likely to be as opaque as any other scheme because of the control-structuring methods that must be imposed on the pure production system form.

12.4 SCENE LABELING AND CONSTRAINT RELAXATION

The general computational problem of assigning labels consistently to objects is sometimes called the "labeling problem," and arises in many contexts, such as graph and automata homomorphism, graph coloring, Latin square generation, and of course, image understanding [Davis and Rosenfeld 1976; Zucker 1976; Haralick and Shapiro 1979]. "Relaxation labeling," "constraint satisfaction," and "cooperative algorithms" are natural implementations for labeling, and their potential parallelism has been a very influential development in computer vision. As should any important development, the relaxation paradigm has had an impact on the conceptualization as well as on the implementation of processes.

Cooperating algorithms to solve the labeling problem are useful in low level vision (e.g., line finding, stereopsis) and in intermediate-level vision (e.g., line-labeling, semantics-based region growing). They may also be useful for the highest-level vision programs, those that maintain a consistent set of beliefs about the world to guide the vision process.

Section 12.4.1 presents the main concepts in the labeling problem. Section 12.4.2 outlines some basic forms that "discrete labeling" algorithms can take. Section 12.4.3 introduces a continuing example, that of labeling lines in a line drawing, and gives a mathematically well-behaved probabilistic "linear operator" labeling method. Section 12.4.4 modifies the linear operator to be more in accord with our intuitions, and Section 12.4.5 describes relaxation as linear programming and optimization, thereby gaining additional mathematical rigor.

12.4.1 Consistent and Optimal Labelings

All labeling problems have the following notions.

1. A set of *objects*. In vision, the objects usually correspond to entities to be labeled, or assigned a "meaning."
2. A finite set of *relations* between objects. These are the sorts of relations we saw in Chapter 10; in vision, they are often geometric or topological relations between segments in a segmented image. Properties of objects are simply unary relations. An input scene is thus a relational structure.
3. A finite set of *labels*, or symbols associated with the "meanings" mentioned above. In the simplest case, each object is to be assigned a single label. A *labeling* assigns one or more labels to (a subset of) the objects in a relational structure. Labels may be weighted with "probabilities"; a (label, weight) pair can indicate something like the "probability of an object having that label."
4. *Constraints*, which determine what labels may be assigned to an object and what sets of labels may be assigned to objects in a relational structure.

A basic labeling problem is then: Given a finite input scene (relational structure of objects), a set of labels, and a set of constraints, find a "consistent labeling." That is, assign labels to objects without violating the constraints. We saw this problem in Chapter 11, where it appeared as a matching problem. Here we shall start with the discrete labeling of Chapter 11 and proceed to more general labeling schemes.

As a simple example, consider the indoor scene of Fig. 12.6. The segmented office scene is to have its regions labeled as Door, Wall, Ceiling, Floor, and Bin, with the obvious interpretation of the labels. Here are some possible constraints, informally stated. Note that these particular constraints are in terms of the input relational structure, not the world from which the structure arose. A more complex (but reasonable) situation arises if scene constraints must be derived from rules about the three dimensional domain of the scene and the imaging process. Unary constraints use object properties to constrain labels; n-ary constraints force sets of label assignments to be compatible.

Unary constraints

1. The Ceiling is the single highest region in the image.
2. The Floor must be checkered.

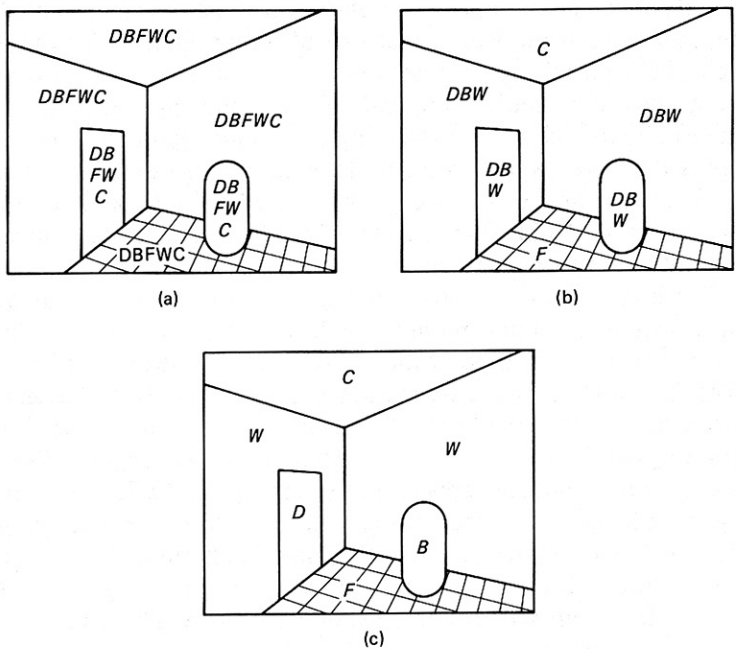


Fig. 12.6 A stylized "segmented office scene." The regions are the objects to be assigned labels D, B, F, W, C (Door, Bin, Floor, Wall, Ceiling). In (a), each object is assigned all labels. In (b) unary constraints have been applied (see text). In (c), relational constraints have been applied, and a unique label for each region results.

3. A Wall is adjacent to the Floor and Ceiling.
4. A Door is adjacent to the Floor and a Wall.
5. A Bin is adjacent to a Floor.
6. A Bin is smaller than a Door.

Obviously, there are many constraints on the appearance of segments in such a scene; which ones to use depends on the available sensors, the ease of computation of the relations and their power in constraining the labeling. Here the application of the constraints (Fig. 12.6) results in a unique labeling. Although the constraints of this example are purely for illustration, a system that actually performs such labeling on real office scenes is described in [Barrow and Tenenbaum 1976].

Labelings may be characterized as *inconsistent* or *consistent.* A weaker notion is that of an *optimal* labeling. Each of these adjectives reflects a formalizable property of the labeling of a relational structure and the set of constraints. If the constraints admit of only completely compatible or absolutely incompatible labels, then a labeling is consistent if and only if all its labels are mutually compatible, and inconsistent otherwise. One example is the line labels of Section 9.5; line drawings that could not be consistently labeled were declared "impossible." Such a black-and-white view of the scene interpretation problem is convenient and neat, but it is sometimes unrealistic. Recall that one of the problems with the line-labeling approach of Chapter 9 is that it does not cope gracefully with missing lines; strictly, missing lines often mean "impossible" line drawings. Such an uncompromising stance can be modified by introducing constraints that allow more degrees of compatibility than two (wholly compatible or strictly incompatible). Once this is done, both consistent and inconsistent labelings may be ranked on compatibility and likelihood. It is possible that a formally inconsistent labeling may rank better than a consistent but unlikely labeling.

Some examples are shown in Fig. 12.7. In 12.7b, the "inconsistent" labels are not nonsensical, but can only arise from (a very unlikely) accidental alignment of convex edges with three of the six vertices of a hexagonal hole in an occluding surface. The vertices that arise are not all included in the traditional catalog of legal vertices, hence the "inconsistent" labeling. The "floating cube" interpretation is consistent, but the "sitting cube" interpretation may be more likely if support and gravity are important concepts in the system. In Fig.12.7c, the scene with a missing line cannot be consistent according to the traditional vertex catalog, but the "inconsistent" labels shown are still the most likely ones. Labelings are only "consistent," "inconsistent," or "optimal" with respect to a given relational structure of objects (an input scene) and a set of constraints. These examples are meant to be illustrative only.

12.4.2 Discrete Labeling Algorithms

Let us consider the problem of finding a consistent set of labels, taken from a discrete finite set. This problem may be placed in an abstract algebraic context [Haralick and Kartus 1978; Haralick 1978; Haralick et al. 1978]. Perhaps the sim-

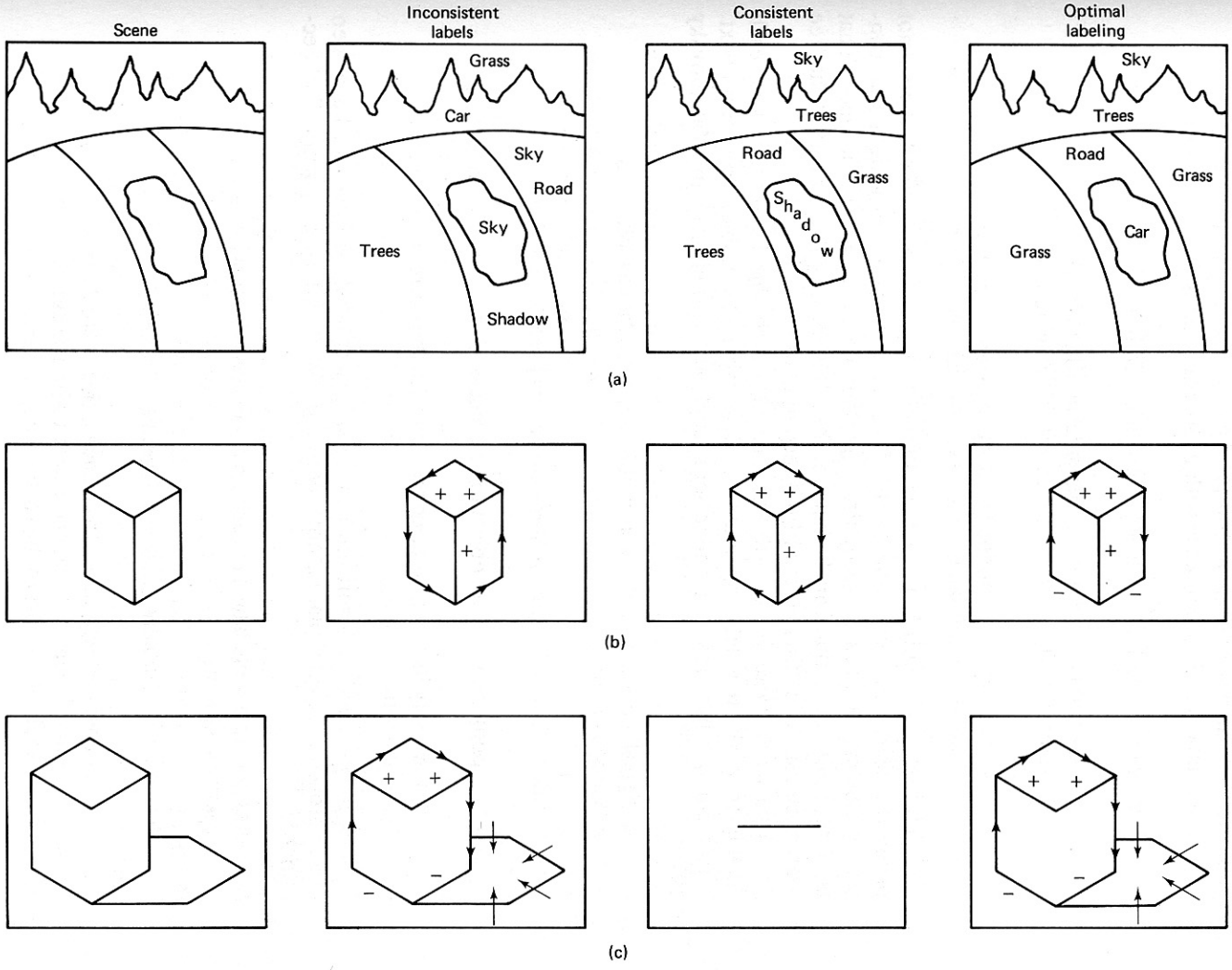


Fig. 12.7 Three scenes (A, B, C) and their labelings. Labelings are only "consistent," "inconsistent," or "optimal" with respect to a given relational structure of objects (an input scene) and a set of constraints. These examples are meant to be illustrative only.

plest way to find a consistent labeling of a relational structure (we shall often say "labeling of a scene") is to apply a depth-first *tree search* of the labeling possibilities, as in the backtracking algorithm (11.1).

Label an object in accordance with unary constraints.

Iterate until a globally consistent labeling is found:

Given the current labeling, label another object consistently—in accordance with all constraints.

If the object cannot be labeled consistently, backtrack and pick a new label for a previously labeled object.

This labeling algorithm can be computationally inefficient. First, it does not prune the search tree very effectively. Second, if it is used to generate all consistent labelings, it does not recognize important independences in the labels. That is, it does not notice that conclusions reached (labels assigned) in part of the tree search are usable in other parts without recomputation.

In a *serial relaxation*, the labels are changed one object at a time. After each such change, the new labeling is used to determine which object to process next. This technique has proved useful in some applications [Feldman and Yakimovsky 1974].

Assign all possible labels to each object in accordance with unary constraints.

Iterate until a globally consistent labeling is found:

Somehow pick an object to be processed.

Modify its labels to be consistent with the current labeling.

A *parallel iterative* algorithm adjusts all object labels at once; we have seen this approach in several places, notably in the "Waltz filtering algorithm" of Section 9.5.

Assign all possible labels to each object in accordance with unary constraints.

Iterate until a globally consistent labeling is found:

In parallel, eliminate from each object's label set those labels that are inconsistent with the current labels of the rest of the relational structure.

A less structured version of relaxation occurs when the iteration is replaced with an *asynchronous interaction* of labeled objects. Such interaction may be implemented with multiple cooperating processes or in a data base with "demons" (Ap-

pendix 2). This method of relaxation was used in MSYS [Barrow and Tenenbaum 1976]. Here imagine that each object is an active process that knows its own label set and also knows about the constraints, so that it knows about its relations with other objects. The program of each object might look like this.

> If I have just been activated, and my label set is not consistent with the labels of other objects in the relational structure, then I change my label set to be consistent, else I suspend myself.
>
> Whenever I change my label set, I activate other objects whose label set may be affected, then I suspend myself.

To use such a set of active objects, one can give each one all possible labels consistent with the unary constraints, establish the constraints so that the objects know where and when to pass on activity, and activate all objects.

Constraints involving arbitrarily many objects (i.e., constraints of arbitrarily high *order*) can efficiently be relaxed by recording acceptable labelings in a graph structure [Freuder 1978]. Each object to be labeled initially corresponds to a node in the graph, which contains all legal labels according to unary constraints. Higher order constraints involving more and more nodes are incorporated successively as new nodes in the graph. At each step the new node constraint is *propagated*; that is, the graph is checked to see if it is consistent with the new constraint. With the introduction of more constraints, node pairings that were previously consistent may be found to be inconsistent. As an example consider the following graph coloring problem: color the graph in Fig. 12.8 so that neighboring nodes have different colors. It is solved by building constraints of increasingly higher order and propagating them. The node constraints are given explicitly as shown in Fig. 12.8a, but the higher-order constraints are given in functional implicit form; prospective colorings must be tested to see if they satisfy the constraints. After the node constraints are given, order two constraints are synthesized as follows: (1) make a node for each node pairing; (2) add all labelings that satisfy the constraint. The result is shown in Fig. 12.8b. The single constraint of order three is synthesized in the same way, but now the graph is inconsistent: the match "Y,Z: Red,Green" is ruled out by the third order legal label set (RGY,GRY). To restore consistency the constraint is propagated through node (Y,Z) by deleting the inconsistent labelings. This means that the node constraint for node Z is now inconsistent. To remedy this, the constraint is propagated again by deleting the inconsistency, in this case the labeling $(Z{:}G)$. The change is propagated to node (X,Z) by deleting $(X,Z$: Red,Green) and finally the network is consistent.

In this example constraint propagation did not occur until constraints of order three were considered. Normally, some constraint propagation occurs after every order greater than one. Of course it may be impossible to find a consistent graph. This is the case when the labels for node Z in our example are changed from (G,Y) to (G,R). Inconsistency is then discovered at order three.

It is quite possible that a discrete labeling algorithm will not yield a unique label for each object. In this case, a consistent labeling exists using each label for the

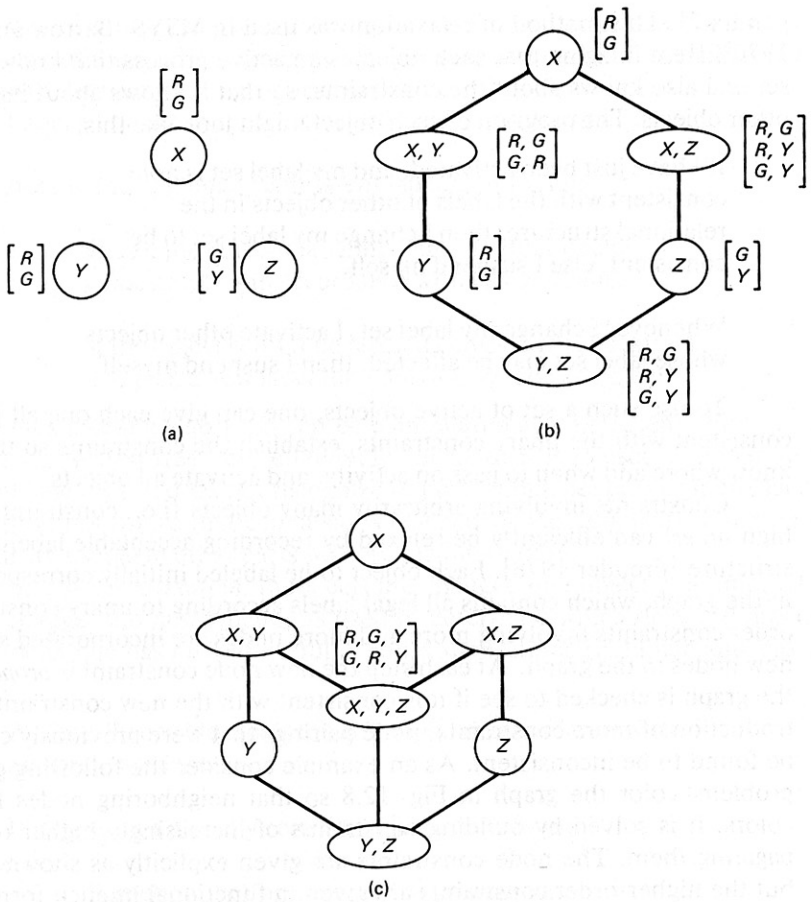


Fig. 12.8 Coloring a graph by building constraints of increasingly higher order.

object. However, which of an object's multiple labels goes with which of another object's multiple labels is not determined. The final enumeration of consistent labelings usually proceeds by tree search over the reduced set of possibilities remaining after the relaxation.

Convergence properties of relaxation algorithms are important; convergence means that in some finite time the labeling will "settle down" to a final value. In discrete labeling, constraints may often be written so that the label adjustment phase always reduces the number of labels for an object (inconsistent ones are eliminated). In this case the algorithm clearly must converge in finite time to a consistent labeling, since for each object the label set must either shrink or stay stable. In schemes where labels are added, or where labels have complex structure (such as real number "weights" or "probabilities"), convergence is often not guaranteed mathematically, though such schemes may still be quite useful. Some probabilistic labeling schemes (Section 12.4.3) have provably good convergence properties.

It is possible to use relaxation schemes without really considering their mathematical convergence properties, their semantics (What is the semantics of weights attached to labels—are they probabilities?), or a clear definition of what exactly the relaxation is to achieve (What is a good set of labels?). The fact that some schemes can be shown to have unpleasant properties (such as assigning nonzero weights to each of two inconsistent hypotheses, or not always converging to a solution), does not mean that they cannot be used. It only means that their behavior is not formally characterizable or possibly even predictable. As relaxation computations become more common, the less formalizable, less predictable, and less conceptually elegant forms of relaxation computations will be replaced by better behaved, more thoroughly understood schemes.

12.4.3 A Linear Relaxation Operator and a Line Labeling Example

The Formulation

We now move away from discrete labeling and into the realm of continuous *weights* or *supposition values* on labels. In Sections 12.4.3 and 12.4.4 we follow closely the development of [Rosenfeld et al. 1976]. Let us require that the sum of label weights for each object be constrained to sum to unity. Then the weights are reminiscent of probabilities, reflecting the "probability that the label is correct." When the labeling algorithm converges, a label emerges with a high weight if it occurs in a probable labeling of the scene. Weights, or supposition values, are in fact hard to interpret consistently as probabilities, but they are suggestive of likelihoods and often can be manipulated like them.

In what follows p refers to probability-like weights (supposition values) rather than to the value of a probability density function. Let a relational structure with n objects be given by a_i, $i = 1, \ldots, n$, each with m discrete labels $\lambda_1, \ldots, \lambda_m$. The shorthand $p_i(\lambda)$ denotes the weight, or (with the above caveats) the "probability" that the label λ (actually λ_k for some k) is correct for the object a_i. Then the probability axioms lead to the following constraints,

$$0 \leqslant p_i(\lambda) \leqslant 1 \tag{12.14}$$

$$\sum_{\lambda} p_i(\lambda) = 1 \tag{12.15}$$

The labeling process starts with an initial assignment of weights to all labels for all objects [consistent with Eqs. (12.14) and (12.15)]. The algorithm is parallel iterative: It transforms all weights at once into a new set conforming to Eqs. (12.14) and (12.15), and repeats this transformation until the weights converge to stable values.

Consider the transformation as the application of an operator to a vector of label weights. This operator is based on the *compatibilities* of labels, which serve as constraints in this labeling algorithm. A compatibility p_{ij} looks like a conditional probability.

$$\sum_{\lambda} p_{ij}(\lambda \mid \lambda') = 1 \qquad \text{for all } i, j, \lambda' \tag{12.16}$$

$$p_{ii}(\lambda|\lambda') = 1 \qquad iff\ \lambda = \lambda', \quad \text{else } 0. \tag{12.17}$$

The $p_{ij}(\lambda|\lambda')$ may be interpreted as the conditional probability that object a_i has label λ given that another object a_j has label λ'. These compatibilities may be gathered from statistics over a domain, or may reflect a priori belief or information.

The operator iteratively adjusts label weights in accordance with other weights and the compatibilities. A new weight $p_i(\lambda)$ is computed from old weights and compatibilities as follows.

$$p_i(\lambda) := \sum_j c_{ij} \{ \sum_{\lambda'} p_{ij}(\lambda|\lambda') p_j(\lambda') \} \tag{12.18}$$

The c_{ij} are coefficients such that

$$\sum_j c_{ij} = 1 \tag{12.19}$$

In Eq. (12.18), the inner sum is the expectation that object a_i has label λ, given the evidence provided by object a_j. $p_i(\lambda)$ is thus a weighted sum of these expectations, and the c_{ij} are the weights for the sum.

To run the algorithm, simply pick the p_{ij} and c_{ij}, and apply Eq. (12.18) repeatedly to the p_i until they stop changing. Equation (12.18) is in the form of a matrix multiplication on the vector of weights, as shown below; the matrix elements are weighted compatibilities, the $c_{ij}p_{ij}$. The relaxation operator is thus a matrix; if it is partitioned into several *component* matrices, one for each set of non-interacting weights, linear algebra yields proofs of convergence properties [Rosenfeld et al. 1976]. The iteration for the reduced matrix for each component does converge, and converges to the weight vector that is the eigenvector of the matrix with eigenvalue unity. This final weight vector is independent of the initial assignments of label weights; we shall say more about this later.

An Example

Let us consider the input line drawing scene of Fig. 12.9a used in [Rosenfeld et.al. 1976]. The line labels given in Section 9.5 allow several consistent labels as shown in Fig. 12.9b-e, each with a different physical interpretation.

In the discrete labelling "filtering" algorithm presented in Section 9.5 and outlined in the preceding section, the relational structure is imposed by the neighbor relation between vertices induced by their sharing a line. Unary constraints are imposed through a catalog of legal combinations of line labels at vertices, and the binary constraint is that a line must not change its label between vertices. The algorithm eliminates inconsistent labels.

Let us try to label the sides of the triangle a_1, a_2, and a_3 in Fig. 12.9 with the solid object edge labels $\{>, <, +, -\}$. To do this requires some "conditional probabilities" for compatibilities $p_{ij}(\lambda|\lambda')$, so let us use those that arise if all eight interpretations of Fig. 12.9 are equally likely. Remembering that

$$p(X|Y) = \frac{p(X,Y)}{p(Y)} \tag{12.20}$$

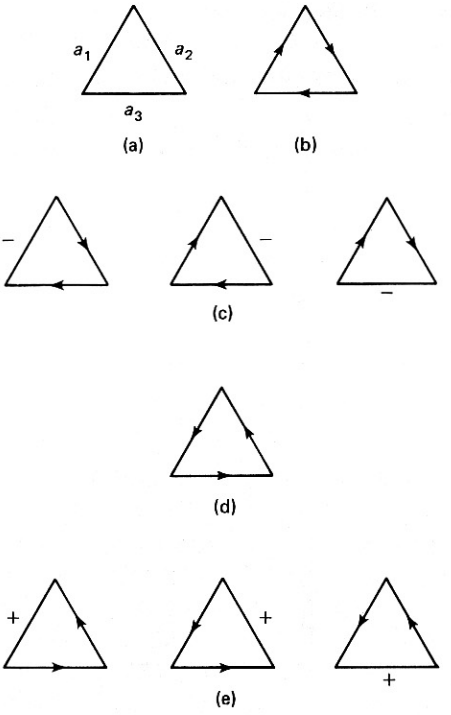


Fig. 12.9 A triangle and its possible labels. (a) Edge names. (b) Floating. (c) Flap folded up. (d) Triangular hole. (e) Flap folded down.

and taking $p(X, Y)$ to mean the probability that labels X and Y occur consecutively in clockwise order around the triangle, one can derive Table 12.2. Of course, we could choose other compatibilities based on any considerations whatever as long as Eqs. (12.16) and (12.17) are preserved.

Table 12.2 shows that there are two noninteracting components, $\{-,>\}$ and $\{+,<\}$. Consider the first component that consists of the weight vector

$$[p_1(>),\ p_1(-),\ p_2(>),\ p_2(-),\ p_3(>),\ p_3(-)] \tag{12.21}$$

The second is treated similarly. This vector describes weights for the subpopulation of labelings given by Fig. 12.9b and c. The matrix M of compatibilities has columns of weighted p_{ij}.

$$M = \begin{bmatrix} c_{11}p_{11}(>|>) & c_{21}p_{21}(>|>) & \cdots \\ c_{11}p_{11}(>|-) & c_{21}p_{21}(>|-) & \cdots \\ c_{12}p_{12}(>|>) & c_{22}p_{22}(>|>) & \cdots \\ c_{12}p_{12}(>|-) & c_{22}p_{22}(>|-) & \cdots \\ c_{13}p_{13}(>|>) & c_{23}p_{23}(>|>) & \cdots \\ c_{13}p_{13}(>|-) & c_{23}p_{23}(>|-) & \cdots \end{bmatrix} \tag{12.22}$$

Table 12.2

λ_1	λ_2	$p(\lambda_1, \lambda_2)$	$p(\lambda_1 \mid \lambda_2)$
>	>	1/4	2/3
>	−	1/8	1
−	>	1/8	1/3
−	−	0	0
>	<	0	0
>	+	0	0
−	<	0	0
−	+	0	0
<	>	0	0
+	>	0	0
<	−	0	0
+	−	0	0
<	<	1/4	2/3
<	+	1/8	1
+	<	1/8	1/3
+	+	0	0

If we let $c_{ij} = 1/3$ for all i, j, then

$$M = \frac{1}{3}\begin{bmatrix} 1 & 0 & 2/3 & 1/3 & 2/3 & 1/3 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 2/3 & 1/3 & 1 & 0 & 2/3 & 1/3 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 2/3 & 1/3 & 2/3 & 1/3 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{12.23}$$

An analytic eigenvector calculation (Appendix 1) shows that the M of Eq. (12.23) yields (for any initial weight vector) the final weight vector of

$$[3/4,\ 1/4,\ 3/4,\ 1/4,\ 3/4,\ 1/4] \tag{12.24}$$

Thus each line of the population in the component we chose (Fig. 12.9b and c) has label > with "probability" 3/4, —with "probability" 1/4. In other words, from an initial assumption that all labelings in Fig. 12.9b and c were equally likely, the system of constraints has "relaxed" to the state where the "most likely" labeling is that of Fig. 12.9b, the floating triangle.

This relaxation method is a crisp mathematical technique, but it has some drawbacks. It has good convergence properties, but it converges to a solution entirely determined by the compatibilities, leaving no room for preferences or local scene evidence to be incorporated and affect the final weights. Further, the algorithm perhaps does not exactly mirror the following intuitions about how relaxation should work.

1. Increase $p_i(\lambda)$ if high probability labels for other objects are compatible with assignment of λ to a_i.
2. Decrease $p_i(\lambda)$ if high probability labels are incompatible with the assignment of λ to a_i.
3. Labels with low probability, compatible or incompatible, should have little influence on $p_i(\lambda)$.

However, the operator of this section decreases $p_i(\lambda)$ the most when other labels have both low compatibility and low probability. Thus it accords with (1) above, but not with (2) or (3). Some of these difficulties are addressed in the next section.

12.4.4 A Nonlinear Operator

The Formulation

If compatibilities are allowed to take on both positive and negative values, then we can express strong incompatibility better and obtain behavior more like (1), (2), and (3) just above. Denote the compatibility of the event "label λ on a_i" with the event "label λ on a_j" by $r_{ij}(\lambda, \lambda')$. If the two events occur together often, r_{ij} should be positive. If they occur together rarely, r_{ij} should be negative. If they are independent, r_{ij} should be 0. The *correlation coefficient* behaves like this, and the compatibilities of this section are based on correlations (hence the the notation r_{ij} for compatibilities). The correlation is defined using the covariance.

$$\text{cov}(X, Y) = p(X, Y) - p(X)p(Y)$$

Now define a quantity σ which is like the standard deviation

$$\sigma(X) = [p(X) - (p(X))^2]^{1/2} \tag{12.25}$$

then the correlation is the normalized covariance

$$\text{cor}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma(X)\sigma(Y)} \tag{12.26}$$

This allows the formulation of an expression precisely analogous to Eq. (12.18), only that r_{ij} instead of p_{ij} is used to obtain a means of calculating the positive or negative change in weights.

$$q_i^{(k)}(\lambda) = \sum_j c_{ij} \left[\sum_{\lambda'} r_{ij}(\lambda, \lambda') p_j^{(k)}(\lambda')\right] \tag{12.27}$$

In Eqs. (12.27)–(12.29) the superscripts indicate iteration numbers. The weight change (Eq. 12.27) could be applied as follows,

$$p_i^{(k+1)}(\lambda) = p_i^{(k)}(\lambda) + q_i^{(k)}(\lambda) \tag{12.28}$$

but then the resultant label weights might not remain nonnegative. Fixing this in a straightforward way yields the iteration equation

$$p_i^{(k+1)}(\lambda) = \frac{p_i^{(k)}(\lambda)[1 + q_i^{(k)}(\lambda)]}{\sum_{\lambda} p_i^{(k)}(\lambda)[1 + q_i^{(k)}(\lambda)]} \tag{12.29}$$

The convergence properties of this operator seem to be unknown, and like the linear operator it can assign nonzero weights to maximally incompatible labelings. However, its behavior can accord with intuition, as the following example shows.

An Example

Computing the covariances and correlations for the set of labels of Fig. 12.9b-e yields Table 12.3.

Figure 12.10 shows the nonlinear operator of Eq. (12.29) operating on the example of Fig. 12.9. Figure 12.10 shows several cases.

1. Equal initial weights: convergence to apriori probabilities ($\frac{3}{8}$, $\frac{3}{8}$, $\frac{1}{8}$, $\frac{1}{8}$).
2. Equal weights in the component $\{>,-\}$: convergence to "most probable" floating triangle labeling.
3. Slight bias toward a flap labeling is not enough to overcome convergence to the "most probable" labeling, as in (2).
4. Like (3), but greater bias elicits the "improbable" labeling.
5. Contradicatory biases toward "improbable" labelings: convergence to "most probable" labeling instead.
6. Like (5), but stronger bias toward one "improbable" labeling elicits it.
7. Bias toward one of the components $\{>,-\}$, $\{<,+\}$ converges to most probable labeling in that component.
8. Like (7), only biased to less probable labelling in a component.

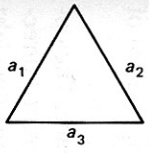
12.4.5 Relaxation as Linear Programming

The Idea

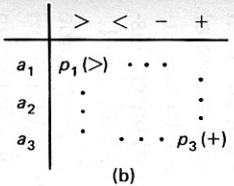
Linear programming (LP) provides some useful metaphors for thinking about relaxation computations, as well as actual algorithms and a rigorous basis [Hummel and Zucker 1980]. In this section we follow the development of [Hinton 1979].

Table 12.3

λ_1	λ_2	$\text{cov}(\lambda_1, \lambda_2)$	$\text{cor}(\lambda_1, \lambda_2)$
$>$	$>$	$\frac{7}{64}$	$\frac{7}{15}$
$>$	$-$	$\frac{5}{64}$	$5/\sqrt{105}$
$-$	$>$	$\frac{5}{64}$	$5/\sqrt{105}$
$-$	$-$	$-\frac{1}{64}$	$-\frac{1}{7}$
$>$	$<$	$-\frac{9}{64}$	$-\frac{3}{5}$
.	.	.	.
.	.	.	.
.	.	.	.



(a)



(b)

Case	Initial weights				After 2 to 3 iterations				After 20 to 30 iterations				Limit			
	0.25	0.25	0.25	0.25	0.3	0.3	0.2	0.2	0.33	0.33	0.17	0.17	0.37	0.37	0.13	0.13
(1)	0.25	0.25	0.25	0.25	0.3	0.3	0.2	0.2	0.33	0.33	0.17	0.17	0.37	0.37	0.13	0.13
	0.25	0.25	0.25	0.25	0.3	0.3	0.2	0.2	0.33	0.33	0.17	0.17	0.37	0.37	0.13	0.13
	0.5	0	0.5	0	0.8	0	0.2	0	0.98	0	0.2	0	1	0	0	0
(2)	0.5	0	0.5	0	0.8	0	0.2	0	0.98	0	0.2	0	1	0	0	0
	0.5	0	0.5	0	0.8	0	0.2	0	0.98	0	0.2	0	1	0	0	0
	0.5	0	0.5	0	0.62	0	0.37	0	1	0	0	0	1	0	0	0
(3)	0.4	0	0.6	0	0.49	0	0.51	0	0.97	0	0.03	0	1	0	0	0
	0.5	0	0.5	0	0.62	0	0.37	0	1	0	0	0	1	0	0	0
	0.5	0	0.5	0	0.64	0	0.36	0	1	0	0	0	1	0	0	0
(4)	0.3	0	0.7	0	0.36	0	0.64	0	0.07	0	0.93	0	0	0	1	0
	0.5	0	0.5	0	0.64	0	0.36	0	1	0	0	0	1	0	0	0
	0.3	0	0.7	0	0.5	0	0.5	0	0.95	0	0.05	0	1	0	0	0
(5)	0.3	0	0.7	0	0.5	0	0.5	0	0.95	0	0.05	0	1	0	0	0
	0.5	0	0.5	0	0.84	0	0.16	0	1	0	0	0	1	0	0	0
	0.2	0	0.8	0	0.3	0	0.7	0	0.06	0	0.94	0	0	0	1	0
(6)	0.3	0	0.7	0	0.51	0	0.49	0	1	0	0	0	1	0	0	0
	0.5	0	0.5	0	0.83	0	0.17	0	1	0	0	0	1	0	0	0
	0.3	0.2	0.3	0.2	0.41	0.13	0.32	0.14	0.98	0	0.02	0	1	0	0	0
(7)	0.3	0.2	0.3	0.2	0.41	0.13	0.32	0.14	0.98	0	0.02	0	1	0	0	0
	0.3	0.2	0.3	0.2	0.41	0.13	0.32	0.14	0.98	0	0.02	0	1	0	0	0
	0.3	0.2	0.3	0.2	0.38	0.17	0.29	0.16	1	0	0	0	1	0	0	0
(8)	0.25	0.25	0.25	0.25	0.35	0.20	0.25	0.20	1	0	0	0	1	0	0	0
	0.2	0.2	0.4	0.2	0.23	0.16	0.45	0.16	0.2	0	0.8	0	0	0	1	0

(c)

Fig. 12.10 The nonlinear operator produces labelings for the triangle in (a). (b) shows how the label weights are displayed, and (c) shows a number of cases (see text).

To put relaxation in terms of linear programming, we use the following translations.

- LABEL WEIGHT VECTORS $\Longrightarrow$ POINTS IN EUCLIDEAN N-SPACE. Each possible assignment of a label to an object is a *hypothesis*, to which a weight (supposition value) is to be attached. With N hypotheses, an N-vector of weights describes a labeling. We shall call this vector a (hypothesis or label) *weight vector*. For m labels and n objects, we need at most Euclidean nm-space.
- CONSTRAINTS $\Longrightarrow$ INEQUALITIES. Constraints are mapped into *linear inequalities* in hypothesis weights, by way of various identities like those of "fuzzy logic" [Zadeh 1965]. Each inequality determines an infinite half-space. The weight vectors within this half-space satisfy the constraint. Those outside do not. The convex solid that is the set intersection of all the half-spaces includes those weight vectors that satisfy all the constraints: each represents a "consistent" labeling. In linear programming terms, each such weight vector is a *feasible solution.* We thus have the usual geometric interpretation of the linear programming problem, which is to find the best (optimal) consistent (feasible) labeling (solution, or weight vector). Solutions should have *integer*-valued (1- or 0-valued) weights indicating convergence to actual labelings, not probabilistic ones such as those of Section 12.4.3, or the one shown in Fig. 12.10c, case 1.
- HYPOTHESIS PREFERENCES $\Longrightarrow$ PREFERENCE VECTOR. Often some hypotheses (label assignments) are preferred to others, on the basis of a priori knowledge, image evidence, and so on. To express this preference, make an N-dimensional *preference vector*, which expresses the relative importance (preference) of the hypotheses. Then
    - The *preference of a labeling* is the dot product of the preference vector and the weight vector (it is the sum for all hypotheses of the weight of each hypothesis times its preference).
    - The preference vector defines a *preference direction* in N-space. The optimal feasible solution is that one "farthest" in the preference direction. Let $\mathbf{x}$ and $\mathbf{y}$ be feasible solutions; they are N-dimensional weight vectors satisfying all constraints. If $\mathbf{z} = \mathbf{x} - \mathbf{y}$ has a component in the positive preference direction, then $\mathbf{x}$ is a better solution than $\mathbf{y}$, by the definition of the preference of a labeling.

  It is helpful for our intuition to let the preference direction define a "downward" direction in N-space as gravity does in our three-space. Then we wish to pick the lowest (most preferred) feasible solution vector.
- LABELING $\Longrightarrow$ OPTIMAL SOLUTION. The relaxation algorithm must solve the linear programming problem—find the best consistent labeling. Under the conditions we have outlined, the best solution vector occurs generally at a vertex of the N-space solid. This is so because usually a vertex will be the "lowest" part of the convex solid in the preference direction. It is a rare coincidence that the solid "rests on a face or edge," but when it does a whole edge or face of the solid contains equally preferred solutions (the preference direction is normal to

the edge or face). For integer solutions, the solid should be the convex hull of integer solutions and not have any vertices at noninteger supposition values.

The "simplex algorithm" is the best known solution method in linear programming. It proceeds from vertex to vertex, seeking the one that gives the optimal solution. The simplex algorithm is not suited to parallel computation, however, so here we describe another approach with the flavor of hill-climbing optimization. Basically, any such algorithm moves the weight vector around in N-space, iteratively adjusting weights. If they are adjusted one at a time, serial relaxation is taking place; if they are all adjusted at once, the relaxation is parallel iterative. The feasible solution solid and the preference vector define a "cost function" over all N-space, which acts like a potential function in physics. The algorithm tries to reach an optimum (minimum) value for this cost function. As with many optimization algorithms, we can think of the algorithm as trying to simulate (in N-space) a ball bearing (the weight vector) rolling along some path down to a point of minimum gravitational (cost) potential. Physics helps the ball bearing find the minimum; computer optimization techniques are sometimes less reliable.

Translating Constraints to Inequalities

The supposition values, or hypothesis weights, may be encoded into the interval [0, 1], with 0 meaning "false," 1 meaning "true." The extension of weights to the whole interval is reminiscent of "fuzzy logic," in which truth values may be continuous over some range [Zadeh 1965]. As in Section 12.4.3, we denote supposition values by $p(\cdot)$; H, A, B, and C are label assignment events, which may be considered as hypotheses that the labels are correctly assigned. $\tilde{}$, $\vee$, $\wedge$, $\Longrightarrow$ and $\Longleftrightarrow$ are the usual logical connectives relating hypotheses. The connectives allow the expression of complex constraints. For instance, a constraint might be "Label x as 'y' if and only if z is labeled 'w' or q is labelled 'v'." This constraint relates three hypotheses: h_1: (x is "y"), h_2: (z is "w"), h_3: (q is "v"). The constraint is then $h_1 \Longleftrightarrow (h_2 \vee h_3)$.

Inequalities may be derived from constraints this way.

1. *Negation.* $p(H) = 1 - p(\tilde{}(H))$.
2. *Disjunction.* The sums of weights of the disjunct are greater than or equal to one. $p(A \vee B \vee \ldots \vee C)$ gives the inequality $p(A) + p(B) + \ldots + p(C) \geqslant 1$.
3. *Conjunction.* These are simply separate inequalities, one per conjunct. In particular, a conjunction of disjunctions may be dealt with conjunct by conjunct, producing one disjunctive inequality per conjunct.
4. *Arbitrary expressions.* These must be put into conjunctive normal form (Chapter 10) by rewriting all connectives as $\wedge$'s and $\vee$'s. Then (3) applies.

As an example, consider the simple case of two hypotheses A and B, with the single constraint that $A \Longrightarrow B$. Applying rules 1 through 4 results in the following five inequalities in $p(A)$ and $p(B)$; the first four assure weights in [0, 1]. The fifth arises from the logical constraint, since $A \Longrightarrow B$ is the same as $B \vee \tilde{}(A)$.

$$0 \leqslant p(A)$$
$$p(A) \leqslant 1$$
$$0 \leqslant p(B)$$
$$p(B) \leqslant 1$$
$$p(B) + (1 - p(A)) \geqslant 1 \qquad \text{or} \qquad p(B) \geqslant p(A)$$

These inequalities are shown in Fig. 12.11. As expected from the $\Longrightarrow$ constraint, optimal feasible solutions exist at: (1,1) or (A,B); (0,1) or $(\tilde{}(A),B)$; (0,0) or $(\tilde{}(A), \tilde{}(B))$. Which of these is preferred depends on the preference vector. If both its components are positive, (A,B) is preferred. If both are negative, $(\tilde{}(A), \tilde{}(B))$ is preferred, and so on.

A Solution Method

Here we describe (in prose) a search algorithm that can find the optimal feasible solution to the linear programming problem as described above. The description makes use of the mechanical analogy of an N-dimensional solid of feasible solutions, oriented in N-space so that the preference vector induces a "downward" direction in space. The algorithm attempts to move the vector of hypothesis weights to the point in space representing the feasible solution of maximum preference. It should be clear that this is a point on the surface of the solid, and unless the preference vector is normal to a face or edge of the solid, the point is a unique "lowest" vertex.

To establish a potential that leads to feasible solutions, one needs a measure of the infeasibility of a weight vector for each constraint. Define the amount a vector violates a constraint to be zero if it is on the feasible side of the constraint hyperplane. Otherwise the violation is the normal distance of the vector to the hyperplane. If $\mathbf{h}_i$ is the coefficient vector of the i^{th} hyperplane (Appendix 1) and $\mathbf{w}$ the weight vector, this distance is

$$d_i = \mathbf{w} \cdot \mathbf{h}_i \tag{12.30}$$

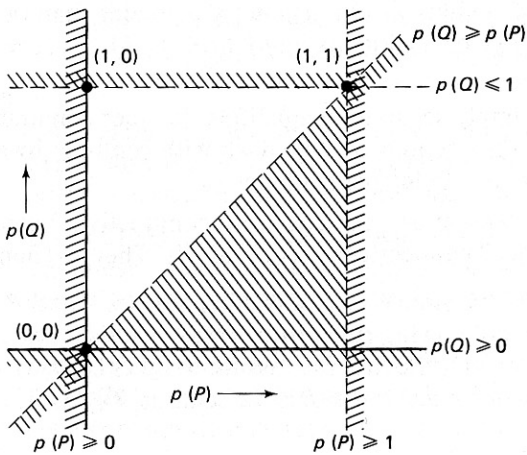


Fig. 12.11 The feasible region for two hypotheses A and B and the constraint A B. Optimal solutions may occur at the three vertices. The preferred vertex will be that one farthest in the direction of the preference vector, or lowest if the preference vector defines "down."

If we then define the infeasibility as

$$I = \sum_i \frac{d_i^2}{2} \tag{12.31}$$

then $\partial I/\partial d_i = d_i$ is the rate the infeasibility changes for changes in the violation. The force exerted by each constraint is proportional to the normal distance from the weight vector to the feasible region defined by that constraint, and tends to pull the weight vector onto the surface of the solid.

Now add a weak "gravity-like" force in the preference direction to make the weight vector drift to the optimal vertex. At this point an optimization program might perform as shown in Fig. 12.12.

Figure 12.12 illustrates a problem: The forces of preference and constraints will usually dictate a minimum potential outside the solid (in the preference direction). Fixes must be applied to force the weight vector back to the closest (presumably the optimum) vertex. One might round high weights to 1 and low ones to 0, or add another local force to draw vectors toward vertices.

Examples

An algorithm based on the principles outlined in the preceeding section was successfully used to label scenes of "puppets" such as Fig. 12.13 with body parts [Hinton 1979].

The discrete, consistency-oriented version of line labeling may be extended to incorporate the notion of optimal labelings. Such a system can cope with the explosive increase in consistent labelings that occurs if vertex labels are included for cases of missing lines, accidental alignment, or "two-dimensional" objects such as folded paper. It allows modeling of the fact that human beings do not "see" all possible interpretations of scenes with accidental alignments. If labelings are given

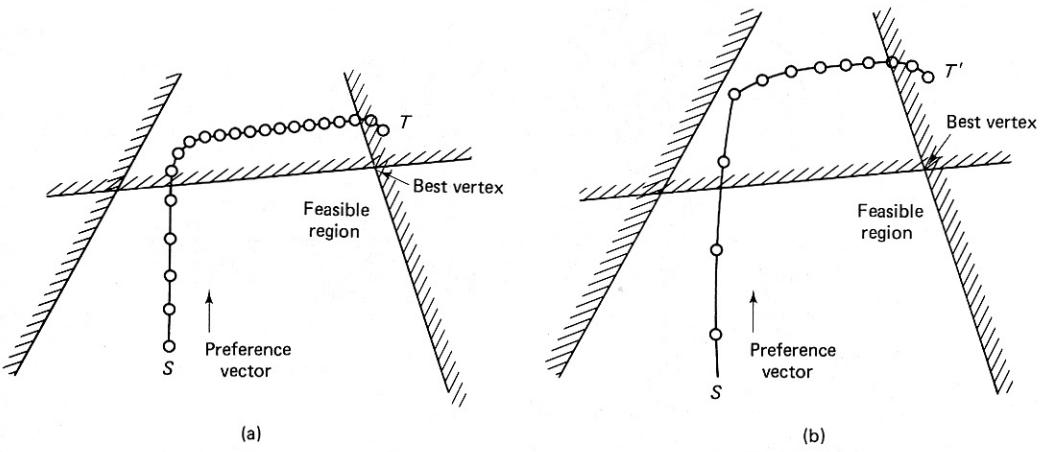
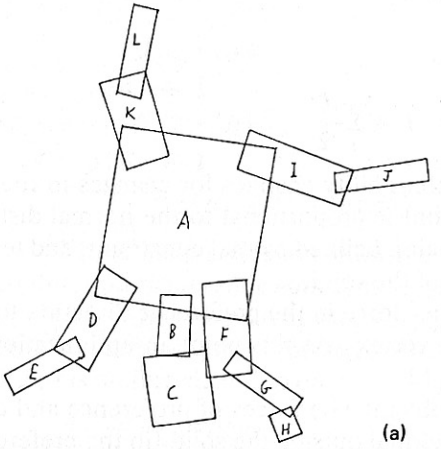


Fig. 12.12 In (a), the weight vector moves from S to rest at T, under the combined influence of the preferences and the violated constraints. In (b), convergence is speeded by making stronger preferences, but the equilibrium is farther away from the optimal vertex.

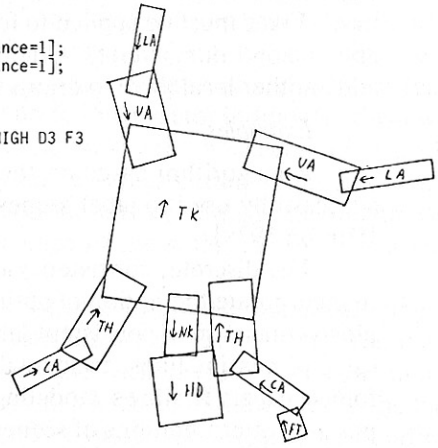



```
!.bestset;
A1  BOT TRUNK  NECK B1  UPPERARM D2 F2  THIGH 13 K2
B1  BOT NECK  HEAD C1  TRUNK A1
C1  BOT HEAD  NECK B1
D2  TOP UPPERARM  TRUNK A1  LOWERARM E4
E4  TOP LOWERARM  UPPERARM D2  HAND -
F2  TOP UPPERARM  TRUNK A1  LOWERARM G2
G2  TOP LOWERARM  UPPERARM F2  HAND H2
H2  TOP HAND  LOWERARM G2
I3  TOP THIGH  TRUNK A1  CALF J4
J4  BOT CALF  THIGH I3  FOOT -
K2  BOT THIGH  TRUNK A1  CALF L4
L4  BOT CALF  THIGH K2  FOOT -
```

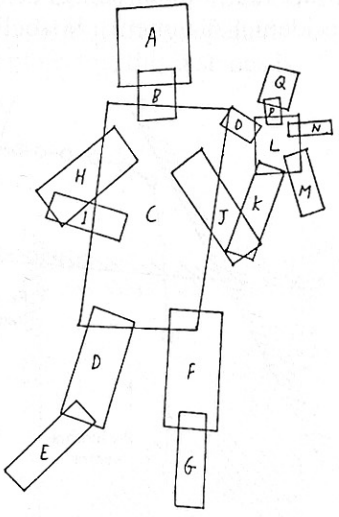
(a)

```
!trytointerpret [trunk as upright importance=1];
!trytointerpret [thigh as upright importance=1];

!.bestset;
A2  TOP TRUNK  NECK -  UPPERARM I2 K1  THIGH D3 F3
B1  BOT NECK  HEAD C1  TRUNK -
C1  BOT HEAD  NECK B1
D3  TOP THIGH  TRUNK A2  CALF E3
E3  TOP CALF  THIGH D3  FOOT -
F3  TOP THIGH  TRUNK A2  CALF E3
G3  TOP CALF  THIGH F3  FOOT H1
H1  TOP FOOT  CALF G3
I2  TOP UPPERARM  TRUNK A2  LOWERARM J3
J3  BOT LOWERARM  UPPERARM I2  HAND -
K1  BOT UPPERARM  TRUNK A2  LOWERARM L3
L3  BOT LOWERARM  UPPERARM K1  HAND -
```



(b)




```
!.bestset;
A1  TOP HEAD  NECK B1
B1  TOP NECK  HEAD A1  TRUNK C2
C2  TOP TRUNK  NECK B1  UPPERARM H1 J1  THIGH D3 F3
D3  TOP THIGH  TRUNK C2  CALF E3
E3  TOP CALF  THIGH D3  FOOT -
F3  TOP THIGH  TRUNK C2  CALF G3
G3  TOP CALF  THIGH F3  FOOT-
H1  TOP UPPERARM  TRUNK C2  LOWERARM I1
I1  TOP LOWERARM  UPPERARM H1  HAND -
J1  TOP LOWERARM  TRUNK C2  LOWERARM K4
K4  BOT LOWERARM  UPPERARM J1  HAND L6
L6  BOT HAND  LOWERARM K4
```

(c)

Fig. 12.13 Puppet scenes interpreted by linear programming relaxation. (a) shows an upside down puppet. (b) is the same input along with preferences to interpret the trunk and thighs as upright; these result in an interpretation with trunk and neck not connected. In (c), the program finds only the "best" puppet, since it was only expecting one.

costs, then one can include labels for missing lines and accidental alignment as high-cost labels, rendering them usable but undesirable. Also, in a scene-analysis system using real data, local evidence for edge appearance can enhance the a priori likelihood that a line should bear a particular label. If such preferences can be extracted along with the lines in a scene, the evidence can be used by the line labeling algorithm.

The inconsistency constraints for line labels may be formalized as follows. Each line and vertex has one label in a consistent labeling; thus for each line L and vertex J,

$$\sum_{\text{all line labels}} p(L \text{ has label LLABEL}) = 1 \tag{12.32}$$

$$\sum_{\text{all vertex labels}} p(J \text{ has label VLABEL}) = 1 \tag{12.33}$$

Of course, the VLABELS and LLABELS in the above constraints must be forced to be compatible (if L has LLABEL, JLABEL must agree with it). For a line L and a vertex J at its end,

$$p(L \text{ has LLABEL}) = \sum_{\substack{\text{all VLABELS} \\ \text{giving LLABEL to } L}} p(J \text{ has label VLABEL}) \tag{12.34}$$

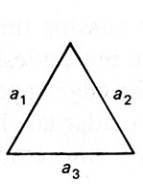
This constraint also enforces the coherence rule (a line may not change its label betwen vertices).

Using these constraints, linear programming relaxation labeled the triangle example of Fig. 12.7 as shown in Fig. 12.14, which shows three cases.

1. Preference 0.5 for each of the three junction label assignments (hypotheses) corresponding to the floating triangle, 0 preference for all other junction and line label hypotheses: converges to floating triangle.
2. Like (1), but with equal preferences given to the junction labels for the triangular hole interpretation, 0 to all other preferences.
3. Preference 3 to the convex edge label for a 2 overrides the three preferences of 1/2 for the floating triangle of case (1). All preferences but these four were 0.

Some Extensions

The translation of constraints to inequalities described above does not guarantee that they produce a set of half-spaces whose intersection is the convex hull of the feasible integer solutions. They can produce "noninteger optima," for which supposition values are not forced to 1 or 0. This is reminiscent of the behavior of the linear relaxation operator of Section 12.4.3, and may not be objectionable. If it is, some effort must be expended to cope with it. Here is an example



(a)

	>	<	−	+
$p(a_1 = >)$				
$p(a_3 = +)$				

(b)

Case	After 10 iterations				After 20 iterations				After 30 to 40 iterations			
(1)	0.65	0.22	0.01	0.14	0.90	0.07	0	0.04	0.99	0	0	0
	0.65	0.22	0.01	0.14	0.90	0.07	0	0.04	0.99	0	0	0
	0.65	0.22	0.01	0.14	0.90	0.07	0	0.04	0.99	0	0	0
(2)	0.39	0.89	0	0	0.14	0.95	0	0	0	0.99	0	0
	0.39	0.89	0	0	0.14	0.95	0	0	0	0.99	0	0
	0.39	0.89	0	0	0.14	0.95	0	0	0	0.99	0	0
(3)	0.56	0.48	0	0.05	0.81	0.23	0	0	0.99	0	0	0
	0	0.34	0	0.99	0	0.15	0	0.99	0	0	0	0.99
	0.56	0.48	0	0.05	0.81	0.23	0	0	0.99	0	0	0

(c)

Fig. 12.14 As in Fig. 12.10, the triangle of (a) is to be assigned labels, and the changing label weights are shown for three cases in (c) using the format of (b). Supposition values for junction labels were used as well, but are not shown. All initial supposition values were 0.

of the problem. Assume three logical constraints, $\tilde{}(A \wedge B)$, $\tilde{}(B \wedge C)$, and $\tilde{}(C \wedge A)$. Suppose A, B, and C have equal preferences of unity (the preference vector is $(1, 1, 1)$). Translating the constraints yields

$$\begin{aligned} p(A) + p(B) &\leqslant 1 \\ p(B) + p(C) &\leqslant 1 \\ p(C) + p(A) &\leqslant 1 \end{aligned} \tag{12.35}$$

The best feasible solution has a total preference of 1½, and is

$$p(A) = p(B) = p(C) = \tfrac{1}{2} \tag{12.36}$$

Here the "best" solution is outside the convex hull of the integer solutions (Fig. 12.15).

The basic way to ensure integer solutions is to use stronger constraints than those arising from the simple rules given above. These may be introduced at first, or when some noninteger optimum has been reached. These stronger constraints are called *cutting planes*, since they cut off the noninteger optima vertices. In the example above, the obvious stronger constraint is

$$p(A) + p(B) + p(C) \leqslant 1 \tag{12.37}$$

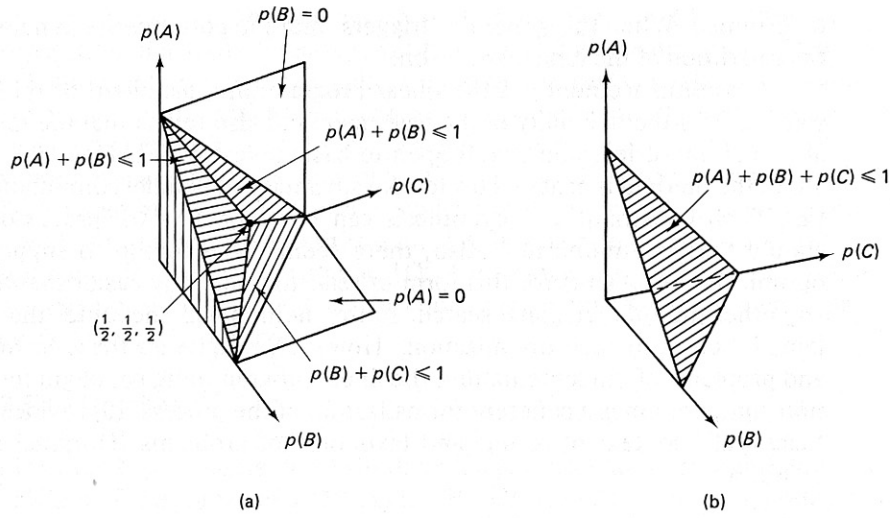


Fig. 12.15 (a) shows part of the surface of the feasible solid with constraints $\neg (A \ \& \ B)$, $\neg (B \ \& \ C)$, $\neg (C \ \& \ A)$, and the non-integer vertex where the three halfspaces intersect. (b) shows a cutting plane corresponding to the constraint "at most one of A, B, or C" that removes the non-integer vertex.

which says that at most one of A, B, and C is true (this is a logical consequence of the logical constraints). Such cutting planes can be derived as needed, and can be guaranteed to eliminate all noninteger optimal vertices in a finite number of cuts [Gomory 1968; Garfinkel and Nemhauser 1972]. Equality constraints may be introduced as two inequality constraints in the obvious way: This will constrain the feasible region to a plane.

Suppose that one desires "weak rules," which are usually true but which can be broken if evidence demands it? For each constraint arising from such a rule, add a hypothesis to represent the situation where the rule is broken. This hypothesis is given a negative preference depending on the strength of the rule, and the constraint enhanced to include the possibility of the broken rule. For example, if a weak rule gives the constraint $P \vee Q$, create a hypothesis H equivalent to $\tilde{}(P \vee Q) = (\tilde{}(P) \wedge \tilde{}(Q))$, and replace the constraint with $P \vee Q \vee H$. Then by "paying the cost" of the negative preference for H, we can have neither P nor Q true.

Hypotheses can be created as the algorithm proceeds by having demon-like "generator hypotheses." The demon watches the supposition value of the generator, and when it becomes high enough, runs a program that generates explicit hypotheses. This is clearly useful; it means that all possible hypotheses do not need to be generated in advance of any scene investigation. The generator can be given a preference equal to that of the best hypotheses that it can generate.

Relaxation sometimes should determine a real number (such as the slope of a line) instead of a truth value. A generator-like technique can allow the method to refine the value of real-valued hypotheses. Basically, the idea is to assign a (Boolean-valued) generator hypothesis to a range of values for the real value to be

determined. When this generator triggers, more hypotheses are generated to get a finer partition of the range, and so on.

The enhancements to the linear programming paradigm of relaxation give some idea of the flexibility of the basic idea, but also reveal that the method is not at all cut-and-dried, and is still open to basic investigation. One of the questions about the method is exactly how to take advantage of parallel computation capabilities. Each constraint and hypothesis can be given its own processor, but how should they communicate? Also, there seems little reason to suppose that the optimization problems for this form of relaxation are any easier than they are for any other multidimensional search, so the method will encounter the usual problems inherent in such optimization. However, despite all these technical details and problems of implementation, the linear programming paradigm for the relaxation computation is a coherent formalization of the process. It provides a relatively "classical" context of results and taxonomy of problems [Hummel and Zucker 1980].

12.5 ACTIVE KNOWLEDGE

Active knowledge systems [Freuder 1975] are characterized by the use of procedures as the elementary units of knowledge (as opposed to propositions or data base items, for instance). We describe how active knowledge might work, because it is a logical extreme of the procedural implementation of propositions. In fact, this style of control has not proven influential; some reasons are given below.

Active knowledge is notionally parallel and heterarchical. Many different procedures can be active at the same time depending on the input. For this reason active knowledge is more easily applied to belief maintenance than to planning; it is very difficult to organize sequential activity within this discipline. Basically, each procedure is responsible for a "chunk" of knowledge, and knows how to manage it with respect to different visual inputs. Control in an active knowledge system is completely distributed. Active knowledge can also be viewed as an extension of the constraint relaxation problem; powerful procedures can make arbitrary detailed tests of the consistency between constraints.

Each piece of active knowledge (program module) knows which other modules it depends on, which depend on it, which it can complain to, and so forth. Thus the choice of "what to do next" is contained in the modules and is not made by an exterior executive.

We describe HYPER, a particular active knowledge system design which illustrates typical properties of active knowledge [Brown 1975]. HYPER provides a less structured mechanism for construction and exploration of hypotheses than does LP-relaxation. Using primitive control functions of the system, the user may write programs for establishing hypotheses and for using the conclusions so reached. The programs are "procedurally embedded" knowledge about a problem domain (e.g. how events relate one to another, what may be conjectured or inferred from a clue, or how one might verify a hypothesis).

When HYPER is in use on a particular task in a domain, hypotheses are created, or instantiated, on the basis of low-level input, high-level beliefs, or any