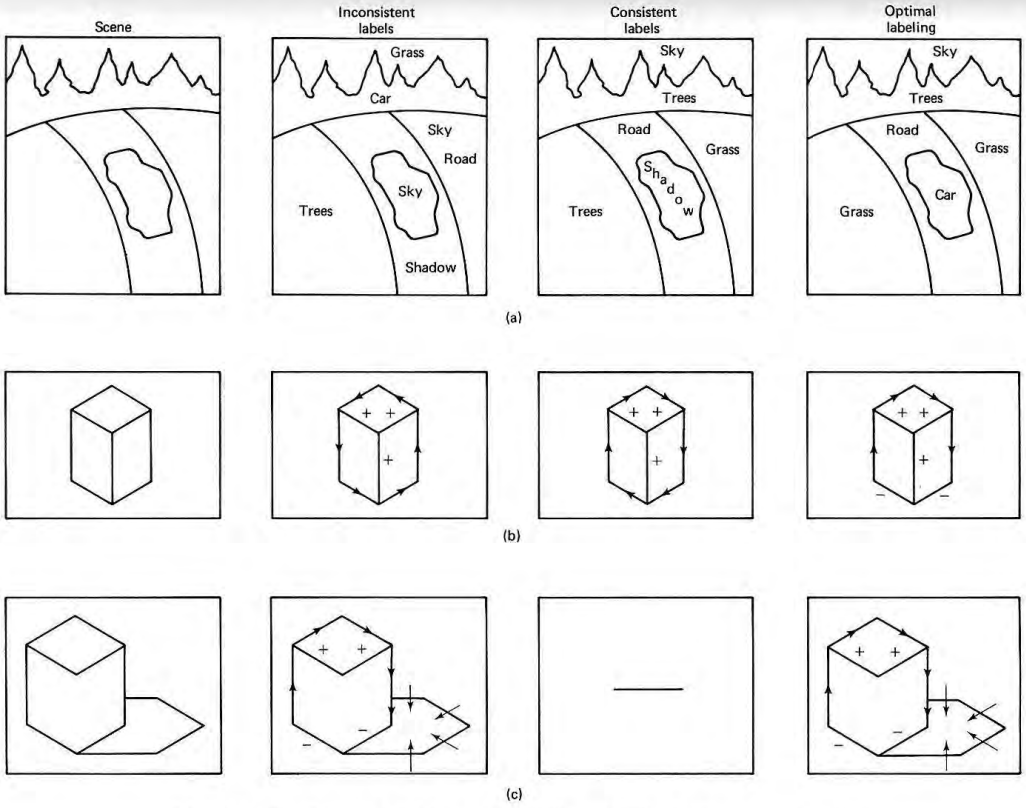

Fig. 12.7 Three scenes (A, B, C) and their labelings. Labelings are only "consistent," "inconsistent," or "optimal" with respect to a given relational structure of objects (an input scene) and a set of constraints. These examples are meant to be illustrative only.

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

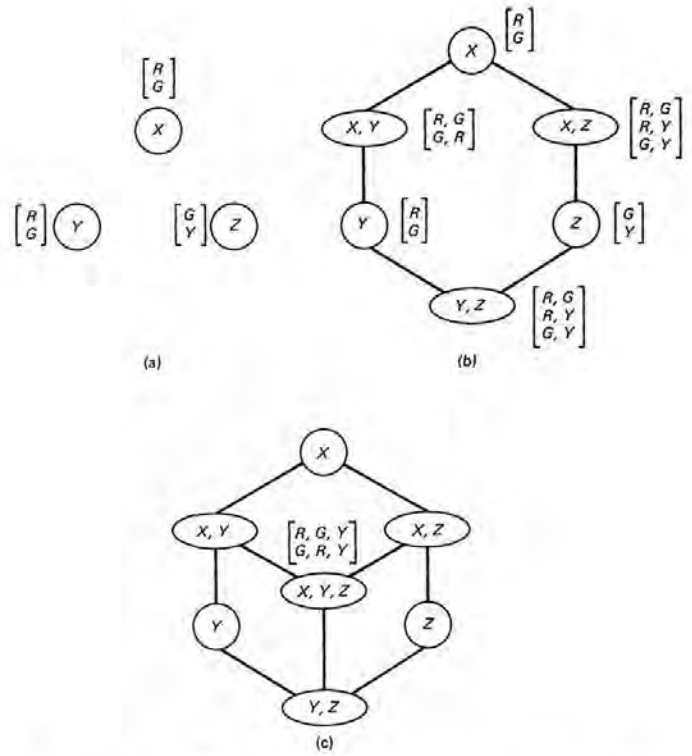

Fig. 12.8 Coloring a graph by building constraints of increasingly higher order.

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.

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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.