Chapter 26

RECOGNITION BY RELATIONS BETWEEN TEMPLATES

An object may have internal degrees of freedom, which mean that its appearance is highly variable — for example, people can move arms and legs; fish deform to swim; snakes wriggle; etc. This phenomenon can make template matching extremely difficult, because one may require either a classifier with a very flexible boundary (and a lot of examples) or many different templates.

Many of these objects have small components which will have a fairly orderly appearance. We could try to match these components as templates, and then determine what objects are present by looking for suggestive relationships between the templates that have been found. For example, instead of finding a face by looking for a single complete face template, we could find one by looking for eyes, nose and a mouth that all lie in an appropriate configuration.

This approach has several possible advantages. Firstly, it may be easier to learn an eye template than it is to learn a face template, because the structure could be simpler. Secondly, it may be possible to obtain and use relatively simple probability models. This is because there may be some independence properties that can be exploited. Thirdly, we may be able to match a very large number of objects with a relatively small number of templates. Animal faces are a good example of this phenomenon — pretty much all animals with recognisable faces have eyes, nose and a a mouth, but with slightly different spatial layouts. Finally, it means that the simple individual templates can be used to construct complex objects. For example, people can move their arms and legs around, and it appears to be much more difficult to learn a single, explicit template for finding whole people than to obtain individual templates for bits of people and a probability model that describes their degrees of freedom.

This topic is not yet well enough understood for there to be a standard approach. However, the main issue — how does one encode a set of relationships between templates in a form that is easily managed? — is quite clear. In this chapter, we

explore a series of different approaches to this problem. Firstly, we could allow each template to vote for the objects that it could represent, and then count the votes in some way (section 26.1). We could place more weight on the specifics of the spatial relations by building some explicit probability model. This could come from the likelihood; in essence, we need a probability distribution function that has a high value when the components are configured like the object, and a low value otherwise. Finding objects then becomes a matter of searching for templates that can be plugged into the probability model to get a high value (section 26.2). Pruning the search requires care; we show one approach in section 26.3. The difficulty with this approach is that, even when pruned, the search could be very expensive. A particular class of probability model allows very efficient search. We introduce these models in section 26.4, and describe two applications (section 26.5 and section 26.6).

26.1 Finding Objects by Voting on Relations between Templates

Very simple object models can result in quite effective recognition. The simplest model is to think of an object as a collection of image **patches** — small image neighbourhoods of characteristic appearance — of several different types, forming an image **pattern**. To tell what pattern is present in an image, we find each patch that is present, and allow it to vote for every pattern in which it appears. The pattern in the image is the one with the most votes. While this strategy is simple, it is quite effective. We will sketch methods for finding patches, and then describe a series of increasingly sophisticated versions of the strategy.

26.1.1 Describing Image Patches

Small image patches can have a quite characteristic appearance, usually when they have many non-zero derivatives (e.g. at corners). We describe a system due to Schmid and Mohr that takes advantage of this property [Schmid and Mohr, 1996; Schmid and Mohr, 1997c]. They find image corners — often called **interest points** (see [Schmid *et al.*, 2000]). They then estimate a set of derivatives of the image grey-level at those corners, and evaluate a set of functions of the image derivatives that are invariant to rotation, translation, some scaling, and illumination changes. These features are called **invariant local jets**. Describing these features in detail would take us out of our way. The value of this approach is that, because the combinations are invariant, we expect that they will take the same value on different views of an object.

We now assume that the image patches fall into a number of classes. We can obtain representatives for each class by having multiple pictures of each object — typically, corresponding patches will be of the same class, but probably have somewhat different invariant local jets, as a result of image noise. We can determine an appropriate set of classes either by classifying the patches by hand, or by clustering example patches (a somewhat better method!). We need to be able to tell when two sets of invariant local jets represent the same class of image patch. Schmid and

Mohr test the Mahalanobis distance between the feature vectors of a patch to be tested and an example patch; if it is below some threshold, the patch being tested is the same as the example.

Notice that this is a classifier — it is allocating patches to classes represented by the examples, or deciding not to classify them — and that the patches are templates. We could build a template matcher with any of the techniques of chapter 25 without using features that are invariant to rotation, etc. To do this, we would use a training set that contained rotated and scaled versions of each example patch, under varying illuminant conditions, so the classifier could learn that rotation, scaling and illuminant changes don't affect the identity of the patch. The advantage of using invariant features is that the classifier doesn't need to learn this invariance from the training set.

26.1.2 Voting and a Simple Generative Model

For a given image, we find the interest points and classify the image patch at each image point. Now, what pattern lies in the image? we can answer this question by constructing a correspondence between image patches and patterns. Assume that there are N_i patches in the image. Furthermore, we assume that there is either a single pattern from our collection in the image, or there is no pattern there. An individual patch could have come either from whatever pattern is present, or from noise. However, patterns typically do not contain every class of patch. This means that asserting that a particular pattern is present is the same as asserting that some of the image patches came from noise (because only one pattern can be present, and these images patches belong to classes that are not in the current pattern).

We now have a (very simple) generative model for an image. When a pattern is present, it produces patches of some classes, but not others. By elaborating this model, we obtain a series of algorithms for matching patterns to images.

The simplest version of this model is obtained by assuming that a pattern produces all patches of the classes that it can produce, and then require that as few patches as possible come from noise. This assumption boils down to voting. We take each image patch, and record a vote for every pattern that is capable of producing that class of patch. The pattern with the most votes, wins, and we say that this pattern is present. This strategy can be effective, but has some problems (figure 26.1).

26.1.3 Probabilistic Models for Voting

We can interpret our simple voting process in terms of a probabilistic model. This is worth doing, because it will cast some light on the strengths and weaknesses of the approach. Our generative model can be made probabilistic by assuming that the patches are produced independently and at random, assuming that the object is present. Let us write

$$P\{\text{patch of type } i \text{ appears in image}|j\text{'th pattern is present}\} = p_{ij}$$

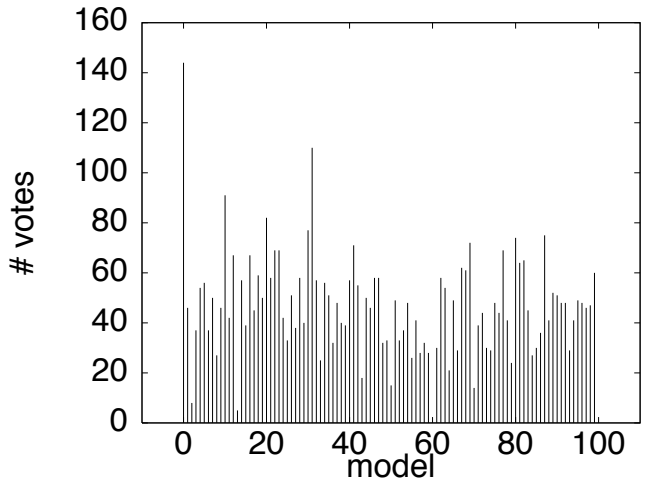

Figure 26.1. The graph shows the number of votes for each pattern recorded for a particular image under the simple voting scheme. Notice that, while the correct match (model # 0) receives the maximum number of votes, three other candidates receives more than half as many votes. Figure from "Local grayvalue invariants for image retrieval," by C. Schmid and R. Mohr, IEEE Trans. Pattern Analysis and Machine Intelligence, 1997 © 1997, IEEE

and

$$P\{\text{patch of type } i|\text{no pattern is present}\} = p_{ix}$$

In the very simplest model, we assume that, for each pattern j, we assume that $p_{ij} = \mu$ if the pattern can produce this patch and 0 otherwise. Furthermore, we assume that $p_{ix} = \lambda < \mu$ for all i. Finally, we assume that each observed patch in the image can come from either a single pattern or from noise. There are a total of n_i patches in the image. Under these assumptions, we need only know which patches came from a pattern and which from noise to compute a likelihood value. In particular, The likelihood of the image, given a particular pattern, and assuming that n_p patches came from that pattern and $n_i - n_p$ patches come from noise, is

$$P(\text{interpretation}|\text{pattern}) = \lambda^{n_p}\mu^{(n_i - n_p)}$$

and this value is larger for larger values of n_p. However, because not every pattern can produce every image patch, the maximum available choice of n_p is *dependent on the pattern we choose*. Our voting method is equivalent to choosing the pattern with the maximum possible likelihood under this (very simple) generative model.

This suggests the source of some difficulties: if the pattern is unlikely, we should take that (prior) information into account. Furthermore, noise may be able to produce some patches more easily than others — ignoring this fact can confuse the vote. Finally, some patches may be more likely given an object than others. For

example, corners appear much more often on a checkerboard pattern than they do on a zebra stripe pattern.

Elaborating the Generative Model

Dealing with these issues in the framework of our current model — that the patches occur independently given that the pattern is present — is relatively simple. Assume that there are N different types of patch. We now assume that each different type of patch is generated with a different probability by different patterns, and by noise. Now assume that there are n_{ij} instances of the j'th type of patch in the image. Furthermore, n_k of these are generated by the pattern, and the rest are generated by noise.

The likelihood function for the l'th pattern is

$$P\left(\begin{array}{c|c} \begin{array}{c} n_1 \text{ of type 1 from pattern,} \\ \ldots, \\ n_N \text{ patches of type } N \text{ from pattern} \\ \text{and } n_{i1} - n_1 \text{ of type 1 from noise,} \\ \ldots, \\ n_{iN} - n_N \text{ from noise} \end{array} & j\text{'th pattern} \end{array}\right)$$

Now because the patches arise independently given the pattern and the noise is independent of the pattern, this likelihood is

$$P(\text{patches from pattern}|j\text{'th pattern})P(\text{patches from noise})$$

The first term is

$$P(\text{type 1}|j\text{'th pattern})^{n_1}P(\text{type 2}|j\text{'th pattern})^{n_2}\ldots P(\text{type } N|j\text{'th pattern})^{n_N}P(\text{noise})$$

which is evaluated as

$$p_{1j}^{n_1}p_{2j}^{n_2}\ldots p_{Nj}^{n_N}$$

We now assume that patches that arise from noise do so independently of one another. This means we can write the noise term as

$$P(\text{type 1}|\text{noise})^{(n_{i1}-n_1)}\ldots P(\text{type } N|\text{noise})^{(n_{iN}-n_N)}$$

which is evaluated as

$$p_{1x}^{(n_{i1}-n_1)}\ldots p_{Nx}^{(n_{iN}-n_N)}$$

This means that the likelihood can be written out as

$$p_{1j}^{n_1}p_{2j}^{n_2}\ldots p_{Nj}^{n_N}p_{1x}^{(n_{i1}-n_1)}\ldots p_{Nx}^{(n_{iN}-n_N)}$$

There are two cases for each type of patch k; if $p_{kj} > p_{kx}$, then this is maximised by $n_k = n_{ik}$, otherwise it is maximised by $n_k = 0$. We write π_j for the prior probability that the image contains the j'th pattern and π_0 for the prior probability it contains

no object. This means that for each object type j, the maximal value of the posterior will look like

$$\left(\prod_m p_{mj}^{n_{im}} \prod_l p_{lx}^{n_{il}}\right) \pi_j$$

where m runs through features for which $p_{mj} > p_{mx}$ and l runs through features for which $p_{lj} < p_{lx}$. We could form this value for each object, and choose the one with the highest posterior value. Notice that this *is* a relational model, though we're not actually computing geometric features linking the templates. This is because the patches are linked by the conditional probability that they occur, given a pattern, *which is different from pattern to pattern.* You should compare this model with the face detector of section 25.2.2; the probabilistic models are identical in spirit.

26.1.4 Voting on Relations

We can use geometric relations to improve on the simple voting strategy fairly easily. A patch should match to an object only if it there are nearby patches that also match to the object, and are in an appropriate configuration. The term "appropriate configuration" can be a source of difficulties, but for the moment assume that we are matching objects up to *plane* rotation, translation and scale (this is a reasonable assumption for such things as frontal faces).

Now assume that we have a patch that matches to some object. We now take the p nearest patches, and check firstly that more than 50% of them match to the same objects, and secondly that the angles between the triples of matching patches are the same as the corresponding angles on the object — these are referred to as **semilocal constraints** (see figure 26.2). If these two tests are passed, we register a vote for that object from the patch. You should compare this strategy — which is due to Schmid and Mohr [Schmid and Mohr, 1997c] — quite closely with geometric hashing, section 24.4.2. It is rather harder to construct a probabilistic interpretation for this approach — the emission probabilities of the patches now depend on *where* they are in the pattern, as well as identity of the pattern that generated them.

26.1.5 Voting and 3D Objects

Although we described Schmid and Mohr's approach in terms of 2D patterns, it can be extended fairly easily to 3D object recognition. We do this by regarding each of a series of views of the object as a different 2D pattern. Given enough views, this will work, because the small changes in the 2D pattern caused by a slight change in viewing angle will be compensated for by the available error range in the process that matches invariant local jets and angles.

This strategy for turning 3D recognition into 2D matching applies quite generally, but comes with difficulties. The main problem is the very large number of models that result, which can make the voting procedure difficult. It isn't known what the minimum number of views required for matching in a scheme like this is. Figure 26.4 illustrates a matching result obtained using this approach.

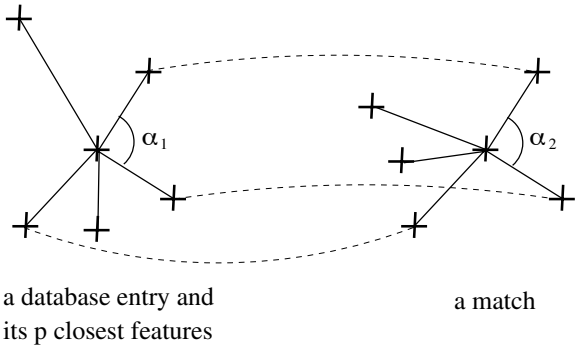

Figure 26.2. Instead of voting for patches that match, we can vote for collections of patches. In this approach, we register a vote only if a patch matches to a particular object, and a percentage of its neighbours match too, and the angles between triples of matching patches have appropriate values, as the figure illustrates. This strategy significantly reduces the number of random matches, as figure 26.3 indicates. Figure from "Local gray-value invariants for image retrieval," by C. Schmid and R. Mohr, IEEE Trans. Pattern Analysis and Machine Intelligence, 1997 © 1997, IEEE

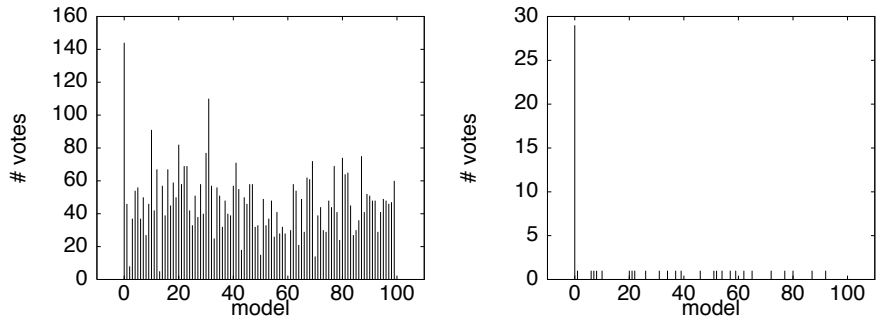

Figure 26.3. In practice, the use of semilocal constraints greatly increases the discriminating power of the original voting scheme. The figure on the **left** shows the number of votes recorded for each model under the original voting scheme, for a particular image. Although the correct match (model # 0) receives the maximum number of votes, three other candidates receives more than half as many votes. When semilocal constraints are added (**right**), the correct match stands out much more clearly. Figure from "Local gray-value invariants for image retrieval," by C. Schmid and R. Mohr, IEEE Trans. Pattern Analysis and Machine Intelligence, 1997 © 1997, IEEE

26.2 Relational Reasoning using Probabilistic Models and Search

The previous section explored methods that assumed that templates were conditionally independent given the pattern. This assumption is a nonsense for most objects (though it can work extremely well in practice) because there are usually

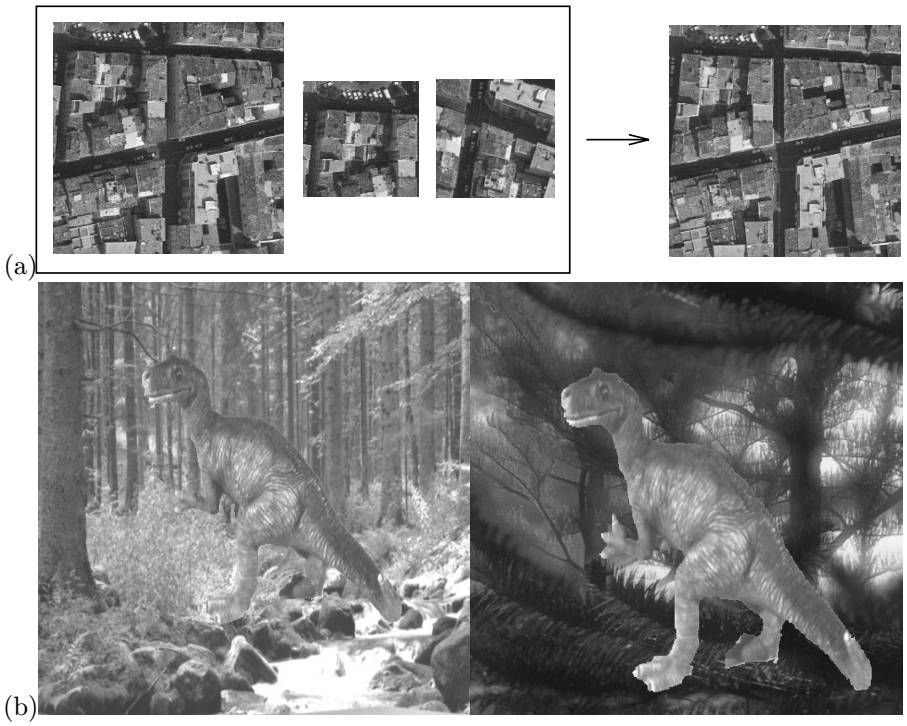

Figure 26.4. Recognition results: (a) image matching in aerial photo interpretation: the image on the right is correctly retrieved using any of the images on the left; (b) three-dimensional object recognition: a toy dinosaur is correctly recognized in both images, despite a large amount of background clutter. Figure from "Local grayvalue invariants for image retrieval," by C. Schmid and R. Mohr, IEEE Trans. Pattern Analysis and Machine Intelligence, 1997 © 1997, IEEE

quite strong relations between features. For example, there are very seldom more than two eyes, one nose and one mouth in a face; the distance between the eyes is roughly the same as that from the bridge of the nose to the mouth; the line joining the eyes is roughly perpendicular to the line from bridge of nose to mouth; etc. We incorporated some these constraints into a voting strategy in section 26.1.4, but didn't really indicate any principled framework within which they can be exploited.

This remains a difficult problem. We need to build models that represent what is significant and allow for efficient inference. We can't present current orthodoxy on this subject, because it doesn't exist; instead, we will describe the main issues and approaches.

26.2.1 Correspondence and Search

In this section, we will explore the core issues in using a probabilistic model for matching. The general approach is as follows. We obtain an interpretation for the image, and compute the value of the posterior probability of the interpretation given the image. We accept interpretations with a sufficiently large value of the posterior probability.

This account hides a myriad of practical problems. The most basic is that we do not know which image information comes from objects and which comes from noise. Generally, we will evade this difficulty and turn our problem into a correspondence problem by matching templates to the image, and then reasoning about the relations between templates. For example, if we wish to find faces, we will apply a series of different detectors — say eye, nose and mouth detectors — to the image, and then look for configurations that are suggestive.

Correspondence

This brings us to the second important problem, which is correspondence. We can't evaluate a posterior (or joint) probability density if we don't know the values of each variable. This means that we must, in essence, engage in a process that hypothesizes that one response is the left eye, another the right eye, a third the nose and a fourth the mouth, and *then* evaluates the posterior. It is clearly important to manage this search very carefully indeed, so that we do not have to look at all possible correspondences.

The techniques of chapter 24 transfer relatively easily; we need to apply a cloak of probability, but the basic line of reasoning will remain. The most basic fact of object recognition — for rigid objects, a small number of correspondences can be used to generate a large number of correspondences — translates easily into the language of probability easily, where it reads: for rigid objects, correspondences are not independent.

In general, our probability model will evaluate the joint probability density for some set of variables. We call a set of values for these variables an **assembly** (other terms are a **group**, or an **hypothesis**). An assembly consists of a set of detector outputs — each may respond with position, or position and orientation, or even more — and a label for each output. These labels give the correspondences, and the labelling is important: if, say, the eye-detector does not differentiate between a left eye and a right eye, we will have to label the eye responses left and right, but there is no point in labelling a mouth detector response as an eye.

It is not possible to form and test all assemblies, because there are usually far too many. For example, assume we have eye detectors — which don't differentiate between left and right eyes — nose detectors and mouth detectors, and a face consists of two eyes, a nose and a mouth. If there are N_e responses from eye detectors, N_n responses from nose detectors and N_m responses from mouth detectors, we would have $O(N_e^2 N_n N_m)$ assemblies to look at. If you read chapter 24 carefully, you should be convinced this is a wild overestimate — the whole point of the chapter is

that quite small numbers of correspondences predict other correspondences.

Incremental Assembly and Search

This suggests thinking of the matching process in the following way. We will assemble collections of detector responses, labelling them as necessary to establish their role in the final assembly. The assembly process will be incremental — we will expand small assemblies to make big ones. We will take each working assembly, and then determine whether: it can be pruned; it can be accepted *as is*; or it should be expanded. Any particular assembly consists of a set of pairs, each of which is a detector response and a label. Some available labels may not have a detector response associated with them. Generally, a correspondence search will have a working collection of hypotheses, which may be quite large. The search involves: taking some correspondence hypothesis, attaching some new pairs, and then either accepting the result as an object, removing it from consideration entirely, or returning it to the pool of working hypotheses.

There are three important components in a correspondence search:

- **When to accept an hypothesis:** if a correspondence hypothesis is sufficiently good, it may be possible to stop expanding it.
- **What to do next:** in a correspondence search, we have to determine which correspondence hypothesis to work on next. Generally, we would wish to work on an hypothesis that is likely to succeed. Usually, it is easier to determine what *not* to do next.
- **When to prune an hypothesis:** if there is no set of detector responses that can be associated with any subset of the empty labels such that the resulting hypothesis would pass the classifier criterion, then there is no point in expanding that search.

Setting up the Search Problem

We will continue to work with the face model, to illustrate the ideas. Assume that the left eye detector responds at $\boldsymbol{x}_1$, the right eye detector responds at $\boldsymbol{x}_2$, the mouth detector responds at $\boldsymbol{x}_3$ and the nose detector responds at $\boldsymbol{x}_4$; we assume that we believe the face is at $\boldsymbol{F}$, and that all other detector responses are due to noise. Furthermore, we assume that we are determining whether there is either a single face or none in the image (or window!). This involves comparing the value of

$$P(\text{one face at } \boldsymbol{F} | \boldsymbol{X}_{\text{le}} = \boldsymbol{x}_1, \boldsymbol{X}_{\text{re}} = \boldsymbol{x}_2, \boldsymbol{X}_{\text{m}} = \boldsymbol{x}_3, \boldsymbol{X}_{\text{n}} = \boldsymbol{x}_4, \text{all other responses})$$

with

$$P(\text{no face} | \boldsymbol{X}_{\text{le}} = \boldsymbol{x}_1, \boldsymbol{X}_{\text{re}} = \boldsymbol{x}_2, \boldsymbol{X}_{\text{m}} = \boldsymbol{x}_3, \boldsymbol{X}_{\text{n}} = \boldsymbol{x}_4, \text{all other responses})$$

Stopping a Search: Detection

Assume that noise responses are independent of the presence of a face (which is fairly plausible). We can then write

$$\begin{aligned}P(\text{one face at } \boldsymbol{F}|\boldsymbol{X}_{\text{le}} = \boldsymbol{x}_1, \boldsymbol{X}_{\text{re}} = \boldsymbol{x}_2, \boldsymbol{X}_{\text{m}} = \boldsymbol{x}_3, \boldsymbol{X}_{\text{n}} = \boldsymbol{x}_4, \text{all other responses}) = \\ P(\text{one face at } \boldsymbol{F}|\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4)P(\text{all other responses}) \propto \\ P(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{one face at } \boldsymbol{F})P(\text{all other responses})P(\text{one face at } \boldsymbol{F})\end{aligned}$$

(where we have suppressed some notation!). We can classify particular groups of detector responses as coming from a face or from noise by comparing the posterior that this configuration comes from a face with the posterior that it comes from noise. In particular, we will compare

$$P(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{one face at } \boldsymbol{F})$$

with

$$(P(\text{noise responses})P(\text{no face})/P(\text{one face at } \boldsymbol{F})) \text{ (term in relative loss)}$$

(recall that there are only two options — face or no face — and check chapter 7 again if the remark seems puzzling). Thus, the likelihood is the main term of interest. This tells us whether a complete assembly represents a face; but an incomplete assembly could represent a face, too. We can score configurations that lack features, too. This involves determining the posterior that a face is present given only some features, and comparing that with the posterior that the features arose from noise.

When a group of features satisfies the classification criterion (that the posterior that a face is present exceeds the posterior that it is not), we can certainly stop searching. It may not be necessary to observe all possible features to determine that a face is present. If a configuration is strongly suggestive of a face and unlikely to have arisen from noise, then we may wish to assert that a face is present and stop searching at that point.

We illustrate with an example; assume we wish to determine whether a right eye, a mouth and a nose represent a face. To evaluate the joint, we will need to evaluate a series of noise terms and the term

$$P(\boldsymbol{X}_{\text{le}} = \text{missing}, \boldsymbol{X}_{\text{re}} = \boldsymbol{x}_2, \boldsymbol{X}_{\text{m}} = \boldsymbol{x}_3, \boldsymbol{X}_{\text{n}} = \boldsymbol{x}_4|\text{one face at } \boldsymbol{F})$$

using our model. This requires a model to explain how a feature went missing; the simplest is to assume that the detector did not respond (a variety of others are possible), and that this failure to respond is independent of the other feature detector responses. This yields

$$\begin{aligned}P(\text{missing}, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{face}) = \\ \int P(\text{le does not respond}|\boldsymbol{X}_1)P(\boldsymbol{X}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{face})d\boldsymbol{X}_1\end{aligned}$$

(again, suppressing some notation). Now if the probability of detector failure is independent of feature position (which is the usual case in vision applications), we have

$$P(\text{missing}, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{face}) = P(\text{le does not respond})P(\boldsymbol{X}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4|\text{face})d\boldsymbol{X}_1$$

This *does not mean* that we can prune a search if a configuration doesn't represent a face when it has a missing feature. This is because there might be some position for the missing feature that did represent a face; as section 26.3 will show, to prune we need a bound. If we do decide that an object is present without matching all components, this is because either (a) missing components didn't get detected, but the configuration of the components found is so distinctive it doesn't matter or (b) we haven't yet found the other components, but the configuration of the other detectors is so distinctive it doesn't matter. Notice that in case (b), we may still want to look for possible responses for the other detectors; exercise **??** discusses how deciding that an object is present changes this search process.

26.2.2 Example: Finding Faces

Perona and colleagues have built a series of face finders that use various forms of probability model [Burl and Perona, 1996; Weber *et al.*, 2000; Leung *et al.*, 1995a]. Each follows broadly the line of incremental search amongst assemblies, identifying assemblies that should be expanded. Features are obtained from the outputs of various filters, as in figure 26.5. We sketch the system of [Leung *et al.*, 1995a].

An assembly is represented by a vector of distances between features. This vector is assumed to have a Gaussian probability distribution, conditioned on the presence of a face. Missing elements can be dealt with by marginalising out the relevant elements of this distribution. An attractive feature of this model is that, once one has a sufficiently large assembly, the position of new features can be predicted by (a) identifying distances that would lead to sufficiently large values of the conditional probability and (b) predicting a range of feature positions from these distances (figure 26.5).

26.3 Using Classifiers to Prune Search

Assume that we have an assembly that consists of a right eye, a mouth and a nose. If we can determine from the probability model that there is no possible position for the left eye that would result in an assembly that is acceptable as a face, then there is no point in trying to grow our assembly. Recall that, if we had a position for the left eye, we would test to see whether the likelihood was greater than some function of the number of noise responses, the priors, and the relative losses. In particular, there is some fixed value that the likelihood must exceed before we can assert that a face is present. If we can assert that *there is no value of* $\boldsymbol{x}_1$ that would result in a likelihood that is over threshold, the search can be pruned.

Figure 26.5. Perona *et al.*'s face detector looks for local patterns, which are classified on their appearance using filter responses. Appropriate arrangements of these patterns are faces. Relations between face points are represented by the inter-element distances, and the class-conditional density for these distances is gaussian. In turn, this means that, once some feature responses have been found, the position of others can be predicted. The figure on the **left** illustrates predictions for different feature points. The features are points around the eyes, nose and mouth, and the variation in inter-point distances comes from individual variations in facial structure. The figure on the **right** illustrates the behaviour of the overall face detector; the column on the left shows the best face in the image, and the column in the right shows the second best, which typically has a posterior value that is smaller by an order of magnitude. *Figure from, "Finding faces in cluttered scenes using random labelled graph matching," by Leung, T. ;Burl, M and Perona, P., Proc. Int. Conf. on Computer Vision, 1995 © 1995, IEEE*

Notice that this line of reasoning extends. If we have a left eye response and a right eye response, and there is no value of the nose and mouth responses that would result in a posterior that is over threshold, we can prune this search — and so not attempt to add either a nose or a mouth to this assembly. It is often quite tricky to supply the necessary bounds, however. One way to do this is to use a classifier.

You can think of a classifier as representing a (very crude) probability model for the posterior. This model has a zero value when the posterior is below threshold, and a non-zero value when it is above. The great advantage of this model is that it is quite easy to prune — we can reject any assembly if there was no element that could be added to yield something that would lie in the non-zero region.

26.3.1 Identifying Acceptable Assemblies Using Projected Classifiers

There is no point in growing a working assembly unless there is some conceivable way in which it could become an acceptable final assembly. Assume that we have a classifier that can determine whether a final assembly is acceptable or not. This classifier can be trained on a large sequence of examples.

We can use this final classifier to predict tests that determine whether small groups should be expanded or not — essentially, we discard groups for which there is no prospect that any conceivable new set of elements would make the group acceptable. In turn, this boils down to projecting the decision boundaries of the classifier onto a set of different factors. For example, assume that we have a two eye detector responses — should they form a pair or not? We can look at only features that come from those eyes, because we don't know what other elements lie in the assembly, and therefore can't use them to compute features. In particular, the issue here is: is there some set of elements (i.e. a nose and a mouth) such that, if they were attached to this group, the resulting group would pass the classifier? This question is answered by a classifier whose decision boundary is obtained by projecting the decision boundary of the face assembly onto the space spanned by the features computed from our two eyes (as in figure 26.6).

By projecting the classifier onto components that can handle small groups, we can prune the collection of groups that must be searched to find a large collection of segments that passes the main classifier. Using this strategy in practice requires a certain amount of care. It is important to have classifiers which project well — the decision boundaries must project to decision boundaries that can be represented reasonably easily. This can be dealt with — the details are rather beyond the scope of this account — and the resulting classifiers can be used to find people and horses relatively efficiently in quite simple cases (figure 26.6 and figure 26.8) [Forsyth and Fleck, 1997b; Forsyth and Fleck, 1997c; Ioffe and Forsyth, 1998].

26.3.2 Example: Finding People and Horses Using Spatial Relations

Assume we wish to find people in images; a natural approach is to find possible body segments, and reason about their configuration. This can be done in some specialised cases. One that may become important is if the people concerned are not wearing clothing (which would allow people to search for or to avoid images that are associated with strong opinions). In this case, body segments can be found by looking for skin (as in section **??**), and then constructing extended image regions with roughly parallel sides (as in section 17.3) that contain skin colour. The resulting extended image regions tend to represent many of the body segments in the image.

Now we can search for people by searching for assemblies of these image segments. For example, assume that we will see only frontal views; then we can look for assemblies of nine image segments, containing a left upper arm, a left lower arm, etc. and a torso. It isn't practical to take all sets of nine segments from the image, so we use the methodology above. This yields a system that can find people in simple images fairly reliably [Ioffe and Forsyth, 1998]

This methodology extends to finding horses as well. We identify all image pixels that could be hide (this means they lie in the right range of colours, and have little texture locally). We then form extended regions of hide (as for people), and regard

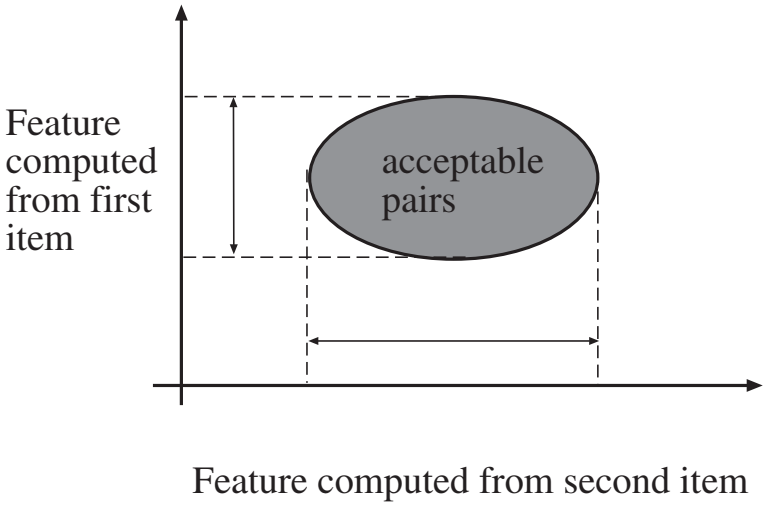

Figure 26.6. A classifier that accepts large groups can be used to predict tests that identify potentially useful small groups. Assume we want to group two items together: this grouping is acceptable only if a point, consisting of one feature computed from each item lies, in the shaded set. There are some items that could not appear in these groups, because the feature computed from the items lies out of the acceptable range, so that there does not exist a second item that could make the pair acceptable. It is possible to obtain classifiers that identify the acceptable range by projecting the main classifier onto its factors, as the picture indicates.

these as potential body segments. We use a crude model of a horse as a four segment group (a body, a neck and two legs); the order in which the tests are performed is illustrated in figure 26.7. The crude model is used because (a) it is robust to changes in aspect and (b) it doesn't require accurate localisation of all segments.

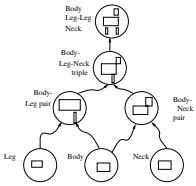

Figure 26.7. The model used for a horse involves two legs, a body and a neck. Segments are first tested to see whether they are acceptable members of one of these classes. We then test pairs of body and leg segments, and pairs of body and leg segments. We then take body-leg pairs and body-neck pairs *which share the same body* and test to see if they form a valid body-leg-neck triple. Pairs of triples that *share the same body and the same neck* are tested to form quadruples. *Figure from "Body Plans," by D.A. Forsyth and M.M. Fleck, Proc. Computer Vision and Pattern Recognition, 1997,* © *1997, IEEE*

This method of finding objects has been useful in some applications, as figure 26.6 and figure 26.8 suggest, but there are significant open questions. Firstly, it isn't obvious what sequence of intermediate groups and tests is best. It is possible to think about this problem by asking which projections of the decision boundary lead to small projected volumes (and so, hopefully, to relatively few passing groups). This looks like a difficult problem, because it deals with combinatorial properties. Secondly, it isn't obvious how we deal with multiple objects efficiently. If a single template occurs only on one object, then, when we observe the template, we immediately have an object hypothesis. Things are much more difficult when templates could have come from many objects. For example, consider recognizing people *and* horses: a single segment could come from either, as could many pairs — how do we arrange the sequence of tests to minimize the work required to decide what we are dealing with? Finally, the use of a classifier generates some annoying problems with false negatives.

26.4 Technique: Hidden Markov Models

Up to this point, adopting a probability model hasn't changed much of significance in our discussion of recognition. While we perform "verification" by evaluating a joint probability model, we are still engaged in a correspondence search, very like those of chapter 24. Our method of pruning is analogous to the interepretation tree of that chapter. However, more is possible: some probability models have a structure that make it possible to get an exactly optimal correspondence very efficiently indeed.

A program that reads American Sign Language from a video sequence of someone signing must infer a state, internal to the user, for each sign. The program will infer state from measurements of hand position that are unlikely to be accurate, but will depend — hopefully quite strongly — on the state. The signs change state in a random (but quite orderly) fashion. In particular, some sequences of states occur very seldom (e.g. a sequence of letter signs for the sequence "wkwk" is extremely unlikely). This means that both the measurements *and* the relative probabilities of different sequences of signs can be used to determine what actually happened.

The elements of this kind of problem are:

- there is a sequence of random variables (in our example, the signs), each of which is conditionally independent of all others given its predecessor;
- each random variable generates a measurement (the measurements of hand position) whose probability distribution depends on the state.

Similar elements are to be found in such examples as interpreting the movement of dancers, or of martial artists. There is an extremely useful formal model, known as a **hidden Markov model**, corresponding to these elements.

The sequence of random variables does not have to be temporal. Instead, we could order the variables by spatial relations, too. Consider an arm: the configura-

tion of the lower arm is (roughly!) independent of the configuration of the rest of the body given the configuration of the upper arm; the configuration of the upper arm is (roughly!) independent of the rest of the body given the configuration of the torso; etc. This gives us a sequence of random variables with the conditional independence properties described above. Now we don't really know the configuration of the lower arm — we have only some image measurements that have some probabilistic relationship with the configuration of the lower arm (as above).

26.4.1 Formal Matters

A sequence of random variables X_n is said to be a **Markov chain** if

$$P(\boldsymbol{X}_n = \boldsymbol{a}|\boldsymbol{X}_{n-1} = \boldsymbol{b}, \boldsymbol{X}_{n-2} = \boldsymbol{c}, \ldots, \boldsymbol{X}_0 = \boldsymbol{x}) = P(\boldsymbol{X}_n = \boldsymbol{a}|\boldsymbol{X}_{n-1} = \boldsymbol{b})$$

and a **homogenous Markov chain** if this probability does not depend on n. Markov chains can be thought of as sequences with very little memory; the new state depends on the previous state, but not on the whole history. It turns out that this property is surprisingly useful in modelling, because many physical variables appear to have it, and because it enables a variety of simple inference algorithms. There are very slightly different notations for Markov chains on discrete and continuous state spaces; we shall discuss only the discrete case.

Assume that we have a discrete state space. It doesn't really matter what dimension the space is, although finite spaces are somewhat easier to imagine. Write the elements of the space as s_i and assume that there are k elements. Assume that we have a sequence of random variables taking values in that state space that forms a homogenous Markov chain. Now we write

$$P(X_n = s_j|X_{n-1} = s_i) = p_{ij}$$

and because the chain is independent of n, so is p_{ij}. We can write a matrix $\mathcal{P}$ with i, j'th element p_{ij} which describes the behaviour of the chain; this matrix is called the **state transition matrix**. Assume that X_0 has probability distribution $P(X_0 = s_i) = \pi_i$, and we will write $\boldsymbol{\pi}$ as a vector with i'th element π_i. This means that

$$\begin{aligned} P(X_1 = s_j) &= \sum_{i=1}^{k} P(X_1 = s_j|X_0 = s_i)P(X_0 = s_i) \\ &= \sum_{i=1}^{k} P(X_1 = s_j|X_0 = s_i)\pi_i \\ &= \sum_{i=1}^{k} p_{ij}\pi_i \end{aligned}$$

and so the probability distribution for the state of X_1 is given by $\mathcal{P}^T\boldsymbol{\pi}$. By a similar argument, the probability distribution for the state of X_n is given by $(\mathcal{P}^T)^n\boldsymbol{\pi}$. For

all Markov chains, there is at least one distribution $\boldsymbol{\pi}^s$ such that $\boldsymbol{\pi}^s = \mathcal{P}^T\boldsymbol{\pi}^s$; this is known as the **stationary distribution** of the chain. Markov chains allow quite simple and informative pictures. We can draw a weighted, directed graph with a node for each state and the weight on each edge indicating the probability of a state transition (figure 26.9).

If we observe the random variable X_n, then inference is easy — we know what state the chain is in. This is a poor observation model, however. A much better model is to say that, for each element of the sequence, we observe *another* random variable, whose probability distribution depends on the state of the chain. That is, we observe some Y_n, where the probability distribution is some $P(Y_n|X_n = s_i) = q_i(Y_n)$. We can arrange these elements into a matrix $\mathcal{Q}$. Specifying a hidden Markov model requires providing the state transition process, the relationship between state and the probability distribution on Y_n and the initial distribution on states, i.e. the model is given by $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})$. We will assume that the state space has k elements.

26.4.2 Computing with Hidden Markov Models

We will assume that we are dealing with a hidden Markov model on a discrete state space — this simplifies computation considerably, usually at no particular cost. There are two important problems:

- **Inference:** We need to determine what underlying set of states gave rise to our observations. This will make it possible to, for example, infer what the dancer is doing or the signer is saying.
- **Fitting:** We need to choose a hidden Markov model that represents a sequence of past observations well.

Each has an efficient, standard solution.

The Trellis Model

Assume that we have a series of N measurements $\boldsymbol{Y}_i$ that we believe to be the output of a hidden Markov model. We can set up these measurements in a structure called a **trellis**. This is a weighted, directed graph consisting of N copies of the state space, which we arrange in columns. There is a column corresponding to each measurement. We weight the node representing state $\boldsymbol{X}_i$ in the column corresponding to $\boldsymbol{Y}_j$ with $\log q_i(\boldsymbol{Y}_j)$.

We join up the elements from column to column as follows. Consider the column corresponding the $\boldsymbol{Y}_j$; we join the element in this column representing state $\boldsymbol{X}_k$ to the element in the column corresponding to $\boldsymbol{Y}_{j+1}$ representing state $\vec{X}_l$ if p_{kl} is non-zero. This arc represents the fact that there is a possible transition between these states. This arc is weighted with $\log p_{kl}$. Figure 26.10 shows a trellis constructed from an HMM.

The trellis has the following interesting property: each (directed) path through the trellis represents a legal sequence of states. Now since each node of the trellis is

weighted with the log of the emission probability, and each arc is weighted with the log of the transition probability, the likelihood of sequence of states can be obtained by identifying the path corresponding to this sequence, and summing the weights (of arcs and nodes) along the path. This yields an extremely effective algorithm for finding the maximum likelihood path, known as **dynamic programming** or **the Viterbi algorithm**.

We start at the *final* column of the tellis. We know the log-likelihood of a one state path, ending at each node, as this is just the weight of that node. Now consider a two state path, which will start at the second last column of the trellis. We can easily obtain the best path leaving each node in this column. Consider a node: we know the weight of each arc leaving the node and the weight of the node at the far end of the arc, so we can choose the path segment with the largest value of the sum — this arc is the best we can do leaving that node. Now for each node, we add the weight at the node to the value of the best path segment leaving that node (i.e. the arc weight plus the weight of the node at the far end). This sum is the best value obtainable on reaching that node — which we'll call the **node value**.

Now, since we know the best value obtainable on reaching each node in the second-last column, we can figure out the best value obtainable on reaching each node in the third-last column. At each node in the third last column, we have a choice of arcs, each reaching a node whose value we know. We choose the arc with the largest value of (arc weight plus node value), add this value to the weight at the starting node in the third last column, and this yields the value of the starting node. We can repeat this process, until we have a value for each of the nodes in the first column; the largest value is the maximum likelihood.

We can also get the path with the maximum likelihood value. When we compute the value of a node, we erase all but the best arc leaving that node. Once we reach the first column, we simply follow the path from the node with the best value. Figure 26.11 illustrates this extremely simple and very powerful algorithm.

In the following sections, we describe dynamic programming rather more formally.

Inference and Dynamic Programming

For inference, we have a series of observations $\{Y_0, Y_1, \ldots, Y_n\}$ and we would like to obtain the sequence of $n+1$ states $\boldsymbol{S} = \{S_0, S_1, \ldots, S_n\}$ that maximises

$$P(\boldsymbol{S}|\{Y_0, Y_1, \ldots, Y_n\}, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$$

which is the same as maximising the joint distribution

$$P(\boldsymbol{S}, \{Y_0, Y_1, \ldots, Y_n\}|(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$$

There is a standard algorithm for this purpose, the **Viterbi algorithm**. We seek an $n+1$ element path through the states (from S_0 to S_n). There are k^{n+1} such paths, because we could choose from each state for every element of the path (assuming that there are no zeros in $\mathcal{P}$ — in most cases, there are certainly $O(k^{n+1})$

paths). We can't look at every path, but in fact we don't have to. The approach is as follows: assume that, for each possible state s_l, we know the value of the joint for the best n step path that ends in $S_{n-1} = s_l$; then the path that maximises the joint for an $n+1$ step path must consist of one of these paths, combined with another step. All we have to do is find the missing step.

We can approach finding the path with the maximum value of the joint as an induction problem. Assume that, for each value j of S_{n-1}, we know the value of the joint for the best path that ends in $S_{n-1} = j$, which we write as

$$\delta_{n-1}(j) = \max_{S_0, S_1, \ldots, S_{n-2}} P(\{S_0, S_1, \ldots, S_{n-1} = j\}, \{Y_0, Y_1, \ldots, Y_{n-1}\} | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$$

Now we have that

$$\delta_n(j) = \left(\max_i \delta_{n-1}(i) P_{ij} \right) q_j(Y_n)$$

We need not only the maximum *value*, but also the path that gave rise to this value. We define another variable

$$\psi_n(j) = \arg\max \left(\delta_{n-1}(i) P_{ij} \right)$$

(i.e. the best path that ends in $S_n = j$). This gives us an inductive algorithm for getting the best path.

The reasoning is as follows: I know the best path to each state for the $n-1$'th measurement; for each state for the n'th measurement, I can look backward and choose the best state for the $n-1$'th measurement; but I know the best path from there, so I have the best path to each state for the n'th measurements. We have put everything together in algorithm 1.

Fitting an HMM with EM

We have a dataset $\boldsymbol{Y}$ for which we believe a hidden Markov model is an appropriate model; but which hidden Markov model should we use? We wish to choose a model that best represents a set of data. To do this, we will use a version of the Expectation-Maximisation algorithm of chapter 18, due to Baum and Welch [**?**]. In this algorithm, we assume that we have an HMM, $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$; we now want to use this model and our dataset to estimate a new set of values for these parameters. We now estimate $(\overline{\mathcal{P}}, \overline{\mathcal{Q}}, \overline{\boldsymbol{\pi}}))$ using a procedure that is given below. There will be two possibilities (a fact which we won't prove). Either $P(\boldsymbol{Y}|(\overline{\mathcal{P}}, \overline{\mathcal{Q}}, \overline{\boldsymbol{\pi}})) > P(\boldsymbol{Y}|(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$, or $(\overline{\mathcal{P}}, \overline{\mathcal{Q}}, \overline{\boldsymbol{\pi}}) = (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})$.

The updated values of the model parameters will have the form:

$$\begin{aligned}
\overline{\pi_i} &= \text{expected frequency of being in state } s_i \text{ at time 1} \\
\overline{p_{ij}} &= \frac{\text{expected number of transitions from } s_i \text{ to } s_j}{\text{expected number of transitions from state } s_i} \\
\overline{q_j(k)} &= \frac{\text{expected number of times in } s_j \text{ and observing } Y = y_k}{\text{expected number of times in state } s_j}
\end{aligned}$$

We need to be able to evaluate these expressions. In particular, we need to be able to determine

$$P(X_t = s_i, X_{t+1} = s_j | \boldsymbol{Y}, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$$

which we shall write as $\xi_t(i,j)$. If we know $\xi_t(i,j)$, we have

$$\begin{aligned}
\text{expected number of transitions from } s_i \text{ to } s_j &= \sum_{t=1}^{T} \xi_t(i,j) \\
\text{expected number of times in } s_i &= \text{expected number of transitions from } s_i \\
&= \sum_{t=1}^{T} \sum_{j=1}^{N} \xi_t(i,j) \\
\text{expected frequency of being in } s_i \text{ at time 1} &= \sum_{j=1}^{N} \xi_1(i,j) \\
\text{expected number of times in } s_i \text{ and observing } Y = y_k &= \sum_{t=1}^{T} \sum_{j=1}^{N} \xi_t(i,j)\delta(Y_t, y_k)
\end{aligned}$$

where $\delta(u,v)$ is one if its arguments are equal and zero otherwise.

To evaluate $\xi_t(i,j)$, we need two intermediate variables: a **forward variable** and a **backward variable**. The forward variable is $\alpha_n(j) = P(Y_0, Y_1, \ldots, Y_n, X_n = s_j | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$. The backward variable is $\beta_t(j) = P(\{Y_{t+1}, Y_{t+2}, \ldots, Y_n\} | X_t = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$.

If we assume that we know the values of these variables, we have that

$$\begin{aligned}
\xi_t(i,j) &= P(X_t = s_i, X_{t+1} = s_j | \boldsymbol{Y}, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \\
&= \frac{P(\boldsymbol{Y}, X_t = s_i, X_{t+1} = s_j | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))}{P(\boldsymbol{Y} | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))} \\
&= \frac{\left\{ \begin{array}{c} P(Y_0, Y_1, \ldots, Y_t, X_t = s_i | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \times \\ P(Y_{t+1} | X_{t+1} = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \times \\ P(X_{t+1} = s_j | X_t = s_i, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \times \\ P(Y_{t+2}, \ldots, Y_N | X_{t+1} = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \end{array} \right\}}{P(\boldsymbol{Y} | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))} \\
&= \frac{\alpha_t(i) p_{ij} q_j(Y_{t+1}) \beta_{t+1}(j)}{P(\boldsymbol{Y} | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))} \\
&= \frac{\alpha_t(i) p_{ij} q_j(Y_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) p_{ij} q_j(Y_{t+1}) \beta_{t+1}(j)}
\end{aligned}$$

Both the forward and backward variables can be evaluated by induction. We get $\alpha_n(j)$, by observing that:

$$\alpha_0(j) = P(Y_0, X_0 = s_j | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$$

$$= \pi_j q_j(Y_0)$$

$$\begin{aligned}
\alpha_{t+1}(j) &= P(Y_0, Y_1, \ldots, Y_{t+1}, X_{t+1} = s_j | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \\
&= P(Y_0, Y_1, \ldots, Y_t, X_{t+1} = s_j | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) P(Y_{t+1} | X_{t+1} = s_j) \\
&= \sum_{l=1}^{k} [P(Y_0, Y_1, \ldots, Y_t, X_t = s_l, X_{t+1} = s_j | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) P(Y_{t+1} | X_{t+1} = s_j)] \\
&= \left[\sum_{l=1}^{k} P(Y_0, Y_1, \ldots, Y_t, X_t = s_l | (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) P(X_{t+1} = s_j | X_t = s_l) \right] P(Y_{t+1} | X_{t+1} = s_j) \\
&= \left[\sum_{l=1}^{k} \alpha_t(l) p_{lj} \right] q_j(Y_{t+1}) \qquad 1 \leq t \leq n-1
\end{aligned}$$

This backward variable can also be obtained by induction as:

$$\begin{aligned}
\beta_N(j) &= P(\text{no further output} | X_n = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \\
&= 1
\end{aligned}$$

$$\begin{aligned}
\beta_t(j) &= P(\{Y_{t+1}, Y_{t+2}, \ldots, Y_n\} | X_t = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \\
&= \sum_{l=1}^{k} [P(\{Y_{t+1}, Y_{t+2}, \ldots, Y_n\}, X_t = s_l | X_{t+1} = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))] \\
&= \left[\sum_{l=1}^{k} P(X_t = s_l, Y_{t+1} | X_{t+1} = s_j) \right] P(\{Y_{t+2}, \ldots, Y_n\} | X_{t+1} = s_j, (\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})) \\
&= \left[\sum_{l=1}^{k} p_{jl} q_l(Y_{t+1}) \right] \beta_{t+1}(j) \qquad 1 \leq t \leq k-1
\end{aligned}$$

As a result, we have a simple fitting algorithm, collected in algorithm 2

26.4.3 Varieties of HMM's

We have not spoken about the topology of the graph underlying our model. This could be a **complete graph** (one where every node is connected to every other in both directions), but does not have to be. One disadvantage of using complete graphs is that there are a large number of parameters to estimate. In applications, a number of variants have proven useful.

A **left-right model** or **Bakis model** has the property that, from state i, the model can move only to states $j > i$. This means that for $j < i$, $p_{ij} = 0$. Furthermore, $\pi_1 = 1$ and for all $i \neq 1$, $p_i = 0$. This means that the state transition matrix $\mathcal{P}$ will be upper triangular. Furthermore, an N state left-right model will

stay in the N'th state once it has been reached. You should notice that this model captures a notion of the order in which events can occur, which could be convenient if we were using HMM's to encode various kinds of motion information (or speech information, where the model originates).

However, a general left-right model suggests that large numbers of events may be skipped (the state may advance freely). This isn't really consistent with a reasonable model of motion, because while we may miss a measurement or two or a state or two, it is unlikely that we will miss a large collection of states. It is common to use the additional constraint that for $j > i + \delta$, $p_{ij} = 0$. Here δ tends to be a small number (two is often used).

Surprisingly, constraining the topology of the model does not create any problems with the algorithms of the previous sections. You should verify that the estimation algorithm preserves zeros, meaning that if we start it with model of a particular topology — which will have zeros in particular spots of the state transition matrix — it will produce a new estimate of the state transition matrix that also has zeros in those spots.

26.5 Application: Hidden Markov Models and Sign Language Understanding

Sign language is language which is rendered using a system of gestures, rather than by manipulating the vocal tract. For most people, learning to understand sign language takes an effort; it would be attractive to have a device that could be pointed at someone who was signing, and would generate speech. This is a problem that has some strong analogies with speech recognition, an area that is now quite highly developed.

Hidden Markov models have been highly successful in speech understanding applications. The hidden states describe the vocal system; the observations are various acoustic measurements. Typically, there is a hidden Markov model associated with each word. These models are attached together using a **language model**, which specifies the probability of a word occurring given some other word has already occurred. The resulting object is another — possibly very big — HMM. The sentence represented by a set of acoustic measurements is then obtained by an inference algorithm applied to the language model. This technique has worked well for the speech community, and it is natural to try to suborn the model to deal with sign language.

Human gestures are rather like human sounds — there is typically a sequence of events, and we have measurements that result from these events, but do not determine them. While there is no guarantee that the rather stiff conditional independence assumptions underlying hidden Markov models are actually true for human gestures, the model is certainly worth trying, because there is no such guarantee for speech models, and HMM's work in that application.

We will describe systems that use HMM's to interpret sign language, but you

should realize that this approach will also work for formal systems of gestures. For example, if I wish to build a system that opens a window when I move my hand one way, and closes it if I move my hand another way, I could regard that set of gestures as a rather restricted sign language.

There are a number of systems that use HMM's to interpret sign language. Generally, word models are left-right models, but constrained not to skip too many states (i.e. $p_{ij} = 0$ for $j > i + \Delta$ and Δ fairly small). Typically, there are a small number of states; each state has a self-loop, meaning that the model can stay in that state for a number of ticks; and there are transitions that skip some states, meaning that it is possible to move through the hidden states rather fast. Only two issues must now be resolved to build a system: firstly, the word models need to be connected together with some form of language model, and secondly, one must decide what to measure.

26.5.1 Language Models: Sentences from Words

We wish to recognise expressions in sign language, rather than isolated words, so we need to specify how words will appear. Language models are a specification of how word HMM's should be strung together to give an HMM that represents full sentences. The simplest language model has words that occur independent of the previous word. The language model can be expressed by a graph, as in figure 26.13. In this model, the state moves to a start state, then emits a word, then either stops or loops back from where another word can be emitted. It is now necessary to find an extremal path through this larger graph; the Viterbi algorithm still applies, there is just no observation corresponding to the shaded states.

More sophisticated language models are desirable — a language model for English that had words drawn independently at random would generate quite long strings of "and"'s and "the"'s — but do generate computational problems. The natural first step is to have a bigram language model, where the probability that a word is produced depends on the word that was produced previously. This generates a language model of the form of figure 26.14. More sophisticated language models involve trigrams — that is, the probability that a word is emitted depends on the last two words — or more detailed context from the preceeding sentence. The difficulty with this approach is that, for a reasonably sized vocabulary, the number of states the Viterbi algorithm has to search can be prodigious. There are a variety of techniques for pruning this search, which are beyond our scope — Jelinek's book gives one approach in chapter 5 [**?**], or see [**?**].

Features and Typical Levels of Performance

There are a few current programs for sign language recognition. Starner has written several programs, using a variety of features. The simplest approach requires a user to wear a yellow glove on the right hand and a red glove on the left hand [Starner and Pentland, 1996]. An alternative approach to wearing gloves — which is something of a nuisance — is to segment hands using skin colour (always assuming there is

nothing else skin-coloured in the vicinity) [Starner *et al.*, 1998]. Pixels of interesting colours (yellow and red, or skin) are identified in the image. Once an appropriate pixel has been found, its eight neighbours are checked to determine if their colour is acceptable, too; this process continues to obtain a blob of pixels.

Blobs admit a variety of features. The center of gravity of the blob gives two features, and the change in center of gravity from the previous frame yields another two. The area of the blob is another feature. The orientation and size of the blob can be measured by forming a matrix of second moments,

$$\begin{pmatrix} \int x^2 dxdy & \frac{1}{2}\int xydxdy \\ \frac{1}{2}\int xydxdy & \int y^2 dxdy \end{pmatrix}$$

The ratio of eigenvalues of this matrix gives an indication of the eccentricity of the blob; the largest eigenvalue is an estimate of the size along the principal direction; and the orientation of the eigenvector corresponding to this eigenvalue gives the orientation of the blob.

Starner's system works on a vocabulary of 40 words, representing a total of four parts of speech. The topology of the HMM's for the words is given and the parameters are estimated using the EM algorithm of chapter 18. For both isolated word recognition tasks and for recognition using a language model that has five word sentences (words always appearing in the order `pronoun verb noun adjective pronoun`), the system displays a word accuracy of the order of 90%. Values are slightly larger or smaller, depending on the features and the task, etc.

Vogler and Metaxas have built a system that uses estimates of arm position, recovered either from a physical sensor mounted on the body or from a system of three cameras that measures arm position fairly accurately [Vogler and Metaxas, 1998; Vogler and Metaxas, 1999]. For a vocabulary of 53 words, and an independent word language model, they report a word recognition accuracy of the order of 90%.

The Future

All the results in the literature are for very small vocabularies and very simple language models. Nonetheless, HMM's seem a promising method for recognising sign language. It isn't clear that very simple features are sufficient to recognise complex signs (though it is known that very coarse scale movies of sign language are still quite intelligible [**?**]). One possible direction for progress is to use features that give a better estimate of what the fingers are doing.

Good language models — and appropriate inference algorithms — are at the core of the success of modern speech recognition systems. These models are typically obtained by measuring the frequency with which a word occurs in some context — for example, trigram frequencies. Building these models requires tremendous quantities of data, because we need an accurate estimate of the relative frequencies of quite infrequent events. For example, as Jelinek points out ([**?**], p. 75) it typically takes a body of 640, 000 words to assemble a set of 15, 000 different words. This means that some words are used very often, and that measuring word frequencies

on a small body of data is extremely dangerous. Both the speech recognition community and the natural language community have a lot of experience with these difficulties, and a variety of sophisticated tricks for dealing with them (e.g. [**?**; **?**; **?**]). Future research in sign language recognition will involve learning these tricks and transferring them to the vision domain.

26.6 Application: Finding People with Hidden Markov Models

It is fairly easy to see how a hidden Markov model is a reasonable choice for recognising sign language — signs occur in a constrained random order, and generate noisy measurements. There are other applications that are not as obvious. The important property of a hidden Markov model is not the temporal sequence; it is conditional independence — that X_{i+1} is independent of the past, given X_i.

This sort of independence occurs in a variety of cases. By far the most important is finding people. We assume that people appear in images as dolls, consisting of nine body segments (upper and lower left and right arms and legs respectively, and a torso) each of which is rectangular. In particular, we assume that the left lower arm is independent of all other segments given the left upper arm; that the left upper arm is independent of all segments given the torso; and extend these assumptions in the obvious way to include the right arm and the legs, too. This gives us a hidden Markov model. We can write the model out to emphasize this point. We write X_{lua} for the configuration of the left upper arm, etc., and have

$$\begin{aligned} P(X_t, X_{lua}, X_{lla}, X_{rua}, X_{rla}, X_{lul}, X_{lll}, X_{rul}, X_{rll}) = \; & P(X_t)P(X_{lua}|X_t) \times \\ & P(X_{lla}|X_{lua}) \times \\ & P(X_{rua}|X_t) \times \\ & P(X_{rla}|X_{rua}) \times \\ & P(X_{lul}|X_t) \times \\ & P(X_{lll}|X_{lul}) \times \\ & P(X_{rul}|X_t) \times \\ & P(X_{rll}|X_{rul}) \end{aligned}$$

which we can draw as a tree indicating the dependencies (figure 26.16).

Now assume that we observe some image measurements relating to segment configuration. For the moment, we assume that each body segment can occupy one of a finite collection of discrete states — we write the event that a particular limb is in a particular state as, say, $X_{lua} = \boldsymbol{x}_{lua}$. This event will result in a measurement, whose conditional probability can be written as $P(M = \boldsymbol{m}|X_{lua} = \boldsymbol{x}_{lua})$ etc. In particular, we will have a system of segments in the image, some of which correspond to limb segments and some of which result from noise. We assume that there are N_s segments in the image. Furthermore, we assume that the probability that a segment arises from noise is independent of anything we can measure from the segment. Now for each correspondence of image segments to limb segments we can

evaluate a likelihood function. We need to be able to write out the correspondence; let us write $\{i_1, \ldots, i_9\}$ as the event that image segment i_1 is the torso, i_2 is the left upper arm, through to i_9 is the right lower leg, and that all others are noise. We write $\boldsymbol{m}_{i_k}$ for the image measurements associated with the i_k'th image segment.

Assume, for the moment, that we expect all body segments to be present at every stage. We now wish to determine which choice of image segments represents the body segments of a person who is present, and what the configuration of that person is. We have a log-likelihood that looks like:

$$\begin{aligned}\log P(\{i_1, \ldots, i_9\} | X_t = \boldsymbol{x}_t, \ldots, X_{rll} = \boldsymbol{x}_{rll}) = \; & \log P(\boldsymbol{m}_{i_1} | \boldsymbol{x}_t) + \\ & \log P(\boldsymbol{m}_{i_2} | \boldsymbol{x}_{lua}) + \ldots \\ & \log P(\boldsymbol{m}_{i_9} | \boldsymbol{x}_{rll}) + \\ & (N_s - 9) P(\text{image segment from noise})\end{aligned}$$

Now if we write $P(X_{lua} = \boldsymbol{x}_{lua} | X_t = \boldsymbol{x}_t)$ as $P(\boldsymbol{x}_{lua} | \boldsymbol{x}_t)$, the the log of the joint probability is

$$\begin{aligned}\log P(\{i_1, \ldots, i_9\}, X_t = \boldsymbol{x}_t, \ldots, X_{rll} = \boldsymbol{x}_{rll}) &= \log P(\{i_1, \ldots, i_9\} | X_t = \boldsymbol{x}_t, \ldots, X_{rll} = \boldsymbol{x}_{rll}) + \\ &\quad \log P(\boldsymbol{x}_t, \boldsymbol{x}_{lua}, \boldsymbol{x}_{lla}, \boldsymbol{x}_{rua}, \boldsymbol{x}_{rla}, \boldsymbol{x}_{lul}, \boldsymbol{x}_{lll}, \boldsymbol{x}_{rul}, \boldsymbol{x}_{rll}) \\ &= \log P(\{i_1, \ldots, i_9\} | X_t = \boldsymbol{x}_t, \ldots, X_{rll} = \boldsymbol{x}_{rll}) + \\ &\quad \log P(\boldsymbol{x}_t) P(\boldsymbol{x}_{lua} | \boldsymbol{x}_t) + \\ &\quad \log P(\boldsymbol{x}_{lla} | \boldsymbol{x}_{lua}) + \\ &\quad \log P(\boldsymbol{x}_{rua} | \boldsymbol{x}_t) + \\ &\quad \log P(\boldsymbol{x}_{rla} | \boldsymbol{x}_{rua}) + \\ &\quad \log P(\boldsymbol{x}_{lul} | \boldsymbol{x}_t) + \\ &\quad \log P(\boldsymbol{x}_{lll} | \boldsymbol{x}_{lul}) + \\ &\quad \log P(\boldsymbol{x}_{rul} | \boldsymbol{x}_t) + \\ &\quad \log P(\boldsymbol{x}_{rll} | \boldsymbol{x}_{rul}) \\ &= \log P(\boldsymbol{m}_{i_1} | \boldsymbol{x}_t) + \\ &\quad \log P(\boldsymbol{m}_{i_2} | \boldsymbol{x}_{lua}) + \ldots \\ &\quad \log P(\boldsymbol{m}_{i_9} | \boldsymbol{x}_{rll}) + \\ &\quad (N_s - 9) P(\text{image segment from noise}) + \\ &\quad \log P(\boldsymbol{x}_t) P(\boldsymbol{x}_{lua} | \boldsymbol{x}_t) + \\ &\quad \log P(\boldsymbol{x}_{lla} | \boldsymbol{x}_{lua}) + \\ &\quad \log P(\boldsymbol{x}_{rua} | \boldsymbol{x}_t) + \\ &\quad \log P(\boldsymbol{x}_{rla} | \boldsymbol{x}_{rua}) + \\ &\quad \log P(\boldsymbol{x}_{lul} | \boldsymbol{x}_t) + \\ &\quad \log P(\boldsymbol{x}_{lll} | \boldsymbol{x}_{lul}) + \\ &\quad \log P(\boldsymbol{x}_{rul} | \boldsymbol{x}_t) +\end{aligned}$$

$$\log P(\boldsymbol{x}_{rll}|\boldsymbol{x}_{rul})$$

Now all this fits into the lattice structure — which is the core of dynamic programming — rather easily. For each body segment, we establish a column of nodes, one for each pair of the form (image segment, body segment state). Attached to each of these nodes is a term of the form $\log P(\boldsymbol{m}_{i_1}|\boldsymbol{x}_t)$, whose value we know because we know both the image segment and the body segment state represented by the node. There is a directed arc from each element of the column representing a body segment in the tree to each element in the columns representing its children. These arcs are labelled with the logs of the appropriate "transition" probabilities, for example $P(\boldsymbol{x}_{lua}|\boldsymbol{x}_t)$ (see figure 26.17). We wish to find the directed path with the largest sum of node and arc values along the path.

The model's structure means that some nodes along the path will have more than one child; this doesn't matter. The element of dynamic programming is that it is possible to process the lattice so that, at each node, we know the value of the best choice available at that node. This element is present here, too. We transfer values back up directed arcs as before, and when a node has more than one children, we sum the best choices available over the children. Figure **??** illustrates the process for a simplified lattice.

Models of this form can be used to find people in images in a fairly straightforward manner. Firstly, we need some model linking image observations to the configuration of the limb segments — i.e. a likelihood model. Felzenszwalb and Huttenlocher assume that segments have known colour patterns — typically, a mixture of skin colour and blue — and then compare the actual image colour with these patterns [Felzenszwalb and Huttenlocher, 2000]. This leads to a fairly satisfactory matcher (figure 26.18), with the proviso that the person's clothing be known in advance.

26.7 Conclusions

This topic is one on which substantial research is being conducted as we write; by the time anyone reads it, the chapter certainly won't represent the cutting edge of research. We have tried to identify the intellectual strands we think will prove most significant, which explains the emphasis on inference as search. The primary — and stunning — attraction of an HMM is that inference is very simple (or, equivalently, the correspondence search is very easily structured).

26.7.1 Hidden Markov Models

Dynamic programming has been used to find people in various ways. In [Ioffe and Forsyth, 1999], it is used to recommend hypotheses to a process that accepts or rejects them based on whether segments are shared or not — the conditional independence assumption prevents one compelling the model to ensure that body segments do not overlap. Felzenszwalb and Huttenlocher found people explicitly with dynamic programming and regional features [Felzenszwalb and

Huttenlocher, 2000]. Song *et al.* formulate finding people using motion information as a correspondence search through a probabilistic model and use dynamic programming to solve the search [Song *et al.*, 1999; Song *et al.*, 2000b; Song *et al.*, 2000a].

HMM's have some very serious difficulties as devices for visual inference. The first is that their discriminative performance on vision problems hasn't been all that good, to date. The second is that the property that inference is easy comes from the very strong structural constraints on the model — these conditional independence constraints can make it difficult to impose quite natural constraints. For example, the model used for finding people does not admit the constraint that different parts of the image should correspond to different body segments (because that would involve adding an edge to the tree of figure 26.18, which would result in a model that was not a tree). The difficulty with models that have more complex conditional independence relations is that inference can be difficult. An important research topic in object recognition involves finding models that (a) are quite a good representation of the world, in the sense that they can be used to achieve efficient discrimination; (b) admit quite simple inference algorithms; and (c) can be composed easily to yield new models.

This question of composition is important, and hasn't had much attention yet. Basically, if one wishes to recognise large numbers of different types of object, it is probably difficult to do so by thinking of this collection as flat — in the sense that one (simultaneously or in sequence) attempts to match to each model independently. However, if each object gave rise to a different set of features, we would be forced to do this (by finding the features corresponding to the first object, the second, etc.). A more attractive model involves having features that, in general form, apply to many different types of object, with the distinctions between these types emerging from (a) the relations between the features and (b) the detailed appearance of the features. How this process works remains extremely vague.

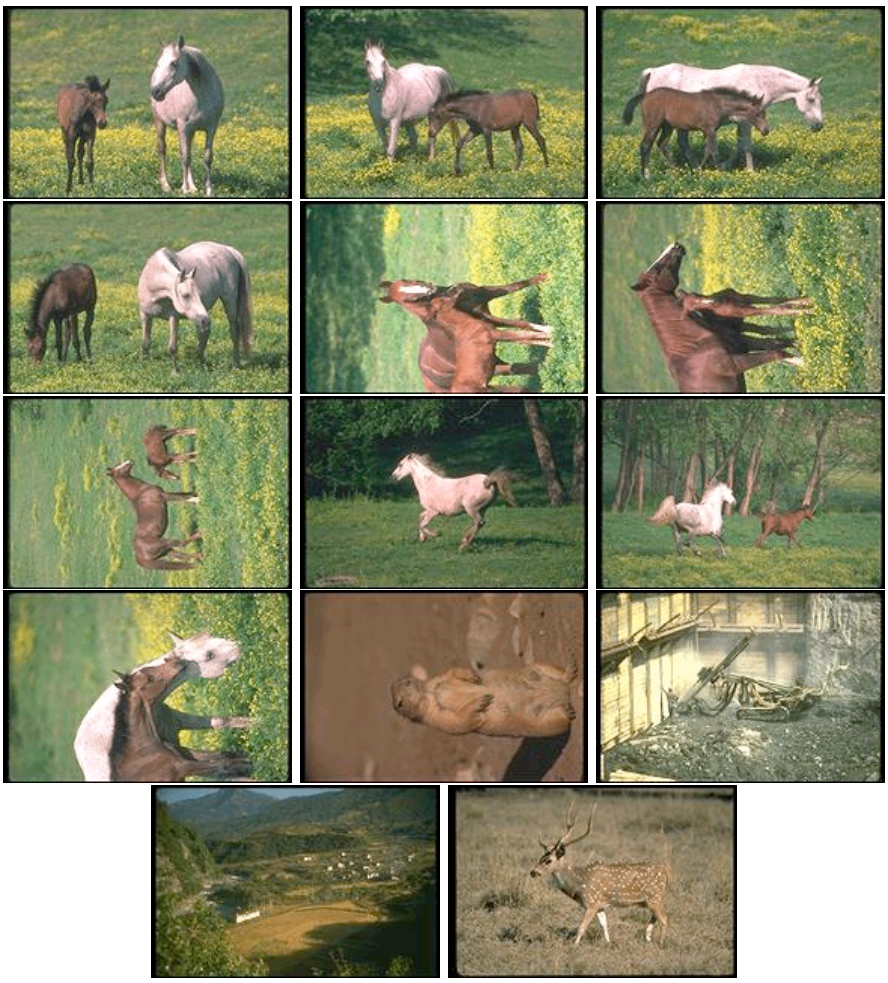

Figure 26.8. Horses can be found by building a classifier of the type described in the text. This example illustrates the effectiveness of the approach. The system finds image regions that have the colour and texture of hide, and look like cylinders (as in section 17.3). A horse is defined as a collection of four cylinders — a body, a neck and two legs — and a classifier that accepts this assembly is learned from examples. This classifier is projected onto factors, as described in the text, and these projected versions are used to build assemblies in an order that was arbitrarily selected. The result is a system that, while not spectacularly accurate, is useful. The figure shows all images that the classifier thinks contains horses, obtained using a control set of 1086 non-horse images and a test set of 100 horse images. The test images recovered contain horses in a wide range of aspects; one control image contains an animal that might reasonably pass for a horse. This figure appears also as figure (it's reproduced here for convenience). *Figure from "Body Plans," by D.A. Forsyth and M.M. Fleck, Proc. Computer Vision and Pattern Recognition, 1997,*

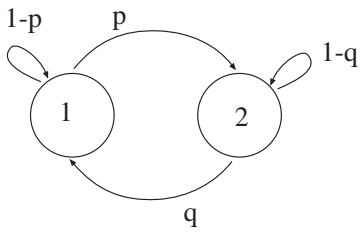

Figure 26.9. A simple, two-state Markov chain. In this chain, the probability of going from state one to state two is p; from state one to state one is $1-p$; etc. We could describe this chain with the state transition matrix Its stationary distribution is $(q/(p+q), p/(p+q))$. This makes sense; for example, if p is very small and q is close to one, the chain will spend nearlly all its time in state one. Notice that, if p and q are both very small, the chain will stay in one state for a long time, and then flip to the other state, where it will stay for a long time.

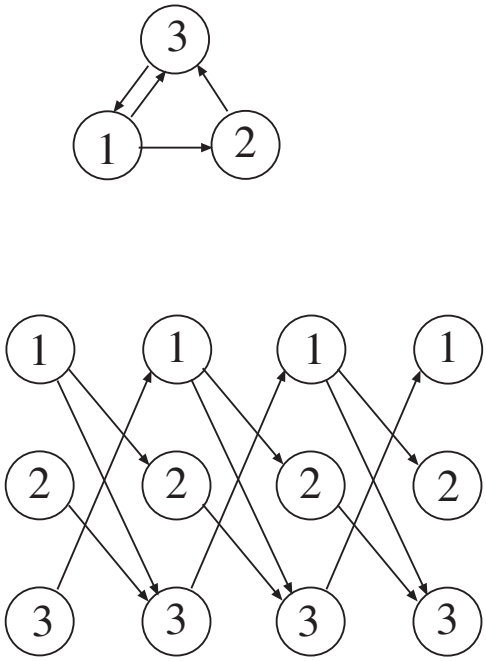

Figure 26.10. *At the top, a simple state transition model. We have labelled each arc with the log of the probability that that transition occurs from the relevant start node. Below, the trellis corresponding to that model. Notice that each path through the trellis corresponds to a legal sequence of states, for a sequence of four measurements. We weight the arcs with the log of the transition probabilities, and the nodes with the log of the emission probabilities; arc weights are not shown here, to reduce the complexity of the drawing.*

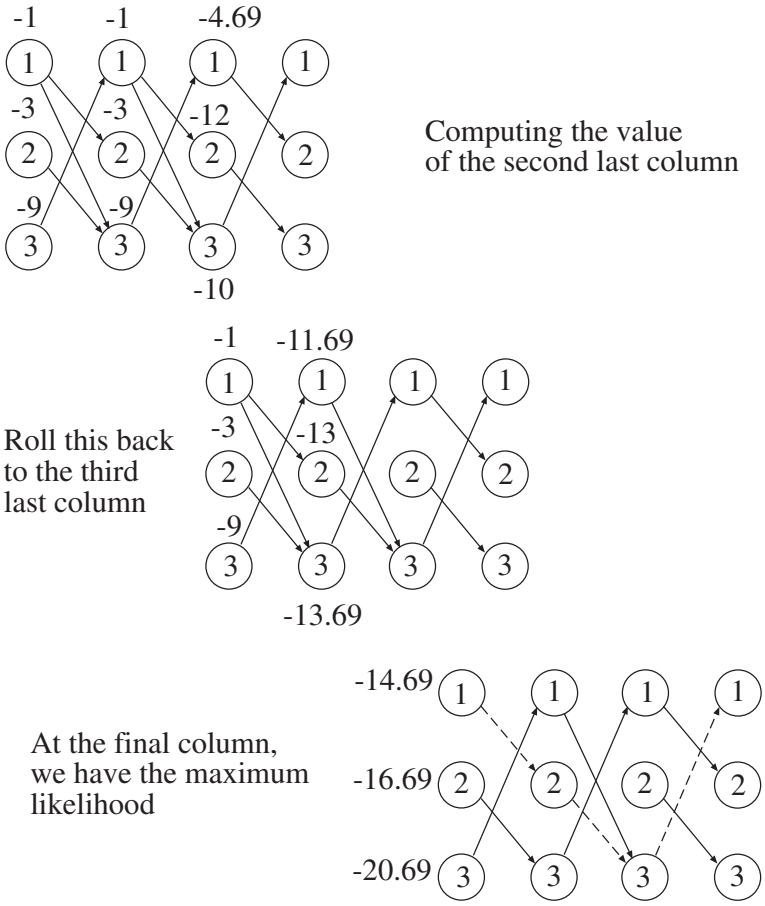

Figure 26.11. *It is a simple matter to find the best path through the trellis of figure 26.10 (or any other trellis, for that matter!). We assume that each 1 node has log-probability -1, each 2 node has log-probability -3 and each 3 node has log probability -9. We also assume that the probabilities of leaving a node are uniform (check our numbers!). Now the value of each node in the second-last column is the value of the node plus the best value to be obtained by leaving that node. This is easily computed. The algorithm involves computing the value of each node in the second-last column; then of each node in the third last column; etc. as described in the text. Once we get to the start of the trellis, the largest weight is the maximum of the log-likelihood; since we erased all but the best path segments, we have the best path, too (indicated by a dashed line).*

1. **Initialization:**

$$\begin{aligned}\delta_1(j) &= \pi_j b_j(Y_1) 1.0\text{cm} 1 \leq j \leq N \\ \psi_1(j) &= 0\end{aligned}$$

2. **Recursion:**

$$\begin{aligned}\delta_n(j) &= \left(\max_i \delta_{n-1}(i) P_{ij}\right) q_j(Y_n) \\ \psi_n(j) &= \arg\max \left(\delta_{n-1}(i) P_{ij}\right)\end{aligned}$$

3. **Termination:**

$$\begin{aligned}p^* &= \max_i \left(\delta_N(i)\right) \\ q_N^* &= \arg\max_i \left(\delta_N(i)\right)\end{aligned}$$

4. **Path backtracking:**

$$q_t^* = \psi_{t+1}(q_{t+1}^*)$$

Algorithm 26.1: The Viterbi algorithm *yields the path through an HMM that maximises the joint, and the value of the joint at this path. Here δ and ψ are convenient bookkeeping variables (as in the text);p^* is the maximum value of the joint; and q_t^* is the t'th state in the optimal path.*

Until $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})_{i+1}$ is the same as $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})_i$
  compute the forward variables α and β
  using the procedures of algorithms 4 and 5

  compute $\xi_t(i,j) = \frac{\alpha_t(i) p_{ij} q_j(Y_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) p_{ij} q_j(Y_{t+1}) \beta_{t+1}(j)}$

  compute the updated parameters using the procedures of algorithm 3

  These values are the elements of $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})_{i+1}$
end

Algorithm 26.2: Fitting Hidden Markov Models *to a data sequence* $\boldsymbol{Y}$ *is achieved by a version of EM. We assume a model* $(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi})_i$*, and then compute the coefficients of a new model; this iteration is guaranteed to converge to a local maximum of* $P(\boldsymbol{Y}|(\mathcal{P}, \mathcal{Q}, \boldsymbol{\pi}))$*.*

$$
\begin{aligned}
\overline{\pi_i} &= \text{expected frequency of being in state } s_i \text{ at time 1} \\
&= \sum_{j=1}^{N} \xi_1(i,j) \\
\overline{p_{ij}} &= \frac{\text{expected number of transitions from } s_i \text{ to } s_j}{\text{expected number of transitions from state } s_i} \\
&= \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \sum_{j=1}^{N} \xi_t(i,j)} \\
\overline{q_i(k)} &= \frac{\text{expected number of times in } s_i \text{ and observing } Y = y_k}{\text{expected number of times in state } s_i} \\
&= \frac{\sum_{t=1}^{T} \sum_{j=1}^{N} \xi_t(i,j) \delta(Y_t, y_k)}{\sum_{t=1}^{T} \sum_{j=1}^{N} \xi_t(i,j)}
\end{aligned}
$$

here $\delta(u,v)$ is one if its arguments are equal and zero otherwise.

Algorithm 26.3: Computing the new values of parameters *for the hidden Markov model fitting process*

$$\alpha_0(j) = \pi_j q_j(Y_0)$$

$$\alpha_{t+1}(j) = \left[\sum_{l=1}^{k} \alpha_t(l) p_{lj}\right] q_j(Y_{t+1}) \qquad 1 \leq t \leq n-1$$

Algorithm 26.4: Computing the forward variable *for the hidden Markov model fitting process.*

$$\beta_N(j) = 1$$

$$\beta_t(j) = \left[\sum_{l=1}^{k} p_{jl} q_l(Y_{t+1})\right] \beta_{t+1}(j) \qquad 1 \leq t \leq k-1$$

Algorithm 26.5: Computing the backward variable *for the hidden Markov model fitting process*

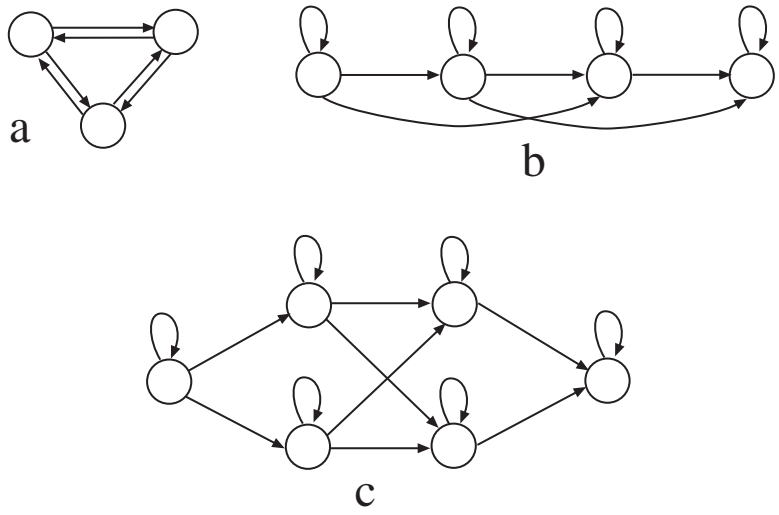

Figure 26.12. A variety of HMM topologies have proven useful in practice. At the **top**, we show an ergodic or fully connected model. In the **middle**, we show a four state left-right model. This model has been constrained so that $p_{ij} = 0$ for $j > i + 2$ as well. At the **bottom**, we show a six state parallel path left-right model. This model has, in essence, two parallel left-right models with the option of switching between them. *figure from A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, L. Rabiner, p.266, in the fervent hope of receiving permission*

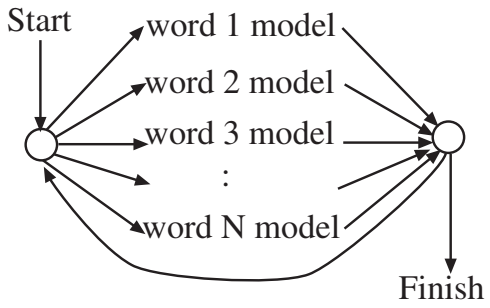

Figure 26.13. If we take an HMM for each word in the vocabulary, and string these models together with independent emission probabilities, we obtain a language model. This is of the very simplest kind, with independent words, and no constraint on the length of a sentence; while it has very little syntax, it is still an HMM and allows extremely simple inference.

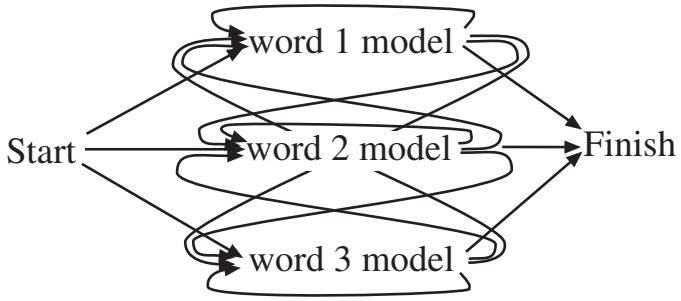

Figure 26.14. In a bigram language model, the probability that a word will be emitted depends on the word that was emitted previously. Such models can be laid out as HMM's, again by stringing the word models together. Notice that the topology is now considerably more complicated; for simplicity, we have allowed only three words.

Figure 26.15. Starner and colleagues have built a sign-language recognition system that uses HMM's to yield word models. This figure shows the view of a signer presented to the system; it is obtained from a camera on the desktop. *Figure from "Real time American sign language recognition using desk and wearable computer based video," T. Starner, et al. Proc. Int. Symp. on Computer Vision, 1995,* © *1995, IEEE*

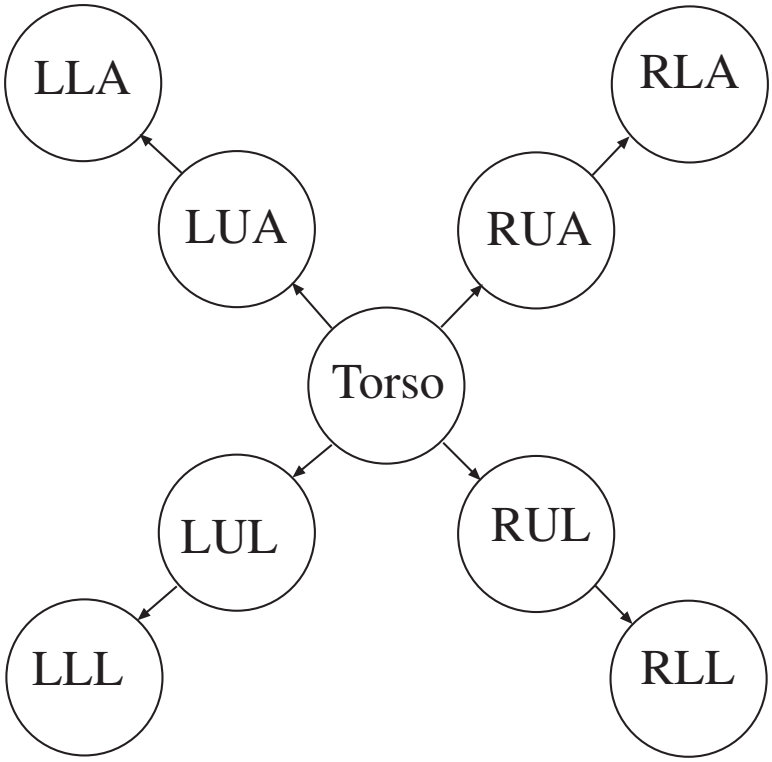

Figure 26.16. One can think of a human as forming a hidden Markov model. The tree in the figure illustrates the structure of one possible model: the torso generates various structures — arms and legs — whose properties are conditionally independent given the torso. The lower leg is conditionally independent of the rest of the model given the upper leg, etc. We can encode these independence properties by drawing a node for each variable, and a directed arc from a variable to another if the second depends directly on the first. If this drawing (which is a directed graph) is a tree, then we have a hidden Markov model. Notice that the semantics of this drawing are somewhat different from those of the drawing of figure 26.9; that drawing showed the possible state transitions and their probabilities, whereas this shows the variable dependencies.

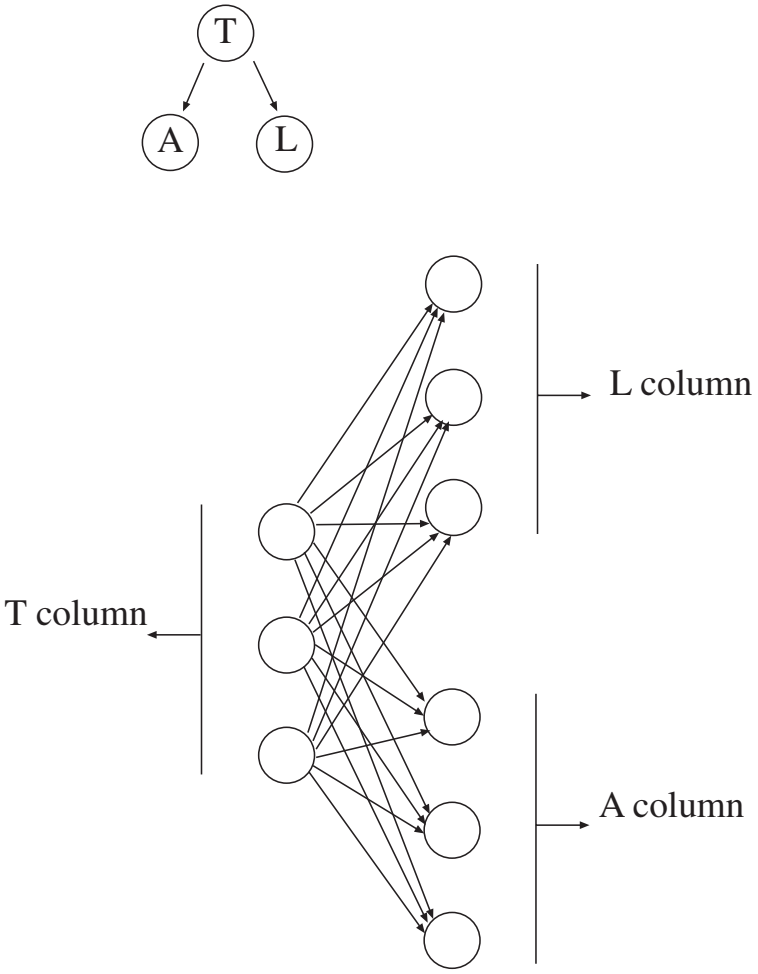

Figure 26.17. The figure shows a trellis derived from a simple tree, based around a simplified human model. You can read leg for L, etc. The elements of a column correspond to different possible correspondences between image segments, body segments and body segment configuration variables. For example, a node might represent the fact that image segment two corresponds to a torso segment at a particular position. The fact that nodes in the column marked "T" have two children each — instead of the single child in our previous example — creates no problem. For each node in this column, we can determine the best available option, and its value. This is obtained by computing the best available "A" option (and value) and the best available "L" option (and its value). In turn, this gives us the value of the "T" node. This observation means that we can use dynamic programming for any tree model.

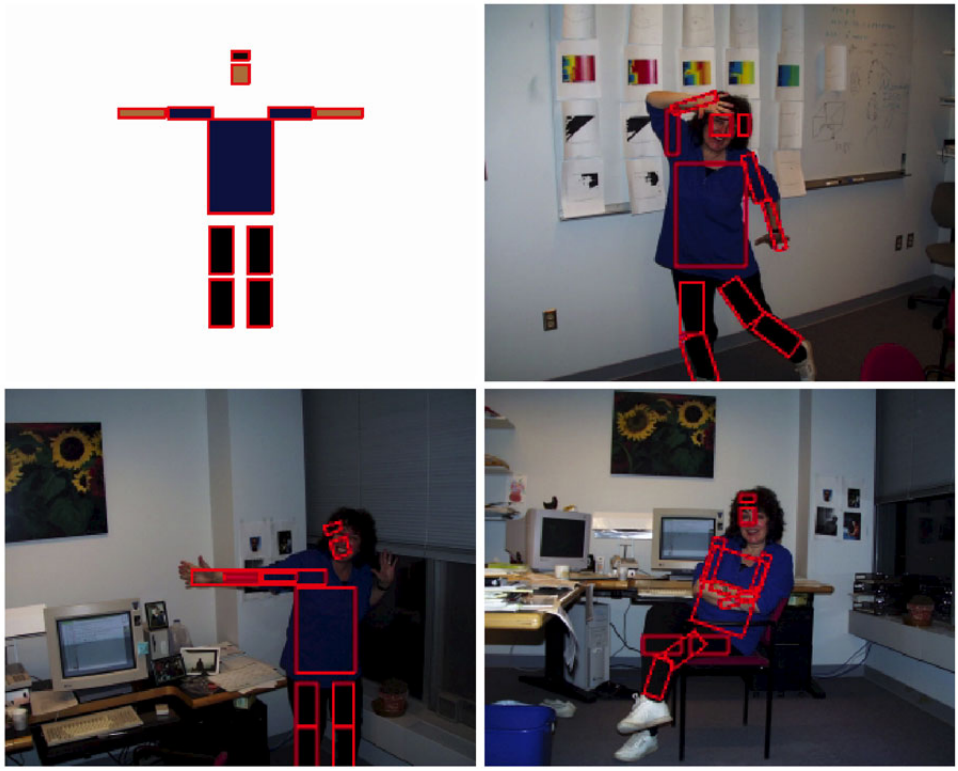

Figure 26.18. On the **top left**, a tree structured model of a person. Each segment is coloured with the image colour expected within this segment. The model attempts to find a configuration of these 11 body segments (9 limb segments, face and hair) that (a) matches these colours and (b) is configured like a person. This can be done with dynamic programming, as described in the text. The other three frames show matches obtained using the method. *Figure from "Efficient Matching of Pictorial Structures," P. Felzenszwalb and D.P. Huttenlocher, Proc. Computer Vision and Pattern Recognition 2000,*