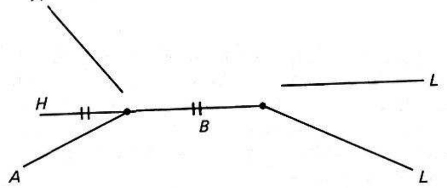
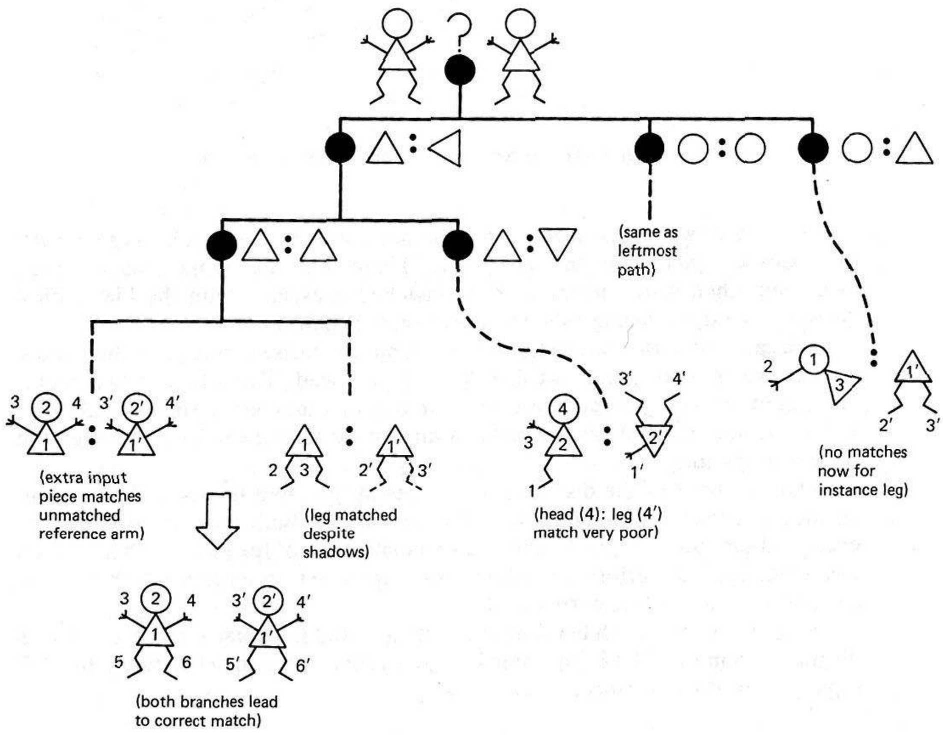


Fig. 11.12 Connection graph of the doll.

The final choice of matches is made with a version of the general relational structure matching metric (Eq. 11.3). It takes into account the connectivity relations, which are the same in this case, and the quality of the individual piece matches. In the doll-horse match, more reference parts are missing, but this can happen if parts are hidden in a view, and do not count against the match. The doll-doll match is preferred on the basis of piece matching, but both matches are considered possible.

In summary, the selection of best match proceeds roughly as follows: unacceptable differences are first sought (not unlike the Winston system). The number of input pieces not matched by the reference is an important number (not vice versa, because of the possibility of hidden input parts). Only elongated, large parts



?
(same as leftmost path)
(extra input piece matches unmatched reference arm)
(leg matched despite shadows)
(head (4): leg (4′) match very poor)
(no matches now for instance leg)
(both branches lead to correct match)

Fig. 11.13 A pictorial guide to the combinations tried by the matcher establishing the best correspondence of the doll input with the doll reference. The graphic shapes are purely pedagogical: the program deals with symbolic connectivity information and geometric measurements. Inferences and discoveries made by the program while matching are given in the diagram. A:B means that structure A is matched with structure B, with the numbered substructures of A matching their primed counterparts in B.

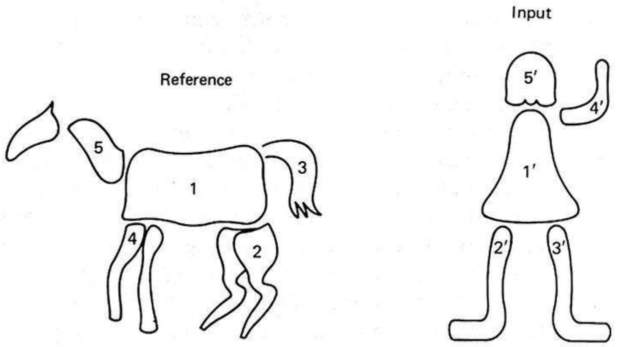


Fig. 11.14 The best match of the doll input with the horse reference model. One doll arm is unmatched, as is the horse head and two legs.

are considered for this determination, to eliminate small "noise" patches. The match with fewest unmatched input pieces is chosen.

If no deciding structural differences exist, the quality of piece matches determines the quality of the match. These correspond to the template cost term in Eq. (11.3). If a "significant" difference in match error exists, the better match is exclusively selected; if the difference is not so great as that, the better match is merely preferred.

Piece matching is a subprocess of joint matching. The difference in the number of pieces attached at the ends of the piece to be matched is the *connectivity difference.* If the object piece has more pieces connected to it than the model piece, the match is a poor one; since pieces may not be visible in a view, the converse is not true. If one match gives fewer excess input pieces, it is accepted at this point. If not, the goodness of the match is computed as a weighted sum of width difference, length-to-width ratio difference, and difference in acuteness of the generalized cylinders (Chapter 9) forming the pieces. The weighted sum is thresholded to yield a final "yes or no" match result. Shadowing phenomena are accommodated by allowing the input piece to be narrower than the reference model piece with no penalty. The error function weights are derived empirically; one would not expect the viewing angle to affect seriously the width of a piece, for example, but it could affect its length. Piece axis shapes (what sort of space curve they are) are not used for domain-dependent reasons, nor are cross section functions (aside from a measure of "acuteness" for cone-like generalized cylinders).

11.4.2 Decision Tree and Subgraph Isomorphism

A robotics program for versatile assembly [Ambler et al. 1975] uses matching to identify individual objects on the basis of their boundaries, and to match several individual blobs on a screen with a reference model containing the known location of multiple objects in the field of view. In both cases the best subgraph isomorphism between input and reference data structures is found when necessary by the clique-finding technique (Algorithm 11.2).

The input data to the part recognizer consist of silhouettes of parts with outlines of piecewise linear and circular segments. A typical set of shapes to be recognized might be stored in terms of boundary primitives as shown in Fig. 11.15a, with matchable and unmatchable scenes shown in Fig. 11.15b.

Generally, the matching process works on hierarchical structures which capture increasing levels of detail about the objects of interest. The matching works its way down the hierarchy, from high-level, easily computable properties such as size down to difficult properties such as the arrangements of linear segments in a part outline. After this decision tree pre-processing, all possible matches are computed by the clique-finding approach to subgraph isomorphism. A scene can be assigned a number of interpretations, including those of different views of the same part. The hierarchical organization means that complicated properties of the scene are not computed unless they are needed by the matcher. Once computed they are never recomputed, since they are stored in accessible places for later retrieval if needed. Each matching level produces multiple interpretations; ambiguity is treated with backtracking. The system recognizes rotational and translational invariance, but must be taught different views of the same object in its different gravitationally stable states. It treats these different states basically as different objects.

11.4.3 Informal Feature Classification

The domain of this work is one of small, curved tabletop objects, such as a teacup (Fig. 11.16) [Barrow and Popplestone 1971]. The primitives in models and image descriptions are regions which are found by a process irrelevant here. The regions have certain properties (such as shape or brightness), and they have certain parameterized relations with other regions (such as distance, adjacency, "aboveness"). The input and reference data are both relational structures. The properties and relations are the following:

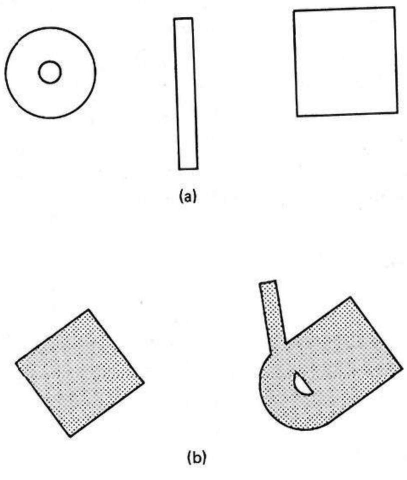


Fig. 11.15 A small catalog of part boundaries (a) and some sample silhouettes (b). The "heap" will not match any part very well, while the square can match the square model in four different ways, through rotations. Gross properties such as area may be used cheaply to reject matches such as the square with the axle.

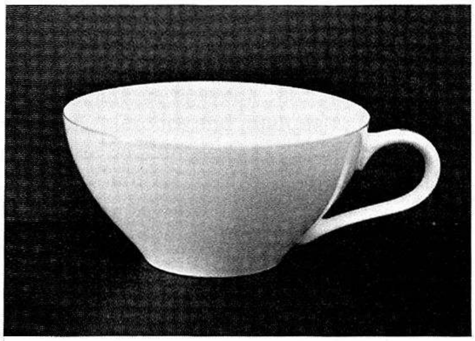


Fig. 11.16 An object for recognition by relational matching.

1. *Region Properties*

   Shape 1–Shape 6: the first six root mean square amplitudes of the Fourier components of the $\phi(s)$ curve [Chapter 8].

2. *Relations between Regions A and B*

   Bigger: Area(A)/Area(B)

   Adjacency: Fraction of A's boundary which also is a boundary of B.

   Distance: Distance between centroids divided by the geometric mean of average radii. The average radius is twice the area over the perimeter. Distance is scale, rotation, translation, reflection invariant.

   Compactness: $4*\pi*area/perimeter^2$

   Above, Beside: Vertical and horizontal distance between centroids, normalized by average radius. Not rotation invariant.

The model that might be derived for the cup of Fig. 11.16 is shown in Fig. 11.17.

The program works on objects such as spectacles, pen, cup, or ball. During training, views and their identifications are given to the program, and the program forms a relational structure with information about the mean and variance of the values of the relations and properties. After training, the program is presented with a scene containing one of the learned objects. A relational structure is built describing the scene; the problem is then to match this input description with a reference description from the set of models.

One approximation to the goodness of a match is the number of successes provided by a region correspondence. A one-region object description has 7 relations to check, a two-region object has 28, a three-region one has 63. Therefore, the "successes" criterion could imply the choice of a terrible three-region interpretation over a perfect one-region match. The solution adapted in the matching evaluation is first to grade failures. A failure weight is assigned to a trial match according to how many standard deviations σ from the model mean the relevant

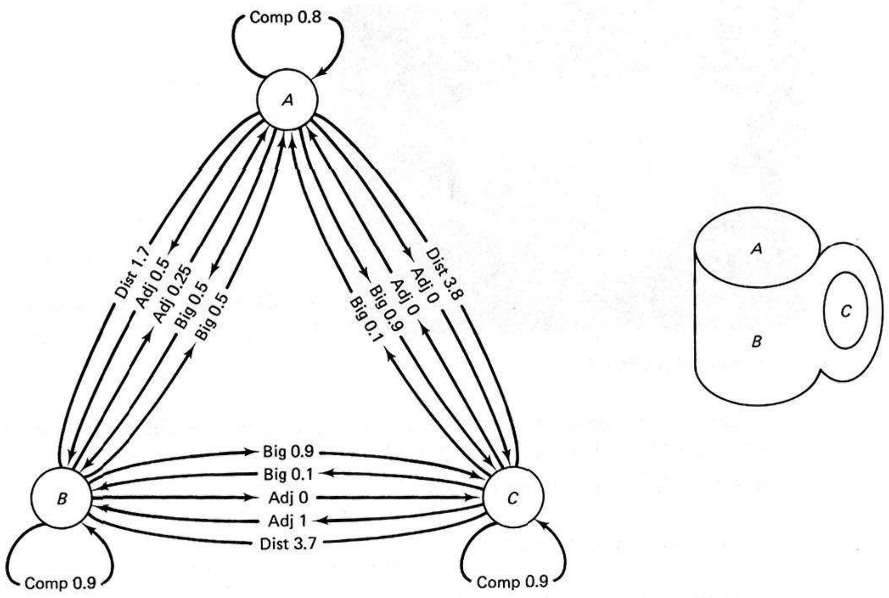


Fig. 11.17 Relational model for cups such as that of Fig. 11.16.

parameter is. From zero to three σ imply a success, or a failure weight of 0; from three to six σ, a failure weight of 1; from six to nine σ, failure weight of 2, and so on. Then the measure "trials–cumulative failure weight" is an improvement on just "successes." On the other hand, simple objects are often found as subparts of complex ones, and one does not want to reject a good interpretation as a complex object in favor of a less explanatory one as a simple object. The final evaluation function adapted is

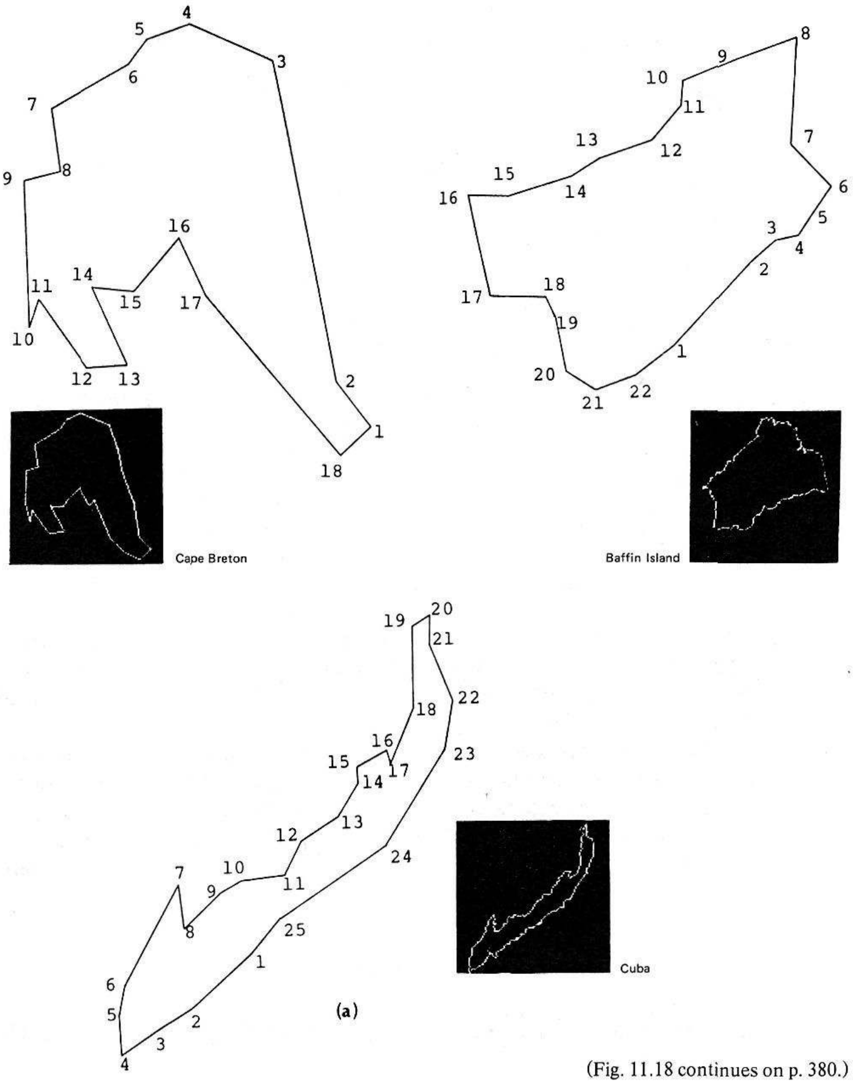
$$\text{Cost of Match} = \frac{1 - (\text{tries-failure weight})}{\text{number of relations}} + \frac{K}{\text{number of regions in view description}} \tag{11.5}$$

As in Eq. (11.4), the first term measures the average badness of matches between properties (unary relations) and relations between regions. The second term is inversely proportional to the number of regions that are matched, effectively increasing the cost of matches that explain less of the input.

11.4.4 A Complex Matcher

A program to match linear structures like those of Fig. 11.18 is described in [Davis 1976]. This matcher presents quite a diversity of matching techniques incorporated into one domain-dependent program.

The matching metric is very close to the general metric of Eg. (11.3). The match is characterized by a structural match of reference and input elements and a geometrical transformation (found by parameter fitting) which accounts for the spatial relations between reference and input. Davis forms an association graph between reference and input data. This graph is reduced by parallel-iterative relaxation (see Section 12.4) using the "spring functions" to determine which node associations are too costly. Eliminating one node–node match may render others



(a)

(Fig. 11.18 continues on p. 380.)

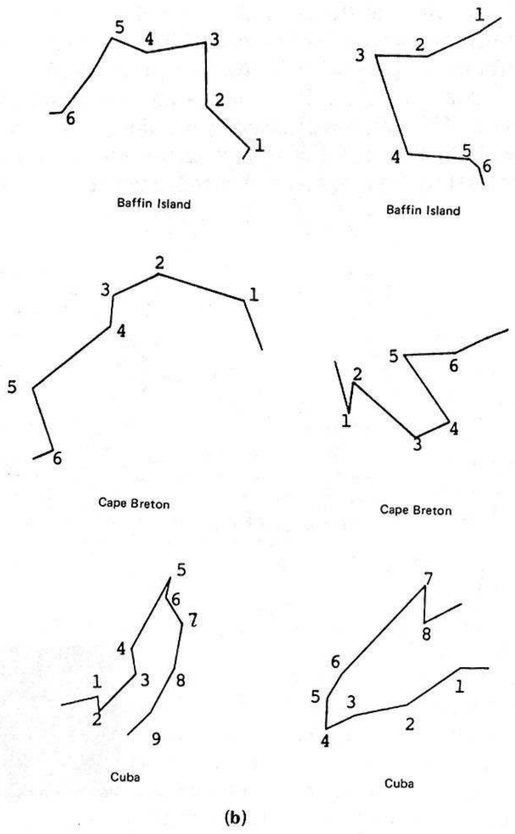


Fig. 11.18 (a) Reference and (b) input data for a complex shape matching program.

more unlikely, so the node-pruning process iterates until no more nodes are eliminated. What remains is something like an *r*-connected component of the graph, which specifies an approximate match supported by some amount of consistent relations between nodes.

After the process of constraint relaxation, there are still in general several locally consistent interpretations for each component of the input structure. Next, therefore, a tree search is used to establish global consistency and therefore the best match. The tree search is the familiar "best first" heuristic search through the partial match space, with pruning taking place between each stage of search again by using the parallel-iterative relaxation technique.

EXERCISES

11.1 Relational structures *A* and *B* are to be matched by the association-graph, clique-finding method.

Relational structure A: entities u, v, w, x, y, z.
relations $P(u), P(w), P(y), R(v), R(x), R(z)$,
$F(u, v), F(v, w), F(w, x), F(x, y), F(y, z), F(z, u)$

Relational structure B: entities a, b, c, d, e, f.
relations $P(a), P(b), P(d), Q(e), Q(f), R(c)$
$F(b, c), F(c, d), F(d, e), F(e, f), F(f, a)$.

(a) Construct graph structures corresponding to the structures A and B. Label the nodes and arcs.

(b) Construct the association graph of structures A and B.

(c) Visually find the largest maximal cliques in the association graph and thus the best matches between A and B. (There are three.)

11.2 Suppose in a geometric match that two input points on the xy plane are identified with two others taken to correspond with two reference points. It is known that the input data comes about only through rotation and translation of the reference data. Given the two input points (x_1, y_1) and (x_2, y_2) and the two reference points (x'_1, y'_1) and (x'_2, y'_2), one way to find the transformation from reference to input is to solve the equation

$$\sum_{i=1}^{2} [x_i - (ax'_i + by'_i + c)]^2 + [y_i - (bx'_i + ay'_i + d)]^2 = 0$$

The resulting values of a, b, c, and d represent the desired transformation. Solve the equation analytically to get expressions for a, b, c, and d in terms of the reference and input coordinates. What happens if the reference and input data are not related by simple rotation and translation?

11.3 What are the advantages and disadvantages of a uniform method (such as subgraph isomorphism algorithm approach) to matching as compared to an ad hoc (such as a decision-tree approach with various empirically derived metrics) one?

11.4 In the worst case, for graphs of n nodes, how many partial solutions total will Algorithm 11.1 have to proceed through? Construct "worst case" graphs X and Y (label their nodes $1, \ldots, n$, of course), assuming that nodes of Y are selected in ascending order at any stage.

11.5 Find out something about the state of associative memories in computers. How do they work? How are they used? Would anything like this technology be useful for computer vision? Introspect about familiar phenomena of visual recall, recognition, and memory. Do you have a theory about how human visual memory could possibly work?

11.6 What graph of N nodes has the maximum number of maximal cliques? How many does it have?

11.7 Think about reasoning by analogy and find out something about programs that do analogical reasoning. In what sense can analogical process be used for computer vision, and technically do the matching techniques necessary provide any insight?

11.8 Compare Nevatia's structure matching with Hinton's relaxation-based puppet recognition (Chapter 12).

11.9 Verify the observation made in Section 11.4.3 about the number of relations that must be checked between regions (one region, 7; two regions, 28; three regions, 63; etc.).

REFERENCES

AHO, A. V., J. E. HOPCROFT and J. D. ULLMAN. *The Design and Analysis of Algorithms.* Reading, MA: Addison-Wesley, 1974.

AMBLER, A. P., H. G. BARROW, C. M. BROWN, R. M. BURSTALL, and R. J. POPPLESTONE. "A versatile computer-controlled assembly system." *Artificial Intelligence 6*, 2, 1975, 129–156.

BARROW, H. G. and R. J. POPPLESTONE. "Relational descriptions in picture processing." In *MI6*, 1971.

BARROW, H. G., J. M. TENENBAUM, R. C. BOLLES, and H. C. WOLF. "Parametric correspondence and chamfer matching: two new techniques for image matching." *Proc.*, DARPA IU Workshop, May 1978, 21-27.

BERGE, C. *Graphs and Hypergraphs* 2nd rev. ed.. New York: American Elsevier, 1976.

BERZTISS, A. T. "A backtrack procedure for isomorphism of directed graphs." *J. ACM 20*, 3, July 1973, 365-377.

BITTNER, J. R. and E. M. REINGOLD. "Backtrack programming techniques." *Comm. ACM 18*, 11, November 1975, 651-656.

BRON, C. and J. KERBOSCH. "Algorithm 457: finding all cliques in an undirected graph (H)." *Comm. ACM 16*, 9, September 1973, 575-577.

CORNEIL, D. G. and C. C. GOTLIEB. "An efficient algorithm for graph isomorphism." *J. ACM 17*, 1, January 1970, 51-64.

DAVIS, L. S. "Shape matching using relaxation techniques." *IEEE-PAMI 1*, 1, January 1979, 60-72.

FISCHLER, M. A. and R. A. ELSCHLAGER. "The representation and matching of pictorial structures." *IEEE Trans. Computers 22*, 1, January 1973, 67-92.

HARALICK, R. M. and G. L. ELLIOTT. "Increasing tree search efficiency for constraint satisfaction problems." *Proc.*, 6th IJCAI, August 1979, 356-364.

HARARY, F. *Graph Theory.* Reading, MA: Addison-Wesley, 1969.

KNODEL, W. "Bestimmung aller maximalen vollstandigen Teilgraphen eines Graphen G nach Stoffers." *Computing 3*, 3, 1968, 239–240 (and correction in *Computing 4*, p. 75).

NEVATIA, R. "Structured descriptions of complex curved objects for recognition and visual memory." AIM-250, Stanford AI Lab, October 1974.

NILLSON, N.J. *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga, 1980.

OSTEEN, R. E. and J. T. TOU. "A clique-directed algorithm based on neighbourhoods in graphs." *International J. Computer and Information Science 2*, 4, December 1973, 257–268.

REINGOLD, E. M., J. NIEVERGELT, and N. DEO. *Combinatorial Algorithm Theory and Practice.* Englewood Cliffs,N. J.: Prentice-Hall, 1977.

SCHUDY, R. B. and D. H. BALLARD. "Model-directed detection of cardiac chambers in ultrasound images." TR12, Computer Science Dept., Univ. Rochester, November 1978.

SHAPIRO, L. G. and R. M. HARALICK. "Structural descriptions and inexact matching." Technical Report CS79011-R, Computer Science Dept., Virginia Polytechnic Institute, November 1979.

ULLMAN, J. R. "An algorithm for a subgraph isomorphism." *J. ACM 23*, 1, January 1976, 31-42.

WINSTON, P. H. "Learning structural descriptions from examples." In *PCV*, 1975.

Inference 12

Classical and Extended Inference

This chapter explores *inference,* the process of deducing facts from other known facts. Inference is useful for belief maintenance and is a cornerstone of rational thought. We start with *predicate logic*, and then explore *extended inference* systems—production systems, relaxation labeling, and active knowledge (procedures).

Predicate logic (Section 12.1) is a system for expressing propositions and for deriving consequences of facts. It has evolved over centuries, and many clear accounts describe predicate logic in its various forms [Mendelson 1964; Robinson 1965]. It has good formal properties, a nontrivial but automatable inference procedure, and a history of study in artificial intelligence. There are several "classical" extensions (modal logics, higher-order logics) which are studied in well-settled academic disciplines of metamathematics and philosophy. *Extended inference* (Section 12.2) is possible in automated systems, and is interesting technically and from an implementational standpoint.

A *production system* (Section 12.3) is a general rewriting system consisting of a set of *rewriting rules* ($A \rightarrow BC$ could mean "rewrite A as BC") and an executive program to apply rewrites. More generally, the rules can be considered "situation–action" pairs ("in situation A, do B and C"). Thus production systems can be used to control computational activities. Production systems, like semantic nets, embody powerful notions that can be used for extended inference.

Labeling schemes (Section 12.4) are unlike most inference mechanisms in that they often involve mathematical optimization in continuous spaces and can be implemented with parallel computation. Labeling is like inference because it establishes consistent "probability-like" values for "hypotheses" about the interpretation of entities.

Active knowledge (Section 12.5) is an implementation of inference in which each chunk of knowledge is a program. This technique goes far in the direction of "proceduralizing" the implementation of propositions. The design issues for such a system include the vocabulary of system primitives and their actions, mechanisms for implementing the flow of control, and overall control of the action of the system.

12.1 FIRST ORDER PREDICATE CALCULUS

Predicate logic is in many ways an attractive knowledge representation and inference system. However, despite its historical stature, important technical results in automated inference, and much research on inference techniques, logic has not dominated all aspects of mechanized inference. Some reasons for this are presented in Sections 12.1.6 and 12.2. The logical system that has received the most study is *first order predicate logic.* General theorem provers in this calculus are cumbersome for reasons which we shall explore. Furthermore, there is some controversy as to whether this logical system is adequate to express the reasoning processes used by human beings [Hayes 1977; Collins 1978; Winograd 1978; McCarthy and Hayes 1969]. We briefly describe some aspects of this controversy in Section 12.1.6. Our main purpose is to give the flavor of predicate calculus-based methods by describing briefly how automated inference can proceed with the formulae of predicate calculus expressed in the convenient *clause form.* Clause form is appealing for two reasons. First, it can be represented usefully in relational n-tuple or semantic network notation (Section 12.1.5). Second, the predicate calculus clause and inference system may be easily compared to production systems (Section 12.3).

12.1.1 Clause-Form Syntax (Informal)

In this section we describe the syntax of clause-form predicate calculus sentences. In the next, a more standard nonclausal syntax is described, together with a method for assigning meaning to grammatical logical expressions. Next, we show briefly how to convert from nonclausal to clausal syntax.

A *sentence* is a set of *clauses.* A clause is an ordered pair of sets of *atomic formulae*, or *atoms.* Clauses are written as two (possibly null) sets separated by an arrow, pointing from the *hypotheses* or *conditions* of the clause to its *conclusion.* The *null clause,* whose hypotheses and conclusion are both null, is written □. For example, a clause could appear as

$$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$$

where the A's and B's are atoms. An atom is an expression

$$P(t_1, \ldots, t_j),$$

where P is a predicate symbol which "expects j arguments," each of which must be a *variable*, *constant symbol*, or a *term.* A term is an expression

$$f(t_1, \ldots, t_k)$$

where f is a *function symbol* which "expects k arguments," each of which may be a term. It is convenient to treat constant symbols alone as terms.

A careful (formal) treatment of the syntax of logic must deal with technical issues such as keeping constant and term symbols straight, associating the number of expected arguments with a predicate or function symbol, and assuring an infinite supply of symbols.

For example, the following are sentences of logic.

→ Obscured(Backface(Block1))
Visible(Kidney) →
Road(x), Unpaved(x) → Narrow(x)

12.1.2 Nonclausal Syntax and Logic Semantics (Informal)

Nonclausal Syntax

Clause form is a simplified but logically equivalent form of logic expressions which are perhaps more familiar. A brief review of non-clausal syntax follows.

The concepts of constant symbols, variables, terms, and atoms are still basic. A set of *logical connectives* provides unary and binary operators to combine atoms to form *well-formed formulae* (wffs). If A and B are atoms, then A is a wff, as is $\tilde{}A$ ("not A") $A \Longrightarrow B$ ("A implies B," or "if A then B"), $A \vee B$ ("A or B"), $A \wedge B$ ("A and B"), $A \Longleftrightarrow B$ ("A is equivalent to B," or "A if and only if B"). Thus an example of a wff is

Back(Face) $\vee$ (Obscured(Face)) $\Longrightarrow \tilde{}$ (Visible(Face))

The last concept is that of *universal* and *existential quantifiers*, the use of which is illustrated as follows.

($\forall$ *x*) (wff using "*x*" as a variable).
($\exists$ thing) (wff using "thing" as a variable).

A universal quantifier $\forall$ is interpreted as a conjunction over all domain elements, and an existential quantifier $\exists$ as a disjunction over all domain elements. Hence their usual interpretation as "for each element . . ." and "there exists an element . . . ."

Since a quantified wff is also a wff, quantifiers may be iterated and nested. A quantifier quantifies the "dummy" variable associated with it (x and thing in the examples above). The wff within the *scope* of a quantifier is said to have this quantified variable *bound* by the quantifier. Typically only wffs or clauses all of whose variables are bound are allowed.

Semantics

How does one assign meaning to grammatical clauses and formulae? The semantics of logic formulae (clauses and wffs alike) depends on an *interpretation* and on the meaning of connectives and quantifiers. An interpretation specifies the following.

1. A *domain* of individuals
2. A particular domain element is associated with each constant symbol
3. A function over the domain (mapping k individuals to individuals) is associated with each function symbol.
4. A relation over the domain (a set of ordered k-tuples of individuals) is associated with each predicate symbol.

The interpretation establishes a connection between the symbols in the representation and a domain of discourse (such as the entities one might see in an office or chest x-ray). To establish the truth or falsity of a clause or wff, a value of TRUE or FALSE must be assigned to each atom. This is done by checking in the world of the domain to see if the terms in the atom satisfy the relation specified by the predicate of the atom. If so, the atom is TRUE; if not, it is FALSE. (Of course, the terms, after evaluating their associated functions, ultimately specify individuals). For example, the atom

GreaterThan$(5,\pi)$

is true under the obvious interpretation and false with domain assignments such that

GreaterThan means "Is the author of"
5 means the book *Gone With the Wind*
π means Rin-Tin-Tin.

After determining the truth values of atoms, wffs with connectives are given truth values by using the *truth tables* of Table 12.1, which specify the semantics of the logical connectives. The relation of this formal semantics of connectives with the usual connectives used in language (especially "*implies*") is interesting, and one must be careful when translating natural language statements into predicate calculus.

The semantics of clause form expressions is now easy to explain. A sentence is the *conjunction* of its clauses. A clause

$$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$$

with variables $x_1, \ldots, x_k$ is to be understood

Table 12.1

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \Longrightarrow B$	$A \Longleftrightarrow B$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

$$\forall\, x_1, \ldots, x_k,\ (A_1 \wedge \ldots \wedge A_n) \Longrightarrow (B_1 \vee \ldots \vee B_m).$$

The null clause is to be understood as a contradiction. A clause with no conditions is an assertion that at least one of the conclusions is true. A clause with null conclusion is a denial that the conditions (hypotheses) are true.

12.1.3 Converting Nonclausal Form to Clauses

The conversion of nonclausal to clausal form is done by applying straightforward rewriting rules, based on logic identities (ultimately the truth tables). There is one trick necessary, however, to remove existential quantifiers. *Skolem functions* are used to replace existentially quantified variables, according to the following reasoning.

Consider the wff

$(\forall\, x)((\exists\, y)(\text{Behind}\,(y, x)))$.

With the proper interpretation, this wff might correspond to saying "For any object x we consider, there is another object y which is behind x." Since the $\exists$ is within the scope of the $\forall$, the particular y might depend on the choice of x. The Skolem function trick is to remove the existential quantifier and use a function to make explicit the dependence on the bound universally quantified variable. The resulting wff could be

$(\forall\, x)\ (\text{Behind}(\text{SomethingBehind}(x), x))$

which might be rendered in English: "Any object x has another object behind it; furthermore, some Skolem function we choose to call SomethingBehind determines which object is behind its argument." This is a notational trick only; the existence of the new function is guaranteed by the existential quantification; both notations are equally vague as to the entity the function actually produces.

In general, one must replace each occurrence of an existentially quantified variable in a wff by a (newly created Skolem) function of all the universally quantified variables whose scope includes the existential quantifier being eliminated. If there is no universal quantifier, the result is a new function of no arguments, or a new constant.

$(\exists\, x)(\text{Red}(x))$,

which may be interpreted "Something is red," is rewritten as something like

Red(RedThing)

or

"Something is red, and furthermore let's call it RedThing."

The conversion from nonclausal to clausal form proceeds as follows (for more details, see [Nilsson 1971]). Remove all implication signs with the identity $(A \Longrightarrow B) \Longleftrightarrow ((\char`~ A) \vee B)$. Use DeMorgan's laws (such as $\char`~(A \vee B) \Longleftrightarrow ((\char`~ A) \wedge (\char`~ B))$, and the extension to quantifiers, together with cancellation of double negations, to force negations to refer only to single predicate letters. Rewrite variables to give each quantifier its own unique dummy variable. Use Skolem functions to remove existential quantifiers. Variables are all now universally quantified, so eliminate the quantifier symbols (which remain implicitly), and rearrange the expression into conjunctive normal form (a conjunction of disjunctions.) The $\wedge$'s now connect disjunctive *clauses* (at last!). Eliminate the $\wedge$'s, obtaining from the original expression possibly several clauses.

At this point, the original expression has yielded multiple disjunctive clauses. Clauses in this form may be used directly in automatic theorem provers [Nilsson 1971]. The disjunctive clauses are not quite in the clause form as defined earlier, however; to get clauses into the final form, convert them into implications. Group negated atoms, reexpanding the scope of negation to include them all and converting the $\vee$ of $\tilde{}$'s into a $\tilde{}$ of $\wedge$'s. Reintroduce one implication to go from

$$B_1 \vee B_2 \ldots \vee B_m \vee (\tilde{}(A_1 \wedge A_2 \ldots \wedge A_n))$$

to

$$A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee B_2 \ldots \vee B_m$$

To obtain the final form, replace the connectives (which remain implicitly) with commas.

12.1.4 Theorem Proving

Good accounts of the basic issues of automated theorem proving are given in [Nilsson 1971; Kowalski 1979; Loveland 1978]. The basic ideas are as follows. A sentence is *inconsistent*, or *unsatisfiable*, if it is false in every interpretation. Some trivially inconsistent sentences are those containing the null clause, or simple contradictions such as the same clause being both unconditionally asserted and denied. A sentence that is true in all interpretations is *valid*. Validity of individual clauses may be checked by applying the truth tables unless quantifiers are present, in which case an infinite number of formulae are being specified, and the truth status of such a clause is not algorithmically decidable. Thus it is said that first order predicate calculus is *undecidable*. More accurately, it is *semidecidable*, because any valid wff can be established as such in some (generally unpredictable) finite time. The validation procedure will run forever on invalid formulae; the rub is that one can never be sure whether it is running uselessly, or about to terminate in the next instant.

The notion of a *proof* is bound up with the notion of logical entailment. A clause C *logically follows* from a set of clauses S (we take S to *prove* C) if every interpretation that makes S true also makes C true. A formal proof is a sequence of inferences which establishes that C logically follows from S. In nonclausal predicate logic, inferences are rewritings of axioms and previously established formulae in accordance with *rules of inference* such as

Modus Ponens: From (A) and ($A \Longrightarrow B$) infer (B)
Modus Tollens: From ($\tilde{}B$) and ($A \Longrightarrow B$) infer (A)
Substitution: e.g. From ($\forall x$)(Convex(x)) infer (Convex(Region31))
Syllogisms,

and so forth.

Automatic clausal theorem provers usually try to establish that a clause C logically follows from the set of clauses S. This is accomplished by showing the *unsatisfiability* of S and (C) taken together. This rather backward approach is a technical effect of the way that theorem provers usually work, which is to derive a contradiction.

The fundamental and surprising result that all true theorems are provable in finite time, and an algorithmic (but inefficient) way to find the proof, is due to Herbrand [Herbrand 1930]. The crux of the result is that although the domain of individuals who might participate in an interpretation may be infinite, only a finite number of interpretations need be investigated to establish unsatisfiability of a set of clauses, and in each only a finite number of individuals must be considered. A computationally efficient way to perform automatic inference was discovered by Robinson [Robinson 1965]. In it, a single rule of inference called *resolution* is used. This single rule preserves the *completeness* of the system (all true theorems are provable) and its *correctness* (no false theorems are provable).

The rule of resolution is very simple. Resolution involves matching a condition of one clause A with a conclusion of another clause B. The derived clause, called the *resolvent*, consists of the unmatched conditions and conclusions of A and B instantiated by the matching substitution. *Matching* two atoms amounts to finding a substitution of terms for variables which if applied to the atoms would make them identical.

Theorem proving now means resolving clauses with the hope of producing the empty clause, a contradiction.

As an example, a simple resolution proof goes as follows. Say it is desired to prove that a particular wastebasket is invisible. We know that the wastebasket is behind Brian's desk and that anything behind something else is invisible (we have a simpleminded view of the world in this little example). The givens are the wastebasket location and our naive belief about visibility:

$$\rightarrow \text{Behind(WasteBasket, DeskOf(Brian))} \tag{12.1}$$
$$\text{Behind(object,obscurer)} \rightarrow \text{Invisible(object)} \tag{12.2}$$

Here Behind and Invisible are predicates, DeskOf is a function, Brian and WasteBasket are constants (denote particular specific objects), and object and obscurer are (universally quantified) variables. The negation of the conclusion we wish to prove is

$$\text{Invisible(WasteBasket)} \rightarrow \tag{12.3}$$

or, "Asserting the wastebasket is invisible is contradictory." Our task is to show this set of clauses is inconsistent, so that the invisibility of the wastebasket is proved. The resolution rule consists of matching clauses on opposite sides of the arrow which can be unified by a substitution of terms for variables. A substitution that works is:

Substitute WasteBasket for object and DeskOf(Brian) for obscurer in (12.2).

Then a cancellation can occur between the right side of (12.1) and the left side of (12.2). Another cancellation can then occur between the right side of (12.2) and the left side of (12.3), deriving the empty clause (a contradiction), Quod Erat Demonstrandum.

Anyone who has ever tried to do a nontrivial logic proof knows that there is searching involved in finding which inference to apply to make the proof terminate. Usually human beings have an idea of "what they are trying to prove," and can occasionally call upon some domain semantics to guide which inferences make sense. Notice that at no time in a resolution proof or other formal proof of logic is a specific interpretation singled out; the proof is about all possible interpretations. If deductions are made by appealing to intuitive, domain-dependent, semantic considerations (instead of purely syntactic rewritings), the deduction system is *informal.* Almost all of mathematics is informal by this definition, since normal proofs are not pure rewritings.

Many nonsemantic heuristics are also possible to guide search, such as trying to reduce the differences between the current formulae and the goal formula to be proved. People use such heuristics, as does the Logic Theorist, an early nonclausal, nonresolution theorem prover [Newell et al. 1963].

A basic resolution theorem prover *is* guaranteed to terminate with a proof if one exists, but usually resource limitations such as time or memory place an upper limit on the amount of effort one can afford to let the prover spend. As all the resolvents are added to the set of clauses from which further conclusions may be derived, the question of selecting which clauses to resolve becomes quite a vital one. Much research in automatic theorem proving has been devoted to reducing the search space of derivations for proofs [Nilsson 1980; Loveland 1970]. This has usually been done through heuristics based on formal aspects of the deductions (such as: make deductions that will not increase drastically the number of active clauses). Guidance from domain-dependent knowledge is not only hard to implement, it is directly against the spirit of resolution theorem proving, which attempts to do all the work with a uniform inference mechanism working on uninterpreted symbol strings. A moderation of this view allows the "intent" of a clause to guide its application in the proof. This can result in substantial savings of effort; an example is the treatment of "frame axioms" recommended by Kowalski (Section 13.1.4). Ad hoc, nonformalizable, domain-dependent methods are not usually welcome in automatic theorem-proving circles; however, such heuristics only guide the activity of a formal system; they do not render it informal.

12.1.5 Predicate Calculus and Semantic Networks

Predicate calculus theorem proving may be assisted by the addition of more relational structure to the set of clauses. The structure in a semantic net comes from *links* which connect *nodes*; nodes are accessed by following links, so the availability of information in nodes is determined by the link structure. Links can thus help by providing quick access to relevant information, given that one is "at" a particular node.

Although there are several ways of representing predicate calculus formulae in networks, we adopt here that of [Kowalski 1979; Deliyanni and Kowalski 1979]. The steps are simple:

1. Use a partition to represent the clause.
2. Convert all atoms to binary predicate atoms.
3. Distinguish between conditions and conclusions.

Recall that in Chapter 10, a partition is defined as a set of nodes and arcs in a graph. The internal structure of the partition cannot be determined from outside it. Partitioning extends the structure of a semantic net enough to allow unambiguous representations of all of first order predicate calculus.

The first step in developing the network representation for clauses is to convert each relation to a binary one. We distinguish between conditions and conclusions by using an additional bit of information for each arc. Diagrammatically, an arc is drawn with a double line if it is a condition and a single line if it is a conclusion. Thus the earlier example $S = \{(12.1), (12.2), (12.3)\}$ can be transformed into the network shown in Fig. 12.1.

This figure hints at the advantages of the network embedding for clauses: It is an indexing scheme. This scheme does not indicate which clauses to resolve next but can help reduce the possibilities enormously. If the most recent resolution involved a given clause with a given set of terms, other clauses which also have those terms will be represented by explicit arcs nearby in the network (this would *not* be true if the clauses were represented as a set). Similarly, other clauses involving the same predicate symbols are also nearby being indexed by those symbols. Again, this would not be true in the set representation. Thus the embedded network

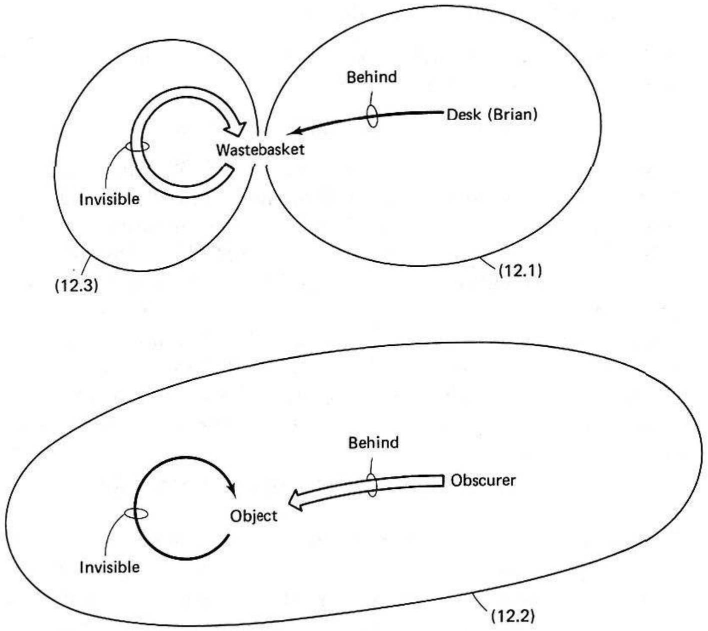


Fig. 12.1 Converting clauses to networks.