CHAPTER

Template Matching

8

8.1 INTRODUCTION

In all previous chapters, the major concern was to assign an unknown pattern to one of the possible classes. The problem that will accompany us throughout this chapter is of a slightly different nature. We will assume that a set of reference patterns (*templates*) are available to us, and we have to decide which one of these reference patterns an unknown one (the *test pattern*) matches best. These templates may be certain objects in a scene or can be strings of patterns, such as letters forming words in a written text or words or phrases in a spoken text. Typically, such problems arise in speech recognition, in automation using robot vision, in motion estimation for video coding, and in image database retrieval systems, to name but a few. A reasonable first step to approaching such a task is to define a *measure* or a *cost* measuring the "distance" or the "similarity" between the (known) reference patterns and the (unknown) test pattern, in order to perform the matching operation known as *template matching*. We know by now that each pattern is expressed in terms of a vector or a matrix with elements the set of the selected features. Then why not use one of the already known distance measures, that is, Euclidean or Frobenius norms, and perform the matching operation based on the minimum distance? A little more thinking reveals that such a straightforward approach is not enough and something more is needed. This is the crucial point that makes template matching different and at the same time interesting.

To understand this issue better, let us consider a written text matching problem, that is, to identify which one from a set of written words is the word, say, "beauty." However, because of errors in the reading sensors, the specific test pattern may appear, for example, as "beety" or "beaut." In a speech recognition task, if a specific word is spoken by the same speaker a number of times, it will be spoken differently every time. Sometimes it will be spoken quickly, and the resulting pattern will be of short duration in time, sometimes slowly, and the pattern will be longer. Yet in all cases it is the "same" word spoken by the same person. In a scene analysis application, the object to be identified may be present in an image, but its location within the image is not known. In content-based image database retrieval systems,

queries often include the shape of an object. However, the shape provided by the user, most often, does not match exactly the shape of the object residing in the database images. The major goal of this chapter is to define measures that accommodate the distinct characteristics for each category of these problems. As is always the case with a textbook, only general directions and typical cases will be treated.

We will begin with the problem of string pattern matching and will deal with the scene analysis and shape recognition problems later on. The tasks, although they share the same goal, require different tools because of their different nature.

8.2 MEASURES BASED ON OPTIMAL PATH SEARCHING TECHNIQUES

We will first focus on a category of template matching, where the involved patterns consist of strings of identified symbols or feature vectors (string patterns). That is, each of the reference and test patterns is represented as a sequence (string) of measured parameters, and one has to decide which reference sequence the test-pattern matches best.

Let $\boldsymbol{r}(i), i = 1, 2, \ldots, I$, and $\boldsymbol{t}(j), j = 1, 2, \ldots, J$, be the respective feature vector sequences for a specific pair of reference and test patterns, where in general $I \neq J$. The objective is to develop an appropriate distance measure between the two sequences. To this end, we form a two-dimensional grid with the elements of the two sequences as points on the respective axes, that is, the reference string at the abscissa (i-axis) and the test one at the ordinate (j-axis). Figure 8.1 is an example for $I = 6$, $J = 5$. Each point of the grid (node) marks a correspondence between the respective elements of the two sequences. For example, node $(3, 2)$ maps the element $\boldsymbol{r}(3)$ to $\boldsymbol{t}(2)$. Each node (i, j) of the grid is associated with a *cost*, which is an appropriately defined function $d(i, j)$ measuring the "distance" between the respective elements of the strings, $\boldsymbol{t}(j)$ and $\boldsymbol{r}(i)$. A path through the grid from an initial node (i_0, j_0) to a final one (i_f, j_f) is an *ordered* set of nodes of the form

$$(i_0, j_0), (i_1, j_1), (i_2, j_2), \ldots, (i_f, j_f)$$

Each path is associated with an overall cost D defined as

$$D = \sum_{k=0}^{K-1} d(i_k, j_k)$$

where K is the number of nodes along the path. For the example of Figure 8.1, $K = 8$. The overall cost up to node (i_k, j_k) will be denoted by $D(i_k, j_k)$, and by convention we assume $D(0, 0) = 0$ and also $d(0, 0) = 0$. The path is said to be *complete* if

$$(i_0, j_0) = (0, 0), (i_f, j_f) = (I, J)$$

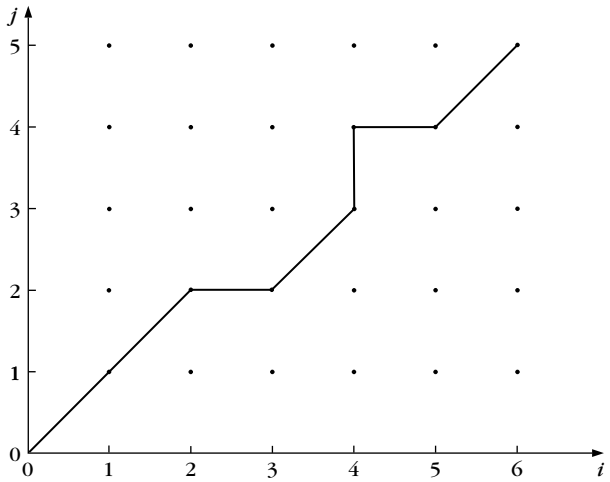

FIGURE 8.1

Each point along the path marks a correspondence between the respective elements of the test and reference patterns.

The distance[1] *between the two sequences is defined as the minimum D over all possible paths*. At the same time, the minimum cost path unravels the optimal pairwise correspondence between the elements of the two sequences, which is the crucial part, since the two sequences are of different lengths. In other words, the optimal path procedure makes the *alignment or warping* of the elements of the test string to the elements of the reference string, corresponding to the best matching score. Before we talk about the optimization procedure, we must point out that there are a number of variations of this scheme. For example, one may not impose the constraint of having necessarily a complete path but may adopt more relaxed constraints known as *end point constraints*. Furthermore, one could associate a cost not only with each node but also with each transition between nodes, making certain transitions more costly than others. In such cases, the cost at a node (i_k, j_k) also depends on the specific transition, that is, from which node (i_{k-1}, j_{k-1}) the (i_k, j_k) node was reached. Thus, the cost d is now of the form $d(i_k, j_k|i_{k-1}, j_{k-1})$ and the overall path cost is

$$D = \sum_k d(i_k, j_k | i_{k-1}, j_{k-1})$$

In some cases the overall path cost is defined as the product

$$D = \prod_k d(i_k, j_k | i_{k-1}, j_{k-1})$$

[1] The term *distance* here must not be interpreted with its strict mathematical definition.

Finally, there are cases where d is chosen so that maximization instead of minimization is required. Obviously, in all these variations appropriate initial conditions have to be adopted. Let us now come back to the optimization problem itself. To obtain the best path, one has to search all possible combinations of paths. However, this is a computationally costly procedure. *Dynamic programming algorithms* based on Bellman's principle are powerful tools that we will adopt to reduce the computational complexity.

8.2.1 Bellman's Optimality Principle and Dynamic Programming

Let the optimal path between an initial node (i_0, j_0) and a final one (i_f, j_f) be denoted as

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f)$$

If (i, j) is an intermediate node between (i_0, j_0) and (i_f, j_f), we will denote the optimal path constrained to pass through (i, j) as

$$(i_0, j_0) \xrightarrow[(i,j)]{opt} (i_f, j_f)$$

Bellman's principle states that [Bell 57]

$$(i_0, j_0) \xrightarrow[(i,j)]{opt} (i_f, j_f) = (i_0, j_0) \xrightarrow{opt} (i, j) \oplus (i, j) \xrightarrow{opt} (i_f, j_f)$$

where $\oplus$ denotes concatenation of paths. In other words, Bellman's principle states that the overall optimal path from (i_0, j_0) to (i_f, j_f) through (i, j) is the concatenation of the optimal path from (i_0, j_0) to (i, j) and the optimal path from (i, j) to (i_f, j_f). The consequence of this principle is that once we are at (i, j) through the optimal path, then to reach (i_f, j_f) optimally we need *to search only* for the optimal path from (i, j) to (i_f, j_f).

Let us now express this in a way that will be useful to us later on. Assume that we have departed from (i_0, j_0) and let the kth node of the path be (i_k, j_k). The goal is to compute the minimum cost required to reach the latter node. The transition to (i_k, j_k) has to take place from one of the possible nodes that are allowed to be in the $(k-1)$th position of the path (that is, the (i_{k-1}, j_{k-1}) node). This is important. For *each node* of the grid we assume that there is a set of allowed *predecessors*, defining the so-called *local constraints*. Bellman's principle readily leads to

$$D_{\min}(i_k, j_k) = \min_{i_{k-1}, j_{k-1}} [D_{\min}(i_{k-1}, j_{k-1}) + d(i_k, j_k | i_{k-1}, j_{k-1})] \tag{8.1}$$

Indeed, the overall minimum cost to reach node (i_k, j_k) is the minimum cost up to node (i_{k-1}, j_{k-1}) plus the extra cost of the transition from (i_{k-1}, j_{k-1}) to (i_k, j_k). Furthermore, the search for the minimum is constrained only within the set of allowable predecessors for the (i_k, j_k) node. This procedure is carried out for all the nodes of the grid. However, in many cases not all nodes of the grid are involved,

and the optimal path searching takes place among a subset of the nodes, which are defined via the so-called *global constraints*. The resulting algorithm is known as *dynamic programming*. Equation (8.1) has to be modified accordingly if the cost D is given in its multiplicative form and/or if maximization is required.

Let us now focus on our string pattern matching task and see how the *recursive* equation (8.1) is used to construct the optimal complete path.

Figure 8.2 illustrates the procedure. The set of nodes involved in the optimization (global constraints) is denoted as dark dots, and the local constraints, defining the allowable transitions among these nodes, are shown in the figure by the black lines. Having decided to search for the complete path and assuming $D(0,\ 0) = 0$, the respective costs $D(i_1,\ j_1)$ for all the allowed nodes involved in step $k = 1$ are computed, via (8.1) (in our case there are only two allowable nodes, $((1,\ 1)$ and $(1,\ 2))$. Subsequently, the costs of the (three) nodes at step $k = 2$ are computed, and the procedure is repeated until we arrive at the final node $(I,\ J)$. The sequence of transitions leading to the minimum $D(I,\ J)$ of the final node defines the minimum cost path, denoted by the red line. The optimal node correspondence, between the test and reference patterns, can then be unraveled by *backtracking the optimal path*. In the example of Figure 8.2, each step k of the recursion involves only nodes with the same abscissa coordinate, which reflects the local constraints

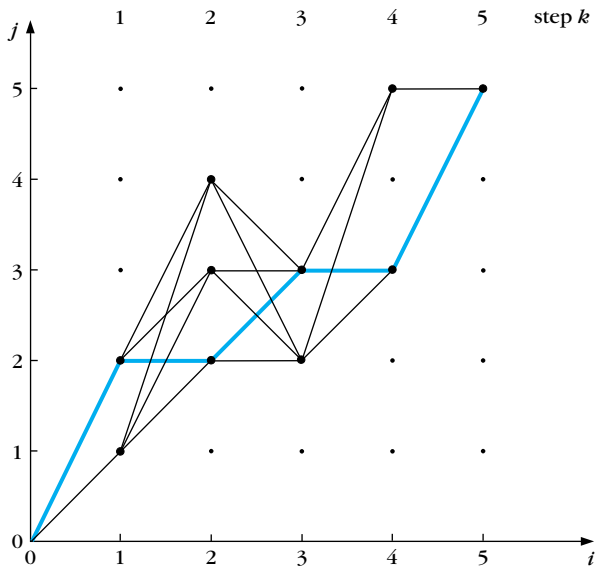

FIGURE 8.2

The optimal path (red line) is constructed by searching among all allowable paths, as defined by the global and local constraints. The optimal node correspondence, between the test and reference patterns, is unraveled by backtracking the optimal path.

adopted. In general, this is not necessary, and more involved topologies may be used. However, the philosophy of the search for the minimum remains the same. In the following subsections, we will apply the procedure in two different popular application areas.

Example 8.1

Figure 8.3 shows the optimal paths (black lines) to reach the nodes at step $k = 3$ starting from the nodes at step $k = 0$. The grid contains three nodes per step. Only the optimal paths, up to step $k = 3$, have been drawn. The goal of this example is to extend the previous paths to the next step and compute the optimal paths terminating at the three nodes at step $k = 4$. Bellman's principle will be employed. Assume that the accumulated costs of the optimal paths $D_{\min}(3, j_3), j_3 = 0, 1, 2$ at the respective nodes are:

$$D_{\min}(3, 0) = 0.8, D_{\min}(3, 1) = 1.2, D_{\min}(3, 2) = 1.0 \tag{8.2}$$

We are also given the transition costs $d(4, j_4|3, j_3), j_3 = 0, 1, 2, j_4 = 0, 1, 2$, in the form of a transition matrix in Table 8.1. In other words, the transition cost, for example, from node $(3, 1)$ to node $(4, 2)$ is equal to 0.2. To obtain the optimal path to node $(4, 0)$ one has to combine the values given in (8.2) and the corresponding transition costs provided in Table 8.1. Thus

Total cost for the transition from $(3, 0)$ to $(4, 0)$ is equal to $0.8 + 0.8 = 1.6$.
Total cost for the transition from $(3, 1)$ to $(4, 0)$ is equal to $1.2 + 0.2 = 1.4$.
Total cost for the transition from $(3, 2)$ to $(4, 0)$ is equal to $1.0 + 0.7 = 1.7$.

Applying Eq. (8.1) shows that the optimal path, with the minimum accumulated cost, to reach node $(4, 0)$ is obtained via the transition from node $(3, 1)$. The reader can verify that the optimal paths to the nodes at step $k = 4$ are the ones shown in Figure 8.3. The optimal costs associated with nodes $(4, 1)$ and $(4, 2)$ are equal to 1.2 and 1.3, respectively. Transitions from step $k = 3$ to $k = 4$ are indicated by red lines. Note that, for the case of our example,

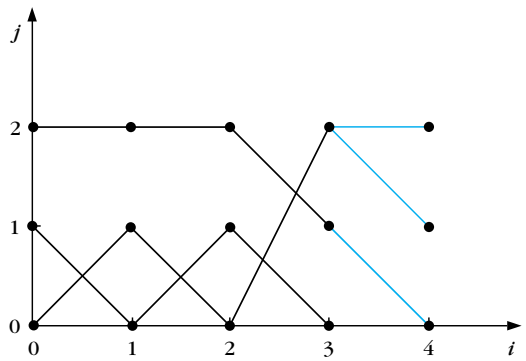

FIGURE 8.3

Optimal paths for the grid of example 8.1. Red lines correspond to the extensions of the optimal paths from step $k = 3$ to step $k = 4$.

Table 8.1 Transition Costs Between Nodes for the Example 8.1

Nodes	$(4,0)$	$(4,1)$	$(4,2)$
$(3,0)$	0.8	0.6	0.8
$(3,1)$	0.2	0.3	0.2
$(3,2)$	0.7	0.2	0.3

the path originating from node $(0,1)$ will not take place in the computations if more steps are added, that is, $k = 5,6,\ldots$. As we say, this path does not survive beyond step $k = 3$.

8.2.2 The Edit Distance

In this section, we will be concerned with patterns that consist of sets of *ordered symbols*. For example, if these symbols are letters, then the patterns are words from a written text. Such problems arise in automatic editing and text retrieval applications. Other examples of symbol strings occur in structural pattern recognition. Once the symbols of a (test) pattern have been identified, for example, via a reading device, the task is to recognize the pattern, searching for the best match of it against a set of reference patterns. The measure to be adopted for the matching procedure should take into account the following errors, which may occur during the symbol identification phase.

- Wrongly identified symbol (e.g., "befuty" instead of "beauty")
- Insertion error (e.g., "bearuty")
- Deletion error (e.g., "beuty")

Obviously, a combination of these errors may also occur. For the matching procedure we will adopt the philosophy behind the so-called *variational similarity*. In other words, the similarity between two patterns is based on the "cost" associated with converting one pattern to the other. If the patterns are of the same length, then the cost is directly related to the number of symbols that have to be changed in one of them so that the other pattern results. More interest arises when the two patterns are not of equal length. In such cases symbols have to be either deleted or inserted at certain places of the test string. The location where deletions or insertions are to be made presupposes an optimal alignment (warping) among the symbols of the two patterns. The Edit distance [Dame 64, Leven 66] between two string patterns A and B, denoted $D(A,B)$, is defined as *the minimum total number of changes C, insertions I, and deletions R* required to change pattern A into pattern B,

$$D(A,B) = \min_j [C(j) + I(j) + R(j)] \tag{8.3}$$

where j runs over all possible combinations of symbol variations in order to obtain B from A. To elaborate a bit, note that there is more than one way to change, say, "beuty" to "beauty." For example, one can either insert "a" after "e" or change "u" to "a" and then insert "u."

We will employ the dynamic programming methodology to compute the required minimum in (8.3). To this end, we form the grid by placing the symbols of the reference pattern in the abscissa axis and the test pattern in the ordinate one. Figure 8.4 demonstrates the procedure via four examples. As already pointed out, the first step in an optimal path searching procedure via dynamic programming

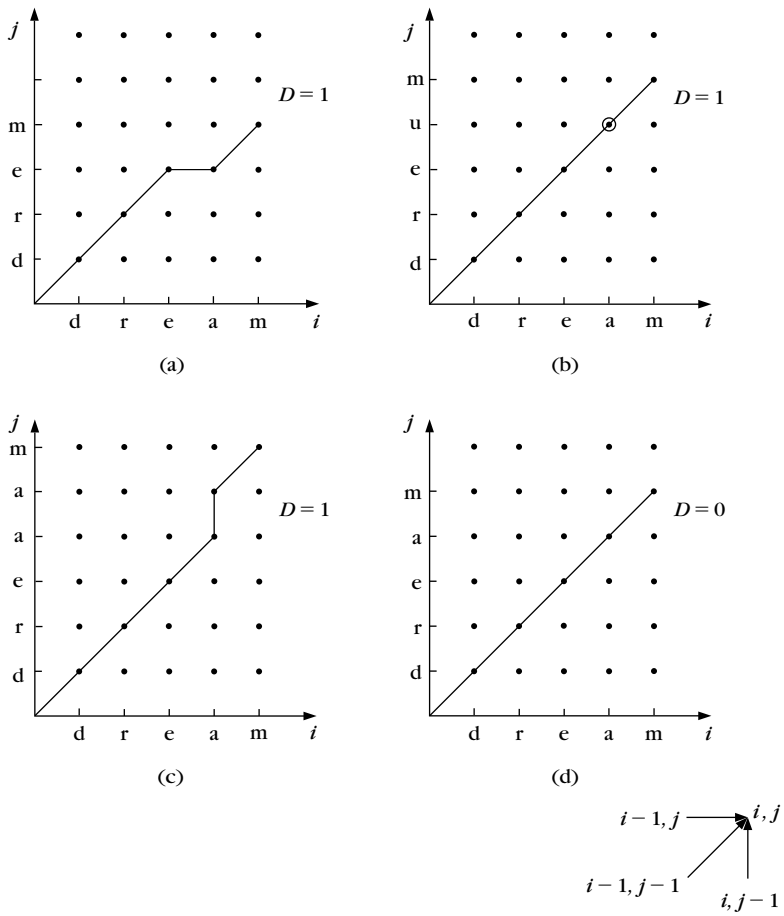

FIGURE 8.4

Computation of the Edit distance with (a) an insertion, (b) a change, (c) a deletion, and (d) an equality. The local constraints are shown at the bottom right corner.

techniques is to state the node transition constraints imposed by the problem. For our case of interest, the following constraints are adopted.

- The cost $D(0,\ 0)$ of the $(0,\ 0)$ node is zero.
- A complete path is searched.
- Each node $(i,\ j)$ can be reached only through three allowable predecessors, that is,

$$(i-1,\ j), \quad (i-1,\ j-1), \quad (i,\ j-1)$$

as indicated at the bottom of Figure 8.4.

The costs associated with the above three transitions are:

1. Diagonal transitions:

$$d(i,\ j|i-1,\ j-1) = \begin{cases} 0 & \text{if } r(i) = t(j) \\ 1 & \text{if } r(i) \neq t(j) \end{cases}$$

That is, the cost of a transition is zero if the symbols corresponding to the $(i,\ j)$ node are the same and unity if they are different; hence a symbol change has to take place.

2. Horizontal and vertical transitions:

$$d(i,\ j|i-1,\ j) = d(i,\ j|i,\ j-1) = 1$$

The meaning of horizontal transitions is that they attempt alignment of the two strings by insertion of a symbol; see Figure 8.4a. Thus, they add to the cost, because they imply local mismatch. Similarly, vertical transitions add to the cost because they imply deletions, Figure 8.4c.

Incorporating these constraints and the distance (8.3) in a dynamic programming procedure, the following algorithm results.

Algorithm for Computing the Edit Distance

- $D(0, 0) = 0$
- For $i = 1$ to I
  - $D(i, 0) = D(i-1, 0) + 1$
- End { For }
- For $j = 1$ to J
  - $D(0, j) = D(0, j-1) + 1$
- End { For }

- For $i = 1$ to I
  - For $j = 1$ to J
    - $c1 = D(i-1, j-1) + d(i,j|i-1,j-1)$
    - $c2 = D(i-1, j) + 1$
    - $c3 = D(i, j-1) + 1$
    - $D(i,j) = \min(c1, c2, c3)$
  - End { For }
- End { For }
- $D(A,B) = D(I,J)$

In other words, we first compute the minimum cost for reaching *each node* of the grid, starting at (0, 0), and the optimal (complete) path is subsequently constructed. Figure 8.4 shows the respective minimum cost paths and the resulting Edit distances for each of the cases. Verify that any other path for the examples of Figure 8.4 results in a higher cost.

The Edit distance is also known as *Levenstein distance*. Over the years a number of variants of the previous basic Edit distance scheme have been suggested to better address problems rising in various applications. In [Ocud 76] the cost for a change of one symbol to another is allowed to take values different to one, depending on the dependence between different symbols in different applications. For example, for the spelling correction task, it is reasonable to assume that changing an "a" to a "q" results in lower cost than changing an "a" to a "b." This is because in touch typing the letters "a" and "q" are typed using the same finger whereas "a" and "b" are not. Another generalization as suggested in [Seni 96] allows for merges, splits, and two-letter substitutions in the context of handwriting recognition.

A drawback of the basic Edit distance scheme is that it takes no account of the length of the string sequences that are compared. Thus, for example, if two string sequences differ in one symbol, their Edit distance will be equal to one regardless of their length being, say, equal to two or fifty. However, common sense leads us to assume that in the latter case the two strings are more similar than in the former, for which the two sequences share only one out of two symbols. In [Marz 93] the normalization by the length of the corresponding optimal path in the grid is proposed to account for the length of the involved sequences.

In [Mei 04] a variant called *Markov Edit distance* is defined that accounts for the local interactions among adjacent symbols. For example, this modified Edit distance assigns a lower cost to symbol changes in the test pattern if these are reshuffles of the corresponding subpattern in the reference pattern. This is natural since in practice it is not uncommon for one to mess up locally in typing. Taking this into consideration, comparing the reference pattern "beauty" with the test pattern "beuaty" will result in lower Markov Edit distance than comparing the same reference pattern with "besrty," in contrast to the basic Edit distance that is equal to two for both cases.

The Edit distance and its variants have been used in a number of applications, where the problem can be posed as a string matching, such as polygon matching ([Koch 89]), OCR ([Tsay 93, Seni 96]), stereo vision ([Wang 90]), computational biology, and genome sequence matching ([Durb 97]).

8.2.3 Dynamic Time Warping in Speech Recognition

In this section, we highlight the application of dynamic programming techniques in speech recognition. We will focus on the simpler form of the task, known as *discrete or isolated word recognition* (IWR). That is, we will assume that the spoken text consists of discrete words, well isolated with sufficient silent periods between them. In tasks of this type, it is fairly straightforward to decide where, in time, one word finishes and another one starts. This is not, however, the case in the more complex *continuous speech recognition* (CSR) systems, where the speaker speaks in a natural way and temporal boundaries between words are not well defined. In the latter case, more elaborate schemes are required (e.g., [Silv 90, Desh 99, Neg 99]). When words are spoken by a single speaker and the purpose of the recognition system is to recognize words spoken by this person, then the recognition task is known as *speaker-dependent recognition*. A more complex task is *speaker-independent recognition*. In the latter case, the system must be trained using a number of speakers and the system must be able to generalize and recognize words spoken by people outside the training population.

At the heart of any IWR system are a set of known reference patterns and a distance measure (recall the footnote 1 in Section 8.2). Recognition of an unknown test pattern is achieved by searching for the best match between the test and each of the reference patterns, on the basis of the adopted measure.

Figures 8.5a and 8.6a show the plots of two time sequences resulting from the sampling of the word "love," spoken twice by the same speaker. The samples were taken at the output of a microphone at a sampling rate of 22,050 Hz. Although it is difficult to describe the differences, these are readily noticeable. Moreover, the two spoken words are of different duration. The arrows indicate (approximately) the intervals in which the spoken segments lie. The intervals outside the arrows correspond to silent periods. Specifically, the sequence in Figure 8.6a is 0.4 second long, and the sequence in Figure 8.5a is 0.45 second long. Furthermore, it is important to say that this is not the result of a simple linear time scaling. On the contrary, a *highly nonlinear mapping* is required to obtain a match between these two "*same*" words spoken by the *same* person. For comparison, Figure 8.6b shows the plot of the time sequence corresponding to another word, "kiss," spoken by the same speaker.

We will resort to dynamic programming techniques to unravel the nonlinear mapping (warping) required to achieve the optimal match between a test and a reference pattern. To this end, we must first express the spoken words as sequences (strings) of appropriate feature vectors, $\boldsymbol{r}(i)$ $i = 1, \ldots, I$, for the reference pattern and $\boldsymbol{t}(j)$, $j = 1, \ldots, J$, for the test one. Obviously, there is more than one way to choose the feature vectors. We will focus on Fourier transform

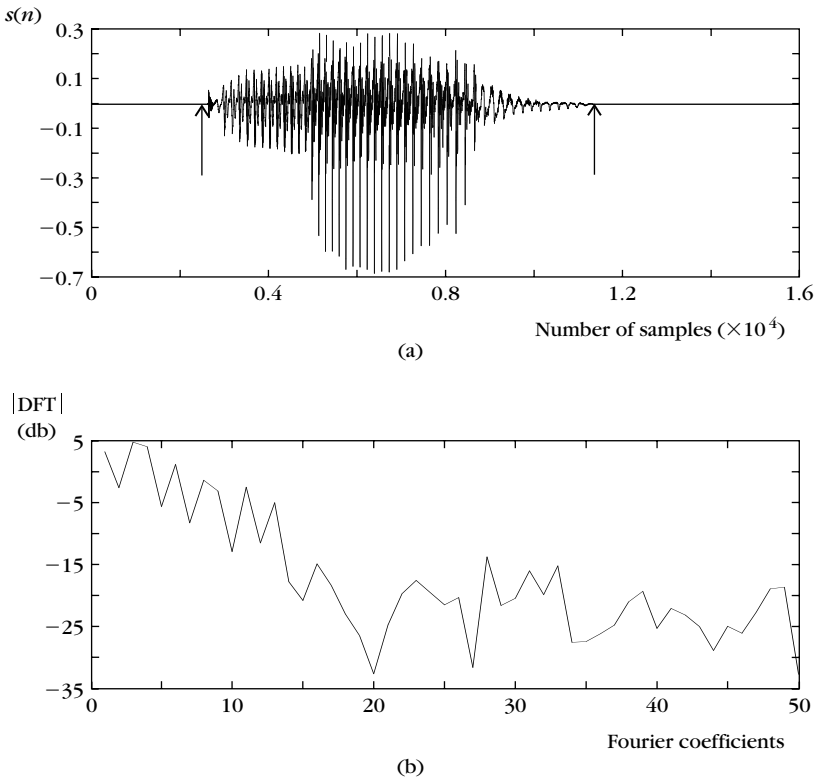

FIGURE 8.5

Plots of (a) the time sequence corresponding to the word "love" and (b) the magnitude of the DFT, in dB, for one of its frames.

features. Each of the time sequences involved is divided into successive overlapped time frames. In our case, each frame is chosen to be $t_f = 512$ samples long and the overlap between successive frames is $t_0 = 100$ samples, as shown in Figure 8.7. The resulting number of frames for the speech segment shown in Figure 8.5a is $I = 24$, and for the other two they are $J = 21$ (Figure 8.6a) and $J = 23$ (Figure 8.6b), respectively. We assume that the former is the reference pattern and the latter two the test patterns. Let $x_i(n), n = 0, \ldots, 511$, be the samples for the ith frame of the reference pattern, with $i = 1, \ldots, I$. The corresponding DFT is given as

$$X_i(m) = \frac{1}{\sqrt{512}} \sum_{n=0}^{n=511} x_i(n) \exp\left(-j \frac{2\pi}{512} mn\right), \quad m = 0, \ldots, 511$$

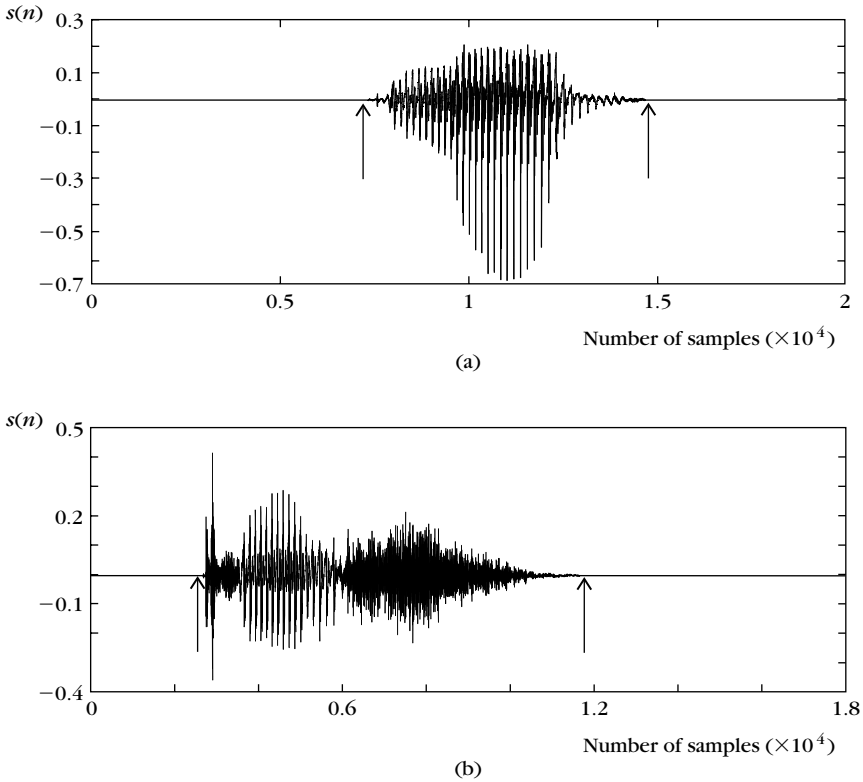

FIGURE 8.6

Plots of the time sequences resulting from the words (a) "love" and (b) "kiss," spoken by the same speaker.

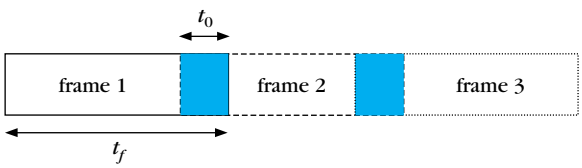

FIGURE 8.7

Successive overlapping frames for computation of the DFT feature vectors.

Figure 8.5b shows the magnitude of the DFT coefficients for one of the I frames of the reference pattern. The plot is a typical one for speech segments. The magnitude of the higher DFT coefficients is very small, with little contribution to the signal. This justifies use of the first l DFT coefficients as features, where usually

$l \ll t_f$. In our case $l = 50$ was considered to be sufficient. Thus, the vector sequence becomes

$$\boldsymbol{r}(i) = \begin{bmatrix} X_i(0) \\ X_i(1) \\ \vdots \\ X_i(l-1) \end{bmatrix}, \quad i = 1, \ldots, I \tag{8.4}$$

The feature vectors $\boldsymbol{t}(j)$ for each of the test patterns are formed in a similar way. The choice of the DFT coefficients as features is just one of various possibilities that have been suggested and used over the years. Other popular alternatives include the parameters from an AR modeling of the speech segment, the *ceptral coefficients* (inverse DFT of the logarithm of the magnitude of the DFT coefficients), and so on (e.g., [Davi 80, Dell 93]). Having completed the preprocessing and feature selection, the reference and test patterns are expressed as (ordered) sequences of feature vectors $\boldsymbol{r}(i)$ and $\boldsymbol{t}(j)$. Our goal becomes to compute the best match among the frames of the test and reference patterns. That is, the test pattern will be *stretched in time* (one test frame corresponds to more than one frame of the reference patterns) or *compressed in time* (more than one test frame corresponds to one frame of the reference pattern). This optimal alignment of the vectors in the two string patterns will take place via the dynamic programming procedure. To this end, we first locate the vectors of the reference string along the abscissa and those of the test pattern along the ordinate. Then, the following need to be determined:

- Global constraints
- Local constraints
- End-point constraints
- The cost d for the transitions

Various assumptions can be adopted for each of these, leading to different results with relative merits. In the sequel, we will focus on some widely used cases.

End-Point Constraints

In our example, we will look for the optimal complete path that starts at (0, 0) and ends at (I, J) and whose first transition is to the node (1, 1). Thus, it is implicitly assumed that the end points of the speech segments (i.e., $\boldsymbol{r}(1)$, $\boldsymbol{t}(1)$ and $\boldsymbol{r}(I)$, $\boldsymbol{t}(J)$) match to a fair degree. These can be the vectors resulting from the silent periods just before and just after the speech segments, respectively. A simple variation of the complete path constraints results if the end points of the path are not specified *a priori* and are assumed to be located within a distance ϵ from points (1, 1) and (I, J). It is left to the optimizing algorithm to locate them.

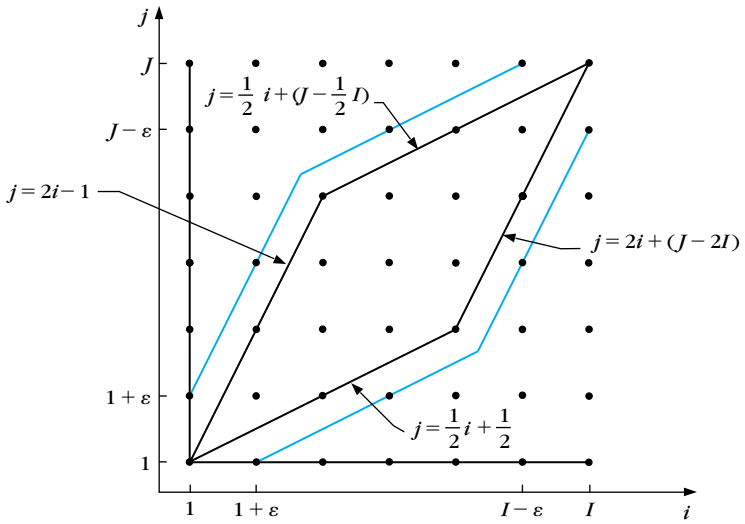

FIGURE 8.8

Itakura global constraints. The maximum compression/expansion factor is 2, and it determines the slope of the boundary line segments. The red lines correspond to the same global constraints when the relaxed end-point constraints are adopted.

Global Constraints

The global constraints define the region of nodes that are searched for the optimal path. Nodes outside this region are not searched. Basically, the global constraints define the overall stretching or compression allowed for the matching procedure. An example is shown in Figure 8.8. They are known as *Itakura constraints* and impose a maximum factor of 2 for any expansion or compression of the test with respect to the reference pattern. The allowable nodes are then located within the parallelogram shown in Figure 8.8 by the black line. The red lines correspond to the same global constraints when the relaxed end-point constraints, mentioned before, are adopted. Observe from the figure that paths across the sides of the parallelogram compress or expand corresponding frame intervals by a factor of 2, and this is the maximum possible factor attained. This constraint is usually reasonable and at the same time it reduces the number of nodes to be searched for the optimal path substantially. If $I \approx J$, then it is not difficult to show that the number of grid points to be searched is reduced by approximately one-third.

Local Constraints

These constraints define the set of predecessors and the allowable transitions to a given node of the grid. Basically, they impose limits for the maximum expansion/

compression rates that successive transitions can achieve. A property that any local constraint must satisfy is *monotonicity*. That is,

$$i_{k-1} \leq i_k \quad \text{and} \quad j_{k-1} \leq j_k$$

In other words, all predecessors of a node are located to its left and south. This guarantees that the matching operation follows the natural time evolution and avoids confusing, for example, the word "from" with the word "form."

Two examples of nonmonotonic paths are shown in Figure 8.9. A popular set of local constraints, known as the Itakura constraints [Itak 75], is shown in Figure 8.10. The maximum achievable expansion (compression) rate over a local path is measured by the associated *slope*, which is defined as the *maximum* ratio of the total change Δi, in the ith direction, to the total change Δj, in the jth direction, over the local path. The slope for the Itakura constraints is 2, and it is the result of a (repetitive) transition of the type $(i - 1, j - 2)$ to (i, j). Another notable characteristic of the Itakura constraints is that horizontal transitions are allowed, but *not successively*, and this is indicated by the cross over the arrow. Thus, Itakura constraints do not allow long horizontal paths, corresponding to ∞ slopes. Finally, these constraints allow the path to skip at *most one* feature vector in the test pattern string, that is, the one at the $j - 1$ position of the ordinate axis, and the path jumps from $(i - 1, j - 2)$ to (i, j). In contrast, feature vectors in the reference string are not skipped, and all take part in the optimal path. Such constraints are known as *asymmetrical*.

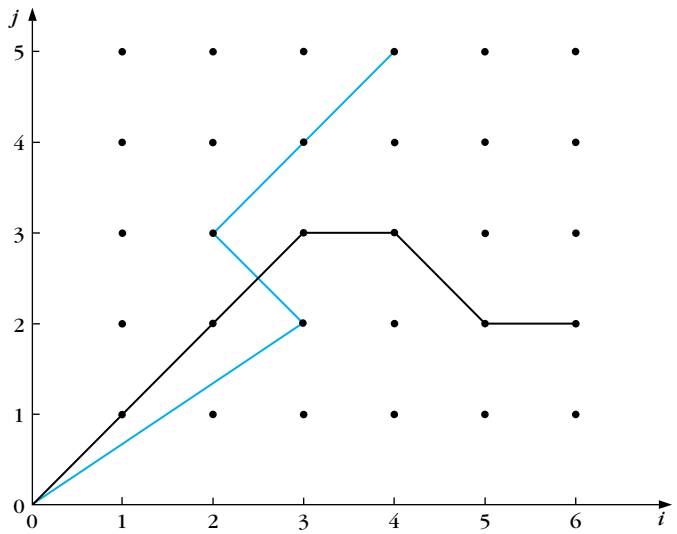

FIGURE 8.9

Examples of nonmonotonic paths. Such paths are not allowed and are not considered in the search for the optimum.

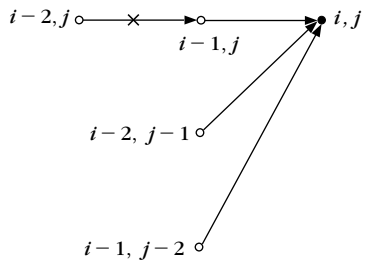

FIGURE 8.10

The Itakura local constraints. Two successive horizontal transitions are not allowed.

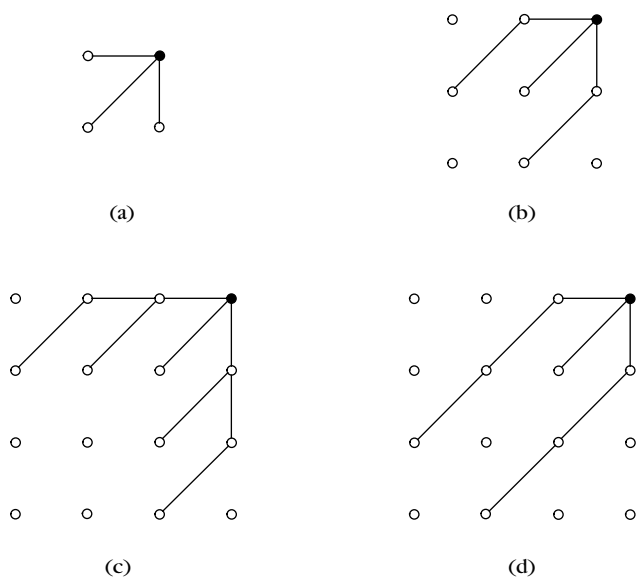

FIGURE 8.11

The Sakoe and Chiba local constraints.

A number of alternative local constraints have also been suggested and used in practice. Figure 8.11 shows four different types of constraints considered by Sakoe and Chiba [Sako 78]. For the type in Figure 8.11a, there is no limit in the rate of expansion/compression, since successive horizontal or vertical transitions can take place, until of course one falls outside the region defined by the global constraints. In contrast, in Figure 8.11b horizontal (vertical) transitions are allowed *only after* a diagonal transition and in Figure 8.11d after two successive diagonal transitions. In Figure 8.11c, at most two successive horizontal (vertical) transitions are allowed only after a diagonal one. The slopes for each of the constraints in Figures 8.11a, b, c and d are ∞, 2, 3, and 3/2, respectively (Problem 8.2).

For a more detailed treatment of the topic, the interested reader may consult [Dell 93, Silv 90, Myer 80].

The Cost

A commonly used cost, which we will also adopt here, is the Euclidean distance between $\boldsymbol{r}(i_k)$ and $\boldsymbol{t}(j_k)$, corresponding to node (i_k, j_k), that is,

$$d(i_k, j_k|i_{k-1}, j_{k-1}) = \|\boldsymbol{r}(i_k) - \boldsymbol{t}(j_k)\| \equiv d(i_k, j_k)$$

In this, we assume that no cost is associated with the transitions to a specific node, and the cost depends entirely on the feature vectors corresponding to the respective node. Other costs have also been suggested and used [Gray 76, Gray 80, Rabi 93]. More recently ([Pikr 03]) used a cost that accounts for the most commonly encountered errors (e.g., different players style) in the context of music recognition. Finally, it must be stated that often a normalization of the overall cost D is carried out. This is to compensate for the difference in the path lengths, offering "equal" opportunities to all of them. A logical choice is to divide the overall cost D by the length of each path [Myer 80].

The resulting overall costs for the two test patterns of Figure 8.6, against the reference pattern of Figure 8.5, using the Itakura constraints, were $D = 11.473$, $D = 25.155$, respectively. Thus, the overall cost for the word "love" is lower than the overall cost for the word "kiss," and our procedure has recognized the spoken word correctly. The resulting normalized overall costs after dividing by the number of nodes along each path were 0.221 and 0.559, respectively.

8.3 MEASURES BASED ON CORRELATIONS

The major task to be addressed in this section can be summarized as follows: "Given a block of recorded data, find *whether* a specific known (reference) pattern is contained within the block and *where* it is located."A typical application of this is found in scene analysis, when we want to search for specific objects within the image. Such problems arise in many applications, such as target detection, robot vision, and video coding. For example, in video coding a major step is that of *motion estimation*—that is, the process of locating *corresponding pixels* (of the same moved object) among successive image frames at different time instants. This step is then followed by the *motion compensation* stage, which compensates for the displacement of moving objects from one frame to another. One then codes the frame difference

$$e(i, j, t) = r(i, j, t) - r(i - m, j - n, t - 1)$$

where $r(i, j, t)$ are the pixel gray levels of the image frame at time t and $r(i - m, j - n, t - 1)$ the *corresponding* pixel values at spatial locations $i - m$,

$j - n$ of the previous frame at time $t - 1$. In this way, we code only the *new information* contained at the latest frame, avoiding redundancies.

Let us assume that we are given a reference pattern expressed as an $M \times N$ image array $r(i, j), i = 0, \ldots, M - 1, j = 0, \ldots, N - 1$, and an $I \times J$ image array $t(i, j), i = 0, \ldots, I - 1, j = 0, \ldots, J - 1$, where $M \leq I, N \leq J$. The goal is to develop a measure for detecting an $M \times N$ subimage within $t(i, j)$ that matches best the reference pattern $r(i, j)$. To this end, the reference image $r(i, j)$ is superimposed on the test image, and *it is translated* to all possible positions (m, n) within it. For each of the points (m, n), the mismatch between $r(i, j)$ and the $M \times N$ subimage of $t(i, j)$ is computed according to

$$D(m, n) = \sum_{i=m}^{m+M-1} \sum_{j=n}^{n+N-1} |t(i,j) - r(i - m, j - n)|^2 \tag{8.5}$$

Template matching is conducted by searching for the location (m, n) for which $D(m, n)$ is minimum. Let us now give this a computationally more attractive form. Equation (8.5) is equivalent to

$$D(m, n) = \sum_i \sum_j |t(i,j)|^2 + \sum_i \sum_j |r(i,j)|^2 - 2 \sum_i \sum_j t(i, j) r(i - m, j - n) \tag{8.6}$$

The second summand is constant for a given reference pattern. *Assuming that the first one does not change much across the image*, that is, there is not much variation of the pixel gray levels over the test image, the minimum of $D(m, n)$ is achieved when

$$c(m, n) = \sum_i \sum_j t(i,j) r(i - m, j - n) \tag{8.7}$$

is maximum for all possible locations (m, n). The quantity $c(m, n)$ is nothing other than a cross-correlation sequence between $t(i, j)$ and $r(i, j)$ computed at the point (m, n). In cases for which the assumption of little gray-level variation is not valid, this measure is very sensitive to gray-level variations within $t(i, j)$. In such cases the *cross-correlation coefficient*, defined as

$$c_N(m, n) = \frac{c(m, n)}{\sqrt{\sum_i \sum_j |t(i,j)|^2 \sum_i \sum_j |r(i,j)|^2}} \tag{8.8}$$

is a more appropriate measure. Here, $c_N(m, n)$ is a normalized version of $c(m, n)$, and variations in $t(i, j)$ tend to cancel out. Recall now the Cauchy–Schwarz inequality

$$\left| \sum_i \sum_j t(i, j) r(i - m, j - n) \right| \leq \sqrt{\sum_i \sum_j |t(i, j)|^2 \sum_i \sum_j |r(i, j)|^2}$$

Equality holds *if and only if*

$$t(i,j) = \alpha r(i-m, j-n), \quad i = m, \ldots, m+M-1 \text{ and}$$
$$j = n, \ldots, n+N-1$$

with α being an arbitrary constant. Hence, $c_N(m, n)$ is always less than unity and achieves its maximum value of one only if the (test) subimage is the same (within a scaling factor) as the reference pattern.

In our discussion so far, we have assumed that the reference pattern has only been translated within $t(i, j)$ and no rotation or scaling has been involved. In applications such as video coding, this is a valid assumption and it has been adopted in the video coding standards [Bhas 95]. However, this is not always the case and the technique has to be modified. One way is to describe the reference and test subimages in terms of invariant moments and measure the similarity using correlations involving these moments [Hall 79] (Problem 8.4). Another rotation- and scale-invariant technique, using a combination of the Fourier and Mellin transforms, is described in [Scha 89]. This technique tries to exploit the translation invariance of the magnitude of the Fourier transform (already discussed in Chapter 7) and the scale invariance of the Mellin transform ([Ravi 95], Problem 8.5). Another path, which of course demands high computational resources, is to have a set of distorted (e.g., rotated and scaled) reference templates to cover all possibilities. Correlation matching will then reveal the best match between a test pattern and one of the reference templates. A computationally more attractive technique is to employ the Karhunen–Loève transform [Ueno 97]. The main idea is that rotated templates are highly correlated, and each of them can be approximated by its projection onto a lower dimension eigenspace, using the most significant eigenvectors of their correlation matrix. Matching of an unknown pattern with the template of the right orientation is performed in the lower dimensional space, leading to substantial computational savings.

Example 8.2

The image $t(i, j)$ in Figure 8.12a contains two objects, a screwdriver and a hammer. The latter is the object that we want to search for in the image. The reference image is shown at the top right corner of Figure 8.12a. The dotted area represents the general (m, n) position of the reference image when it is superimposed on the test one. Figure 8.12b shows the cross-correlation $c(m, n)$ between the two images. We readily observe that the maximum (black) occurs at the position (13, 66), that is, where the hammer is located in $t(i, j)$.

Computational Considerations

- In some cases, it is more efficient to compute the cross-correlation via its Fourier transform. Recall that in the frequency domain (8.7) is written as

$$C(k,l) = T(k,l)R^*(k,l) \tag{8.9}$$

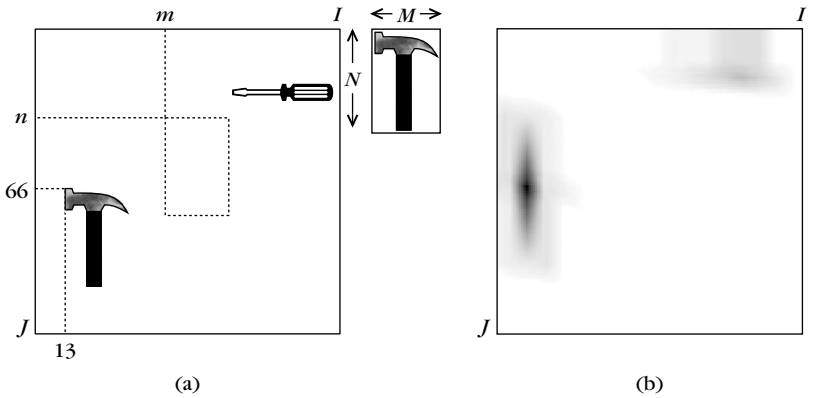

FIGURE 8.12

Example of (a) reference and test images and (b) their respective correlation.

where $T(k, l)$, $R(k, l)$ are the DFT transforms of $t(i, j)$ and $r(i, j)$, respectively, with "$*$" denoting complex conjugation. Of course, in order to write (8.9), both images must be of the same size. If they are not, which is usually the case, a number of zeros must be appended to extend the smaller sized image. $c(m, n)$ is obtained via the inverse DFT of $C(k, l)$. Taking into account the computational efficiency of the FFT, this procedure may lead to savings depending, of course, on the relative size of M, N and I, J.

- A major computational load in correlation-based template matching is searching over the pixels of $t(i, j)$ in order to locate the maximum correlation. Usually, the search is restricted within a rectangle $[-p, p] \times [-p, p]$ centered at a point (x, y) in $t(i, j)$. For example, in video coding, if the $M \times N$ block is centered at a position (x, y) in the frame at time $t - 1$, then the current frame is searched within $(x \pm p, y \pm p)$. The value of p depends on the application. For broadcast TV $p = 15$ is sufficient. For sporting events (high motion) $p = 63$ is more appropriate. Thus, an exhaustive search for the maximum of $c(m, n)$, defined in (8.7), will require a number proportional to $(2p + 1)^2 MN$ additions and multiplications. This leads to a huge number of operations indeed (Problem 8.6). Thus, in practice, suboptimal heuristic searching techniques are usually adopted, which, although they do not guarantee locating the maximum, reduce the required number of operations substantially. There are two major directions. One is to reduce the search points and the other is to reduce the size of the involved images.

Two-Dimensional Logarithmic Search

Logarithmic Search Figure 8.13 shows the rectangular $[-p, p] \times [-p, p]$ searching area for the case of $p = 7$. The center of the rectangle is assumed to be the point $(0, 0)$. The cross-correlation computation is first performed at the center as well

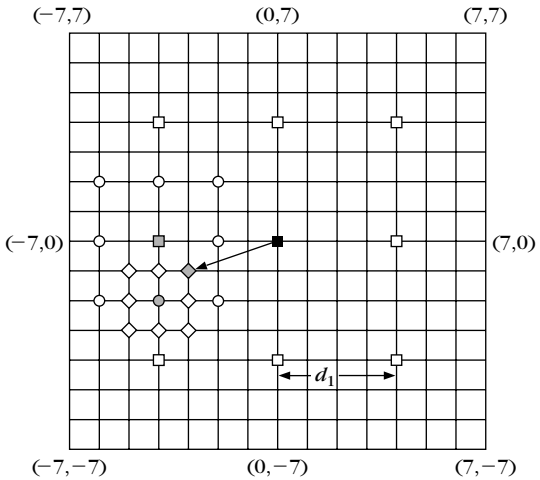

FIGURE 8.13

Logarithmic search to find the point of maximum cross-correlation.

as the eight points located on the perimeter of the inner $[-p/2, p/2] \times [-p/2, p/2]$ area ($p/2$ rounded to an integer). These points are denoted by a square. The spacing between these points is $d_1 = 4$ pixels, that is, $d_1 = 2^{k-1}$ and $k = \lceil \log_2 p \rceil$, where $\lceil \cdot \rceil$ denotes rounding to the first larger integer. For $p = 7$ we get $k = 3$ and $d_1 = 4$. We will demonstrate the procedure via an example. Let us assume that the largest cross-correlation value resulted at the position $(-4, 0)$ (shaded square). Then we consider this point as the center of a rectangle of size $[-p/4, p/4] \times [-p/4, p/4]$ ($[-2, 2] \times [-2, 2]$ in our case) and compute the correlation at the eight points of its perimeter. These points are denoted by a circle, and the spacing between them is now $d_2 = d_1/2$ (2). The process is repeated, and finally the computation is performed on the eight (diamond) points on the perimeter of the rectangle of size $[-1, 1] \times [-1, 1]$, which is centered at the optimum (of the previous step) shaded circle point. The spacing between the diamond points is $d_3 = 1$. The shaded diamond corresponds to the point with the maximum cross-correlation, and the process is terminated. The number of computations has now been reduced to $MN(8k + 1)$ operations, which is a substantial saving compared with the exhaustive search.

A variant of the two-dimensional logarithmic search is to search the i and j directions independently. The point whose coordinates are the resulting best values of i and j becomes the new origin of the coordinate system, and the search is repeated in the new i, j directions, with smaller spacing d. The process is repeated until the spacing becomes unity.

Hierarchical Search

The hierarchical search technique springs from the multiresolution concept considered in Chapter 6. Let us again consider an example.

- Step 1: A reference block of, say, 16×16 is given, and the search area is assumed to be the rectangle $[-p,\ p] \times [-p,\ p]$, centered at the point $(x,\ y)$ in the test image. We refer to level 0 versions of the images. Both the reference block and the test image are low-pass filtered and subsampled by 2, resulting in their level 1 versions. The total number of pixels in the level 1 versions has been reduced by 4. Level 1 versions are in turn low-pass filtered and subsampled, resulting in level 2 versions. In general, this process can continue.
- Step 2: At level 2 the search for the maximum takes place with the 4×4 low-pass version of the reference block. The search area in the level 2 low-pass version of the test image is the rectangle $[-p/4, p/4] \times [-p/4, p/4]$ centered at $(x/4, y/4)$. Either a full or a logarithmic search can be employed. Let (x_1, y_1) be the coordinates of the optimum, with respect to $(x/4, y/4)$.
- Step 3: At level 1, the search for the maximum is performed using the corresponding 8×8 version of the reference block. The search area, within the level 1 version of the test image, is the rectangle of size $[-1, 1] \times [-1, 1]$ centered at $(x/2 + 2x_1, y/2 + 2y_1)$, that is, nine pixels in total. This is because the eight pixels at the perimeter of this area were not involved at level 2, due to the subsampling (see Figure 6.23 of Chapter 6). The center point must also be included in order to have a fair comparison at this level for the search for the maximum. Let the maximum occur at $(x_2,\ y_2)$ with respect $(x/2, y/2)$.
- Step 4: At level 0 the search is performed using the original reference template, within the area of size $[-1, 1] \times [-1, 1]$ centered at $(x + 2x_2,\ y + 2y_2)$ in the test image. The location of the maximum is the final one and the process terminates.

The computational saving with this method depends on the number of levels, as well as the type of search adopted at the highest level (Problem 8.5). In general, hierarchical methods are very efficient from a computational point of view. This is gained at the expense of increased memory requirements, due to the various image versions that must be kept. A disadvantage of the method is that if small objects are present in the templates, they may disappear at the lowest resolution images due to the subsampling. Furthermore, the method cannot guarantee to find the global best match. In [Alkh 01] an alternative philosophy is suggested that results in the global best match. Computational savings are achieved by pruning the number of candidates for the best match in a level, using the results in a higher level and an appropriately chosen threshold value.

Sequential Method

A number of other techniques have also been suggested. For example, the *sequential search* method computes a variant of (8.5) directly. Specifically, define

$$D_{pq}(m, n) = \sum_{i=0}^{p-1} \sum_{j=0}^{q-1} |t(i + m, j + n) - r(i, j)| \tag{8.10}$$

Thus, the error is computed in a smaller and sequentially increasing window area, for $p, q = 1, 2, \ldots$ and $p \leq M, q \leq N$. The computations stop when $D_{pq}(m, n)$ becomes larger than a predetermined threshold. Then computations start in a different direction (m, n). Hence, saving is achieved, because for bad positions only a small number of computations need to be performed.

8.4 DEFORMABLE TEMPLATE MODELS

In the previous section, we considered the problem of searching for a known reference pattern (template) within a test image. We assumed that the template and the object, residing in the image, were identical. The only differences that were allowed to enter into our discussion were those imposed by a different orientation and/or scaling. However, there are many problems where we know *a priori* that the available template and the object we search for in the image may not look exactly the same. This may be due to varying imaging conditions, occlusion, and imperfect image segmentation. Furthermore, in a content-based image database retrieval system, the user may provide the system with a sketch of the shape of the object to be retrieved. Obviously, the sketch will not match exactly the corresponding object in the database images. Our goal in this section is to allow the template matching procedure to account for deviations between the reference template and the corresponding test pattern in the image. In our discussion, we will assume that the reference template is available in the form of an image array containing the object's boundary information (contour). That is, we will focus on shape information only. Extensions incorporating more information, for example, texture, are also possible.

Let us denote the reference template image array as $r(i, j)$. This is also known as *prototype*. The basic idea behind the *deformable template matching* procedure is simple: *Deform* the prototype and produce *deformed* variants of it. From a mathematical point of view a deformation consists of the application of a *parametric transform* $T_{\boldsymbol{\xi}}$ on $r(i,j)$ to produce a deformed version $T_{\boldsymbol{\xi}}[r(i,j)]$. Different values of the vector parameter $\boldsymbol{\xi}$ lead to different versions. From the set of the deformed prototype variants that can be generated, there will be one that "best" matches the test pattern. The goodness of fit is measured via a cost, which we will call the *matching energy* $E_m(\boldsymbol{\xi})$. Obviously, the goal is to choose $\boldsymbol{\xi}$ so that $E_m(\boldsymbol{\xi})$ is *minimum*. However, this is not enough. If, for example, the optimal set of parameters is such that the corresponding deformed template has been deformed to such an extent that it bears little resemblance to the original prototype, the method will be meaningless. Thus, one more term has to be taken into account in the optimization process. This is the cost measuring the "deformation," which the prototype needs to undergo in order to fit the test pattern. We will call this term of the cost *deformation energy* $E_d(\boldsymbol{\xi})$. Then the optimal vector parameter is computed so that

$$\boldsymbol{\xi} : \min_{\boldsymbol{\xi}} \{E_m(\boldsymbol{\xi}) + E_d(\boldsymbol{\xi})\} \tag{8.11}$$

In words, one could think of the boundary curve of the prototype as made by rubber. Then with the help of a pencil we deform the shape of the rubber curve to match the test pattern. The more we deform the shape of the prototype, the higher the energy we have to spend for it. This energy, quantified by $E_d(\boldsymbol{\xi})$, depends on the shape of the prototype. That is, it is an intrinsic property of the prototype, and this is the reason that it is also known as *internal energy*. The other energy term, $E_m(\boldsymbol{\xi})$, depends on the input data (test image), and we usually refer to it as *external energy*. The optimal vector parameter, $\boldsymbol{\xi}$, is chosen so that the best trade-off between these two energy terms is achieved. Sometimes, a weighting factor C is used to give preference to one of the two terms, and $\boldsymbol{\xi}$ is computed so that

$$\boldsymbol{\xi}: \min_{\boldsymbol{\xi}}\{E_m(\boldsymbol{\xi}) + CE_d(\boldsymbol{\xi})\} \tag{8.12}$$

Hence, in order to apply the above procedure in practice, one must have at one's disposal the following ingredients:

- A prototype
- A transformation procedure to deform the prototype
- The two energy function terms

Choice of the Prototype

This should be carefully chosen so that it is a (typical) representative of the various instances in which this object is expected to appear in practice. In a way, the prototype should encode the "mean shape" characteristics of the corresponding "shape class."

Deformation Transformation

This consists of a set of parametric operations. Let (x, y) be the (continuous) coordinates of a point in a two-dimensional image. Without loss of generality, assume that the image is defined in a square $[0, 1] \times [0, 1]$. Then each point (x, y) is mapped using a continuous mapping function, as

$$(x, y) \longrightarrow (x, y) + (D^x(x, y), D^y(x, y)) \tag{8.13}$$

For discrete image arrays a quantization step is necessary after the transformation. Different functions can be used to perform the above mapping. A set that has successfully been used in practice is ([Amit 91])

$$D^x(x,\ y) = \sum_{m=1}^{M}\sum_{n=1}^{N} \xi^x_{mn} e^x_{mn}(x,\ y) \tag{8.14}$$

$$D^y(x,\ y) = \sum_{m=1}^{M}\sum_{n=1}^{N} \xi^y_{mn} e^y_{mn}(x,\ y) \tag{8.15}$$

$$e^x_{mn}(x,\ y) = \alpha_{mn} \sin \pi nx \cos \pi my \tag{8.16}$$

$$e^y_{mn}(x, y) = \alpha_{mn} \cos \pi mx \sin \pi ny \tag{8.17}$$

for appropriately chosen values of M, N. The normalizing constants α_{mn} can be taken as

$$\alpha_{mn} = \frac{1}{\pi^2(n^2 + m^2)}$$

Other basis functions can also be used, such as splines or wavelets. Figure 8.14 shows a prototype for an object and three deformed versions obtained for the simplest case of the transformation model in (8.14)–(8.17), that is, $M = N = 1$.

Internal Energy

This should be minimum for no deformation, that is, for $\boldsymbol{\xi} = \mathbf{0}$. A reasonable choice, associated with the transformation functions considered above, is

$$E_d(\boldsymbol{\xi}) = \sum_m \sum_n ((\xi_{mn}^x)^2 + (\xi_{mn}^y)^2) \tag{8.18}$$

External Energy

Here again a number of choices are possible, measuring the goodness of fit between the test pattern and each one of the deformed template variants. For example, for a specific position, orientation, and scale of a deformed template, this energy term can be measured in terms of the distance of each point in the contour of the deformed

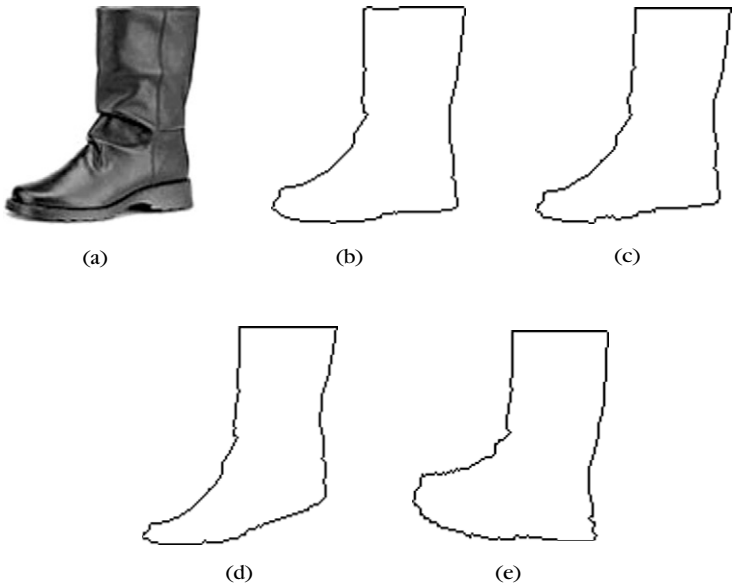

FIGURE 8.14

(a) A reference pattern, (b) its contour used as prototype, and (c), (d), (e) three of its deformed variants.

template from the nearest point in the contour of the test image, I. One way to achieve this is via the following energy function:

$$E_m(\boldsymbol{\xi},\ \boldsymbol{\theta},\ I) = \frac{1}{N_d}\sum_{i,j}(1 + \Phi(i,\ j)) \tag{8.19}$$

where $\boldsymbol{\theta}$ is the vector of the parameters defining the position, orientation, and scaling and N_d the number of contour pixels of the corresponding deformed template and

$$\Phi(i,j) = -\exp\left(-\rho(\delta_i^2 + \delta_j^2)^{1/2}\right) \tag{8.20}$$

where ρ a constant and (δ_i, δ_j) is the displacement of the (i,j) pixel of the deformed template from the nearest point in the test image. In [Jain 96] directional information is also incorporated into the cost.

Remarks

- One can arrive at (8.11) in a more systematic way via probabilistic arguments, that is, by seeking to maximize the a posteriori probability density of $(\boldsymbol{\xi}, \boldsymbol{\theta})$ given the test image array, that is,

$$p(\boldsymbol{\xi}, \boldsymbol{\theta}|I) = \frac{p(\boldsymbol{\xi}, \boldsymbol{\theta})p(I|\boldsymbol{\xi}, \boldsymbol{\theta})}{p(I)} \tag{8.21}$$

  where the Bayes rule has been employed. In this framework, (8.18) results if one assumes that the various parameters ξ_{mn}^x, ξ_{mn}^y are statistically independent and are normally distributed, for example, $p(\xi) = \mathcal{N}(0, \sigma^2)$. The higher the variance σ^2, the wider the range of the values that occur with high probability. To obtain (8.19)–(8.20) the model

$$p(I|\boldsymbol{\xi}, \boldsymbol{\theta}) = \alpha \exp\left(-E_m(\boldsymbol{\xi}, \boldsymbol{\theta}, I)\right) \tag{8.22}$$

  is adopted, where α is a normalizing constant [Jain 96].

- The cost in (8.11) is a nonlinear function, and its minimization can be achieved via any nonlinear optimization scheme. Besides complexity, the major drawback is the omnipresent danger that the algorithm will be trapped in a local minimum. In [Jain 96] a procedure has been suggested that builds around the gradient descent methodology (Appendix C). The idea is to adopt a multiresolution iterative approach and use larger values of ρ in (8.20) for the coarser levels and smaller values for the finer ones. This procedure seems to help the algorithm to avoid local minima, at an affordable computing cost.

- The methodology we described in this section belongs to a more general class of deformable template matching techniques, which builds around an available prototype. This is not the only available path. Another class of deformable models stems from an analytic description of the prototype shape, using a set of parameterized geometrical shapes, for example, ellipses or parabolic curves.

To delve deeper into these issues the reader may refer to the review articles [Jain 98, McIn 96, Cheu 02] and the references therein. [Widr 73, Fisc 73] seem to be the first attempts to introduce the concept of deformable models in computer vision.

- In the pattern recognition context, given an unknown test pattern, we seek to see to which one from a known set of different prototypes this matches best. For each prototype, the best deformed variant is selected, based on the minimum energy cost. The prototype that wins is the one whose best deformed variant results in the overall minimum energy cost.

8.5 CONTENT-BASED INFORMATION RETRIEVAL: RELEVANCE FEEDBACK

With the rapid development and spread of the Internet, a large corpus of information is stored and distributed over the Web. Search engines have become indispensable tools for searching and retrieving information in all possible forms, including text, images, audio and more recently video. The more traditional way of information retrieval is text-based; stored information is manually annotated by text descriptors, which are in turn used by a distributed database system to perform the information retrieval task. Such a procedure has the obvious drawback of requiring manual annotation, which, besides being time consuming and costly, is vulnerable to annotation inaccuracies and also to the subjectivity of the human perception. Due to the advances in pattern recognition, an alternative search procedure, known as *content-based* retrieval, is gaining in importance, and it is becoming more and more popular. Stored information is now indexed and searched based on its content. For example, in image retrieval, images can be indexed automatically by using features "describing" image content qualities such as color, texture and shapes. A music or speech segment could be represented in terms of a number of features such as those described in Section 7.5.

Content-based information retrieval is similar in concept to template matching, as introduced in this chapter. The goal is to search for and retrieve stored pieces of information, that is, patterns/templates that are most "similar" to the pattern presented as input to the search engine system. Similarity is quantified in terms of a similarity measure defined in the feature space. The similarity measures described at the beginning of this chapter may be possible candidates for some content-based retrieval tasks. A popular metric that has extensively been used is the weighted l_p metric between two feature vectors $\boldsymbol{x}$, $\boldsymbol{y}$, given by:

$$d(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{l} w_i |x_i - y_i|^p\right)^{\frac{1}{p}}$$

Obviously, for $p = 2$ and $w_i = 1,\ i = 1, 2, \ldots, l$, this becomes the Euclidean distance and for $p = 1$ the so called weighted l_1 (Manhattan) norm. A more detailed treatment of similarity/dissimilarity measures is given in Chapter 11.

A major disadvantage of such a contend-based approach is that search and subsequent retrieval is based solely on the derived features, which are usually referred to as *low-level* features. Humans, being much more intelligent than the machines they themselves develop, utilize a number of so-called *high-level* concepts when they come to recognize objects (patterns). The notion of *semantic gap* is usually adopted to express this discrepancy between the low-level features, derived from and describing the patterns, and the high-level descriptions that are meaningful to a human. Artificial Intelligence is the discipline that focuses on developing methodologies and techniques for machine high-level reasoning in association with the low-level derived features. However, so far, such techniques are feasible and applicable only to restricted domains and applications.

The goal of the current section is a more humble one, compared to the goals originally set in the field of artificial intelligence, and yet very interesting. Since learning machines cannot compete with the high-level concepts and reasoning of a human being, let the user be involved and become part of the learning "game". Such a methodology offers the system the advantage of exploiting the user's own way of conceptualizing the patterns, which are experienced through his or her senses. To this end, the search/retrieval session is divided into a number of consecutive loops. At every loop, the user provides *feedback* regarding the results by characterizing the retrieved patterns as either *relevant* or *irrelevant*. Relevance is usually defined by a characteristic that is shared by some of the patterns. It can be a perceptual characteristic or a more semantic one [Cruc 04]. Such a methodology is known as *relevance feedback* (RF). Since the user is directly involved, for the learning process to be useful in practice, convergence should be achieved within a few iteration steps and the search engine must operate in real time. In turn, this imposes the constraint that the selected (low-level) features must be as informative as possible. Thus the feature selection techniques, as exposed in Chapter 5, are of significance here too. One of the most successful paradigms for RF is the case of *content-based image retrieval* (CBIR), for which commercial products are already available; see, for example, [Liu 07].

A typical scenario, met in a number of RF tasks, is given below and it is schematically presented in Figure 8.15.

1. The system provides an initial set of patterns "similar" to the pattern presented by the user to the search engine (e.g., an image, a Web page, or an audio segment from a piece of music).

2. The user marks a number from the returned patterns as "relevant" or "irrelevant."

3. A classification procedure is used to "learn" the user's feedback.

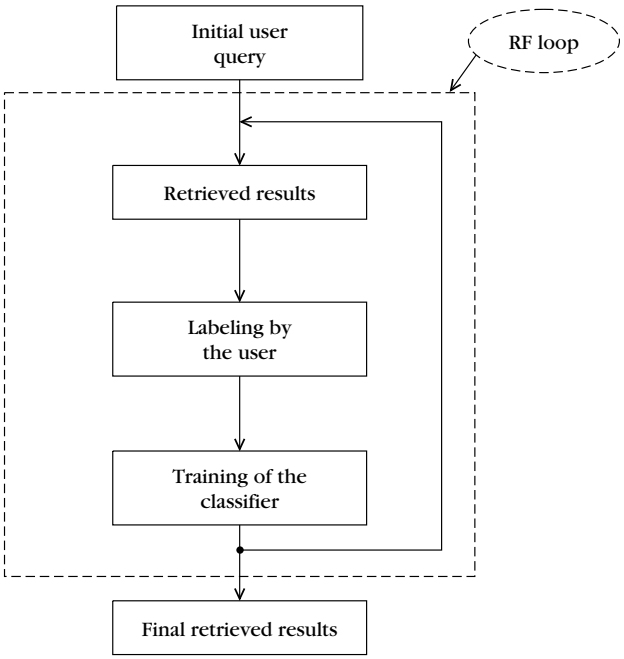

FIGURE 8.15

Block diagram illustrating the basic steps involved in a relevance feedback task.

Steps 2–3 are repeated untill the RF algorithm converges to a level that satisfies the user; that is, enough of truly relevant patterns have been retrieved by the search engine. Obviously, different techniques have been suggested and used for all three steps in the loop.

For Step 1, one can initialize the system either randomly or, preferably, by retrieving an initial set of "similar" patterns based on a similarity metric, such as the l_p metric. For Step 3, a popular choice is to employ a binary classifier that is trained to classify the marked patterns, in Step 2, either to the "relevant" or the "irrelevant" class, according to their label as judged by the user. Support vector machines have enjoyed a high popularity among the researchers, although other classifiers can also be used, see, [Druc 01, Cruc 04, Liu 07].

Interestingly enough, Step 2 is of critical importance. A *selection strategy* is first adopted concerning the *type* of the patterns that the system returns and on which the user must apply the query concept and label these patterns accordingly. Obviously, the patterns that the search engine returns in each round depend on the current knowledge of the "learner" in Step 3. In each iteration step, the system performs a ranking of the retrieved patterns, according to a confidence measure associated with the classifier' s decisions. For example, in the case an SVM has been adopted, this confidence measure can be the distance of the feature vector,

representing the pattern, from the decision hyperplane in the RKHS feature space, which is proportional to the value of $|g(\boldsymbol{x})|$ in Eq. (4.72) (Section 4.18). In [Druc 01], the remotest instance to the positive (relevant) side is ranked at the top of the list, and the remotest one to the negative side (irrelevant) is ranked at the bottom. The user selects a number of patterns (say, 10 to 20) among the top ranked in the list. That is, selection is done among patterns that have been classified (by the current classifier) as relevant and with high confidence. Obviously, if the system has not yet converged, some of the returned patterns will not satisfy the user and will be judged as being irrelevant. This strategy seems to be the most popular one. The strong point of such a selection procedure is that the user gets a few good relevant patterns quite early in the iteration process. On the other hand, such a philosophy turns out to lead to a relatively slow convergence of the method.

Another point of view has been adopted in the strategy proposed in [Tong 01]. This is inspired by the notion of *active learning* used in pattern recognition (see, [Lewi 94, Scho 00]). Active learning is an approach that trains the classifier by using a *subset* of the data, that are considered the most *informative* ones. Hence, one can achieve better performance with less training data. The most informative data points are taken to be the most *uncertain* instances. Thus, in such a RF setting, the user is asked to label a number of pool patterns that lie closest to the classifier' s boundary decision. In [Tong 01] a strong theoretical justification is also provided for such a scenario. In other words, such a selection criterion forces the system to elicit from the user a crucial part of information: what makes distinct the "relevant" from the "irrelevant." This is because, once the user labels as relevant or irrelevant such "uncertain" patterns, a significant part of uncertainty is removed from the system. The advantage of this selection criterion is that it speeds up the convergence of the RF scheme.

Figure 8.16 compares the convergence performance of an RF system for the two previous strategies and for two different users; a "lazy" user, who only marks up to two relevant and two irrelevant patterns (if a single relevant pattern is returned by the system, the user marks only one) and a "patient" user who marks all the relevant and all the irrelevant patterns, among the patterns that the engine returns at each iteration step. The horizontal axis corresponds to the number of iteration steps and the vertical one to the *precision*, denoted as Pr. As precision, we define the ratio of the relevant patterns to the total number of returned (at each iteration round) patterns. In all cases, the curves start from the same point. This is the precision value corresponding to the initialization step, where a simple similarity measure was used. Also, all curves tend to $Pr = 1$; that is, as the learning process advances, more and more of the returned patterns are judged by the user as being relevant. For both users, the curve corresponding to the active learning strategy tends to $Pr = 1$ faster. As it is natural, the system tends to $Pr = 1$ much faster for the cases of the patient user, since, at each iteration step, more patterns are available for training the classifier in Step 3. The curves in the figure have been obtained using the Wang image database [Wang 01] and an SVM classifier. More details about the features and

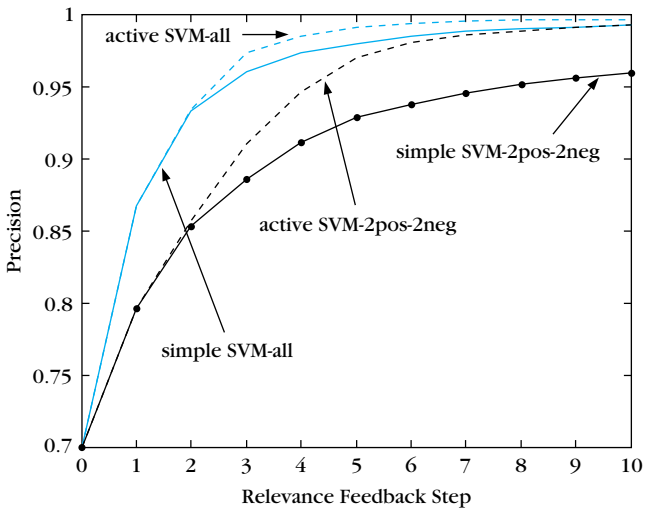

FIGURE 8.16

Dotted lines correspond to the active learning scenario. Learning curves for the "lazy"user, who, in each iteration round, marks up to two positive (relevant) and two negative (irrelevant) images, converge slower compared to those associated with the user who labels all the returned images as either positive or negative. For both users, active learning leads to faster convergence. We have used the terms *active* and *simple* to annotate the curves corresponding to the two strategies.

the parameters used are available in the book's Web site, and the interested reader can also perform a set of related experiments.

For the experienced researcher, it will not come as a surprise to say that a third route to the selection strategy has been proposed by combining the previous two "extremes." In [Xu 03], a hybrid approach has been suggested consisting of the following steps:

1. An SVM is trained by the user' s initial query.
2. The system returns a set of M patterns consisting of the $K \leq M$ remotest ones, from the decision hyperplane in the RKHS and on the positive side, and the $M-K$ closest ones to it.
3. The user labels the patterns as relevant or irrelevant.
4. The SVM classifier is retrained using all the patterns labeled, so far, by the user. The algorithm is either terminated, if the user is satisfied, or it is redirected to Step 2.

In [Xu 03], it is claimed that such a scheme shares the advantages of both previous selection criteria. It speeds up convergence and at the same time presents to the user some truly relevant patterns early enough in the iteration process.

Although in this section we have focused on the use of a classifier in Step 3, other alternatives are also possible. For example, in the so-called *query point method* (QPM) the existence of an ideal query point in the feature space is assumed, which, if found, would provide the appropriate answer to the user's query. Each feature in the feature vector is weighted and the task of the learner is to adjust the weights so that to move the point in the feature space appropriately. Learning can be based either on the positive examples (e.g., [Scla 97]) or on both positive and negative ones (e.g., [Rui 98]). For a more detailed reference survey on the topic the interested reader may consult [Liu 07, Long 03, Cruc 04].

8.6 PROBLEMS

8.1 Find the Edit distance between the word "poem" and its misspelled version "poten." Draw the optimal path.

8.2 Derive the slopes for the Sakoe–Chiba constraints and draw paths that achieve the maximum expansion/compression rates as well as paths corresponding to intermediate rates.

8.3 Develop a computer program for dynamic time warping, for a complete optimum path, and for the Itakura constraints. Verify the algorithm, using speech segments available from the book's web site.

8.4 Let the seven Hu moments of the reference $M \times N$ image block be $\phi_i, i = 1, 2, \ldots, 7$. Also, denote by $\psi_i(m, n), i = 1, 2, \ldots, 7$, the respective moments resulting from the test subimage located at (m, n). Explain why

$$\mathcal{M}(m,n) = \frac{\sum_{i=1}^{7} \phi_i \psi_i(m,n)}{\left(\sum_{i=1}^{7} \phi_i^2 \sum_{i=1}^{7} \psi_i^2(m,n)\right)^{1/2}}$$

is a reasonable measure of similarity.

8.5 Show that the Mellin transform $M(u, v)$ of a function $f(x, y)$, defined as

$$M(u,v) = \iint f(x,y) x^{-ju-1} y^{-jv-1}\, dx\, dy$$

is scale invariant.

8.6 For the motion compensation step in a video coding application, the image frame of size $I \times J$ is divided into subblocks of size $M \times N$. For each subblock the search for its corresponding one, in the current frame, is restricted in an area of size $[-p, p] \times [-p, p]$. Find the required number of operations per second for the computation of the maximum cross-correlation for a full search, a two-dimensional and a one-dimensional (independent i,j) logarithmic search, and a hierarchical search of three levels. Typical values for broadcast TV are $M = N = 16, I = 720, J = 480$, and $p = 15$. The number of frames transmitted per second is $f = 30$.

MATLAB PROGRAMS AND EXERCISES

Computer Programs

8.1 *Edit distance.* Write a MATLAB function named *edit_distance* that computes the Edit distance between two strings. This function takes as inputs (a) the reference string *ref_str* and (b) the test string *test_str*. It returns (a) the matching cost *edit_cost* and (b) the node predecessors matrix *pred*. Each element of the last matrix stores the coordinates of the predecessor of a node as a complex number, the real part of which stands for the row index and the imaginary part for the column index.

Solution

In the following implementation it is assumed that the reference pattern is associated with the horizontal axis.

```
function [edit_cost,pred]=edit_distance(ref_str,test_str)
  I = length(ref_str);
  J = length(test_str);
  D = zeros(J,I);
  %Initialization
  D(1,1) = ~(ref_str(1) == test_str(1));
  pred(1,1) = 0;
  for j = 2:J
    D(j,1) = D(j-1,1)+1;
    pred(j,1) = (j-1) + sqrt(-1)*1;
  end
  for i = 2:I
    D(1,i) = D(1,i-1)+1;
    pred(1,i) = 1+ sqrt(-1)*(i-1);
  end
  %Main Loop
  for i = 2:I
    for j = 2:J
      if(ref_str(i) == test_str(j))
        d(j,i) = 0;
      else
        d(j,i) = 1;
      end
      c1 = D(j-1,i-1)+d(j,i); c2 = D(j,i-1)+1; c3 = D(j-1,i)+1;
      [D(j,i),ind] = min([c1 c2 c3]);
      if(ind == 1)
        pred(j,i) = (j-1)+sqrt(-1)*(i-1);
      elseif(ind == 2)
```

```
        pred(j,i) = j+sqrt(-1)*(i-1);
      else
        pred(j,i) = (j-1)+sqrt(-1)*i;
      end
      end
   end
  edit_cost = D(J,I);
```

8.2 *Backtracking.* Write a MATLAB function named *back_tracking*, which takes as inputs (a) a node predecessors matrix *pred*, (b) the coordinates k and l of the node from which the backtracking will start. It returns the best path on the cost grid, *best_path*, and also plots the best path. It is assumed that backtracking always terminates when a node whose predecessor is $(0, 0)$ is reached.

Solution

```
function best_path=back_tracking(pred,k,l)
  Node = k+sqrt(-1)*l;
  best_path = [Node];
  while (pred(real(Node),imag(Node)) = 0)
    Node = pred(real(Node),imag(Node));
    best_path = [Node;best_path];
  end
  %Plot the best path
  [I,J] = size(pred);
  clf;
  hold
  for j = 1:J
    for i = 1:I
      plot(j,i,'r.')
    end
  end
  plot(imag(best_path),real(best_path),'g')
  axis off
```

8.3 *Dynamic time warping with Sakoe–Chiba local path constraints.* Write a MATLAB function named *Dtw_Sakoe* that implements a dynamic time-warping scheme where the Sakoe–Chiba local path constraints are adopted. More specifically, the function takes as input (a) a row vector *ref* that corresponds to the reference sequence and (b) a row vector *test* that corresponds to the test sequence. It returns (a) the time-warping matching cost, *matching_cost* and (b) the best path *best_ path*. It is assumed that (i) the cost assigned to each node in the grid upon initialization is equal to the Euclidean distance of the respective pattern elements and (ii) the cost of a transition

depends only on the cost that has been assigned to the node at the end of the transition.

Solution

In the following implementation, function *back_tracking* introduced before is utilized to determine the best path.

```
function [matching_cost,best_path]=Dtw_Sakoe(ref,test)
  I = length(ref);
  J = length(test);
  for i = 1:I
    for j = 1:J
      %Euclidean distance
      node_cost(i,j) = sqrt(sum((ref(:,i)-test(:,j)).^2));
    end
  end
  %Initialization
  D(1,1) = node_cost(1,1);
  pred(1,1) = 0;
  for i = 2:I
    D(i,1) = D(i-1,1)+node_cost(i,1);
    pred(i,1) = i-1 + sqrt(-1)*1;
  end
  for j = 2:J
    D(1,j) = D(1,j-1)+node_cost(1,j);
    pred(1,j) = 1 + sqrt(-1)*(j-1);
  end
  %Main Loop
  for i = 2:I
    for j = 2:J
      [D(i,j),ind] = min([D(i-1,j-1) D(i-1,j) ...
          D(i,j-1)]+node_cost(i,j));
      if (ind == 1)
        pred(i,j) = (i-1)+sqrt(-1)*(j-1);
      elseif (ind == 2)
        pred(i,j) = (i-1)+sqrt(-1)*(j);
      else
        pred(i,j) = (i)+sqrt(-1)*(j-1);
      end
    end %for j
  end %for i
  %End of Main Loop
  matching_cost = D(I,J);
  best_path = back_tracking(pred,I,J);
```

Computer Experiments

8.1 **a.** Compute the Edit distance between the following pairs of strings: (i) (*beauty*, *beaty*), (ii) (*beauty*, *biauty*), (iii) (*beauty*, *betty*), using the first element of each pair as the reference string.

b. Plot the respective matching paths using the function *back_tracking*.

8.2 Use the *Dtw_Sakoe* function to compute the time-warping cost between the sequences $\{1,\ 2,\ 3\}$ and $\{1,\ 1,\ 2,\ 2,\ 2,\ 3,\ 3,\ 3\}$, using the former as reference sequence. Comment on the results.

8.3 Let $r1 = [1,\ 0]^T$, $r2 = [0,\ 1]^T$ and $ref = [r1,\ r2]$. Generate a sequence of 10 two-dimensional vectors with the following MATLAB command $test = [1 + rand(1,4)/2 \quad rand(1,6)/3;\quad rand(1,4)/2 \quad 1 + rand(1,6)/3]$. Use the *Dtw_Sakoe* function to compute the time-warping cost and the respective best path between *ref* and *test*, taking the former as the reference sequence. Comment on the results.

REFERENCES

[Alkh 01] Ghavari-Alkhansavi M. "A fast globally optimal algorithm for template matching using low resolution pruning," *IEEE Transactions on Image Processing*, Vol. 10(4), pp. 526–533, 2001.

[Amit 91] Amit Y., Grenander U., Piccioni M. "Structural image restoration through deformable template," *J. Amer. Statist. Association*, Vol. 86(414), pp. 376–387, 1991.

[Bell 57] Bellman R.E. *Dynamic Programming*, Princeton University Press, 1957.

[Bhas 95] Bhaskaran V., Konstantinides K. *Image and Video Compression Standards*, Kluwer Academic Publishers, 1995.

[Cheu 02] Cheung K.-W., Yeung D.-Y., Chin R.T. "On deformable models for visual pattern recognition," *Pattern Recognition*, Vol. 35, pp. 1507–1526, 2002.

[Cruc 04] Crucianu M., Ferecatu M., Boujemaa N. "Relevance feedback for image retrieval: a short survey," in *Audiovisual Content-based Retrieval, Information Universal Access and Interaction, Including Datamodels and Languages*, Report of the DELOS2 European Network of Excellence, FP6, 2004.

[Dame 64] Damerau F.J. "A technique for computer detection and correction of spelling errors," *Commun. ACM*, Vol. 7(3), pp. 171–176, 1964.

[Davi 80] Davis S.B., Mermelstein P. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 28(4), pp. 357–366, 1980.

[Dell 93] Deller J., Proakis J., Hansen J.H.L. *Discrete-Time Processing of Speech Signals*, Macmillan, 1993.

[Desh 99] Deshmukh N., Ganapathirajn A., Picone J. "Hierarchical search for large vocabulary conversational speech recognition," *IEEE Signal Processing Magazine*, Vol. 16(5), pp. 84–107, 1999.

[Druc 01] Drucker H., Shahraray B., Gibbon D. "Relevance feedback using support vector machines," *Proceedings of th 18th International Conference on Machine Learning*, pp. 122–129, 2001.

[Durb 97] Durbin K., Eddy S., Krogh A., Mitchison G. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press, Cambridge, MA, 1997.

[Fisc 73] Fischler M., Elschlager R. "The representation and matching of pictorial structures," *IEEE Transations on Computers*, Vol. 22(1), pp. 67–92, 1973.

[Gray 76] Gray A.H., Markel J.D. "Distance measures for speech processing," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 24(5), pp. 380–391, 1976.

[Gray 80] Gray R.M., Buzo A., Gray A.H., Matsuyama Y. "Distortion measures for speech processing," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 28(4), pp. 367–376, 1980.

[Hall 79] Hall E. *Computer Image Processing and Recognition*, Academic Press, 1979.

[Itak 75] Itakura F. "Minimum prediction residual principle applied to speech recognition," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 23(2), pp. 67–72, 1975.

[Jain 98] Jain A.K., Zhong Y., Dubuisson-Jolly M.P. "Deformable template models: A review," *Signal Processing*, Vol. 71, pp. 109–129, 1998.

[Jain 96] Jain A.K., Zhong Y., Lakshmanan S. "Object matching using deformable templates," *IEEE Transactions on Pattern Analysis and Machine Itelligence*, Vol. 18(3), pp. 267–277, 1996.

[Koch 89] Koch M.W., Kashyap R.L. "Matching polygon fragments," *Pattern Recognition Letters*, Vol. 10, pp. 297–308, 1989.

[Leven 66] Levenshtein V.I. "Binary codes capable of correcting deletions, insertions and reversals," *Soviet phys. Dokl.*, Vol. 10(8), pp. 707–710, 1966.

[Lewi 94] Lewis D., Gale W. "A sequential algorithm for training text classifiers," *Proceedings of the 11th International Conference on Machine Learning*, pp. 148–156, Morgan Kaufmann, 1994.

[Liu 07] Liu Y., Zhang D., Lu G., Ma W.-Y. "A survey of content-based image retrieval with high-level semantics," *Pattern Recognition*, Vol. 40, pp. 262–282, 2007.

[Long 03] Long F., Zang H.J., Feng D.D. "Fundamentals of content-based image retrieval," in *Multimedia Information Retrieval and Management* (Feng D. ed.), Springer, Berlin, 2003.

[Marz 93] Marzal A., Vidal E. "Computation of normalized edit distance and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(9), 1993.

[McIn 96] McInerney T., Terzopoulos D. "Deformable models in medical image analysis: A survey," *Med. Image Anal.*, Vol. 1(2), pp. 91–108, 1996.

[Mei 04] Mei J. "Markov edit distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6(3), pp. 311–320, 2004.

[Myer 80] Myers C.S., Rabiner L.R., Rosenberg A.E. "Performance tradeoffs in dynamic time warping algorithms for isolated word recognition," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 28(6), pp. 622–635, 1980.

[Neg 99] Neg H., Ortmanns S. "Dynamic programming search for continuous speech recognition," *IEEE Signal Processing Magazine*, Vol. 16(5), pp. 64–83, 1999.

[Ocud 76] Ocuda T., Tanaka E., Kasai T. "A method for correction of garbled words based on the Levenstein metric," *IEEE Transactions on Computers*, pp. 172–177, 1976.

[Pikr 03] Pikrakis A., Theodoridis S., Kamarotos D. "Recognition of isolated musical patterns using context dependent dynamic time warping," *IEEE Transactions on Speech and Audio Processing*, Vol 11(3), pp. 175–183, 2003.

[Rabi 93] Rabiner L., Juang B.H. *Fundamentals of Speech Recognition*, Prentice Hall, 1993.

[Ravi 95] Ravichandran G., Trivedi M.M. "Circular-Mellin features for texture segmentation," *IEEE Transactions on Image Processing*, Vol. 2(12), pp. 1629–1641, 1995.

[Rui 98] Rui Y., Huang T.S., Ortega M., Mehrotra S. "Relevance feedback: power tool in interactive content-based image retrieval," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 8(5), pp. 644–655, 1998.

[Sako 78] Sakoe H., Chiba S. "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 26(2), pp. 43–49, 1978.

[Scha 89] Schalkoff R. *Digital Image Processing and Computer Vision*, John Wiley & Sons, 1989.

[Scho 00] Schohn G., Cohn D. "Less is more: Active learning with support vector machines," *Proceedings of the 17th International Conference on Machine Learning*, pp. 839–846, Morgan Kaufmann, 2000.

[Scla 97] Sclaroff S., Taycher L. Cascia M "Imagerover: A content-based image browser for the world wide web," *Proceedings of the 1997 Workshop on Content-Based Access of Image and Video Libraries (CBAIVL'97)*, pp. 2–9, IEEE Computer Society, 1997.

[Seni 96] Seni G., Kripasundar V., Srihari R. "Generalizing Edit distance to incorporate domain information: Handwritten text recognition as a case study," *Pattern Recognition*, Vol. 29(3), pp. 405–414, 1996.

[Silv 90] Silverman H., Morgan D.P. "The application of the dynamic programming to connected speech recognition," *IEEE Signal Processing Magazine*, Vol. 7(3), pp. 7–25, 1990.

[Tong 01] Tong S., Chang E. "Support vector machine active learning for image retrieval," *Proceedings of the 9th ACM International Conference on Multimedia*, pp. 107–118, ACM Press, 2001.

[Tsay 93] Tsay Y.T., Tsai W.H. "Attributed string matching split and merge for on-line Chinese character recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(2), pp. 180–185, 1993.

[Ueno 97] Uenohara M., Kanade T. "Use of the Fourier and Karhunen–Loève decomposition for fast pattern matching with a large set of templates," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19(8), pp. 891–899, 1997.

[Wang 01] Wang J.Z., Li J., Wiederhold G. "SIMPLIcity: Semantics-sensitive Integrated Matching for Picture LIbraries," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 23, no.9, pp. 947–963, 2001.

[Wang 90] Wang Y.P., Pavlidis T. "Optimal correspondence of string subsequences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12(11), pp. 1080–1086, 1990.

[Widr 73] Widrow B. "The rubber mask technique, Parts I and II," *Pattern Recognition*, Vol. 5, pp. 175–211, 1973.

[Xu 03] Xu Z., Xu X., Yu K., Tresp V. "A hybrid relevance-feedback approach to text retrieval," *Proceedings of the 25th European Conference on Information Retrieval Research*, Lecture Notes in Computer Science, Vol. 2633, Springer Verlag, 2003.