Contents

List of Algorithms

Chapter 7

Object recognition

Not even the simplest machine vision tasks can be solved without the help of recognition. Pattern recognition is used for region and object classification, and basic methods of pattern recognition must be understood in order to study more complex machine vision processes.

Classification of objects or regions has been mentioned several times; recognition is then the last step of the bottom-up image processing approach. It is also often used in other control strategies for image understanding. Almost always when information about an object or region class is available, some pattern recognition method is used.

Consider a simple recognition problem. Two different parties take place at the same hotel at the same time – the first is a celebration of a successful basketball season, and the second a yearly meeting of jockeys. The doorman is giving directions to guests, asking which party they are to attend. After a while the doorman discovers that no questions are necessary and he directs the guests to the right places, noticing that instead of questions, he can just use the obvious physical features of basketball players and jockeys. Maybe he uses two features to make a decision, the weight and the height of the guests. All small and light men are directed to the jockey party, all tall and heavier guests are sent to the basketball party. Representing this example in terms of recognition theory, the early guests answered the doorman's question as to which party they are going to visit. This information, together with characteristic features of these guests, resulted in the ability of the doorman to classify them based only on their features. Plotting the guests' height and weight in a two-dimensional space (see Figure 7.1), it is clear that jockeys and basketball players form two easily separable classes and that this recognition task is extremely simple. Although real object recognition problems are often more difficult, and the classes do not differ so substantially, the main principles remain the same.

The theory of pattern recognition is thoroughly discussed in several references [Fu 82, Devijver and Kittler 82, Oja 83, Devijver and Kittler 86, Patrick and Fattu 86, Fukunaga 90, Dasarathy 91, Sethi and Jain 91, Schalkoff 92, Chen et al. 93, Pavel 93, Nigrin 93, Cherkassky et al. 94, Hlavac and Sara 95], and here only a brief introduction will be given. In addition, we will introduce some other related techniques: graph matching, neural nets, genetic algorithms, simulated annealing, and fuzzy logic.

No recognition is possible without knowledge. Decisions about classes or groups into which recognized objects are classified are based on such knowledge – knowledge about objects and their classes gives the necessary information for object classification. Both specific knowledge

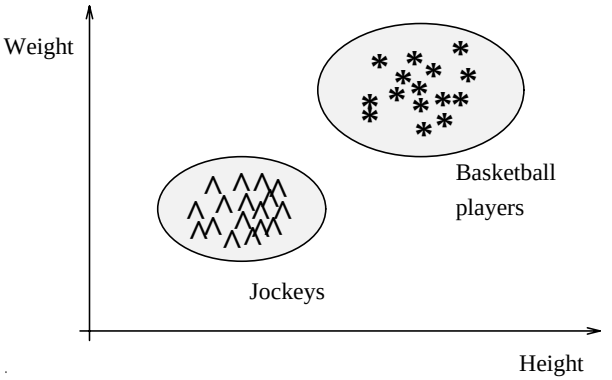

Figure 7.1: *Recognition of basketball players and jockeys; features are weight and height.*

about the objects being processed and hierarchically higher and more general knowledge about object classes is required. First, common knowledge representation techniques will be introduced because the concept of knowledge representation in suitable form for a computer may not be straightforward.

7.1 Knowledge representation

Knowledge as well as knowledge representation problems are studied in artificial intelligence (AI), and computer vision takes advantage of these results. Use of AI methods is very common in higher processing levels and a study of AI is necessary for a full appreciation of computer vision and image understanding. Here we present a short outline of common techniques as they are used in AI, and an overview of some basic knowledge representations. More detailed coverage of knowledge representation can be found in [Michalski et al. 83, Winston 84, Simons 84, Devijver and Kittler 86, Wechsler 90, Reichgelt 91, Lakemeyer and Nebel 94, Masuch and Polos 94].

Experience shows that a good knowledge representation design is the most important part of solving the understanding problem. Moreover, a small number of relatively simple control strategies is often sufficient for AI systems to show complex behavior, assuming an appropriately complex knowledge base is available. In other words, a high degree of control sophistication is not required for intelligent behavior, but a rich, well structured representation of a large set of a priori data and hypotheses is needed [Schutzer 87].

Other terms of which regular use will be made are **syntax** and **semantics** [Winston 84].

- The **syntax** of a representation specifies the symbols that may be used and the ways that they may be arranged.
- The **semantics** of a representation specifies how meaning is embodied in the symbols and the symbol arrangement allowed by the syntax.
- A **representation** is a set of syntactic and semantic conventions that make it possible to describe things.

The main knowledge representation techniques used in AI are: Formal grammars and languages, predicate logic, production rules, semantic nets and frames. Even if features and

descriptions are not usually considered knowledge representations, they are added for practical reasons; these low-level forms of knowledge representation will be mentioned many times throughout the coming sections.

Note that knowledge representation data structures are mostly extensions of conventional data structures like lists, trees, graphs, tables, hierarchies, sets, rings, nets and matrices.

Descriptions, features

Descriptions and features cannot be considered pure knowledge representations. Nevertheless, they can be used for representing knowledge as a part of a more complex representation structure.

Descriptions usually represent some scalar properties of objects, and are called **features**. Typically, a single description is insufficient for object representation, therefore the descriptions are combined into **feature vectors**. Numerical feature vectors are inputs for statistical pattern recognition techniques (see Section 7.2).

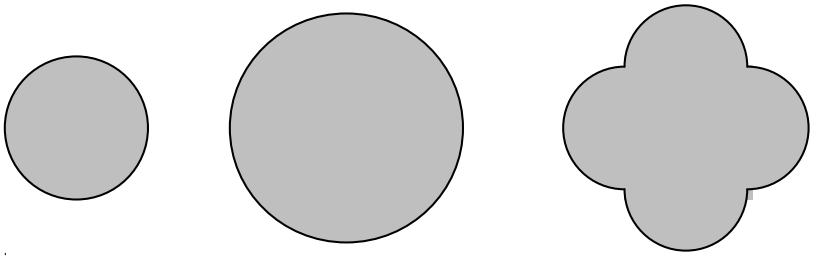

Figure 7.2: *Feature description of simple objects.*

A simple example of feature description of objects is shown in Figure 7.2. The *size* feature can be used to represent an area property and the *compactness* feature describes circularity (see Section 6.3.1). Then the feature vector $\mathbf{x} = (size, compactness)$ can be used for object classification into the following classes of objects: small, large, circular, noncircular, small and circular, small and noncircular, etc. assuming information about what is considered small/large and circular/noncircular is available.

Grammars, languages

If an object's structure needs to be described, feature description is not appropriate. A structural description is formed from existing primitives and the relations between them.

Primitives are represented by information about their types. The simplest form of structure representations are chains, trees and general graphs. Structural description of chromosomes using border segments as primitives is a classic example of structural object description [Fu 82] (see Figure 6.14), where borders are represented by a chain of symbols, the symbols representing specific types of border primitives. Hierarchical structures can be represented by trees – the concavity tree of Figure 6.30 serves as an example. A more general graph representation is used in Chapter 14 where a graph grammar (Figure 14.6) is used for texture description. Many examples of syntactic object description may be found in [Fu 74, Fu 77, Fu 80, Fu 82].

One object can be described by a chain, a tree, a graph, etc. of symbols. Nevertheless, the whole class of objects cannot be described by a single chain, a single tree, etc., but a class of structurally described objects can be represented by **grammars** and **languages**. Grammars and languages (similar to natural languages) provide rules defining how the chains, trees or graphs can be constructed from a set of symbols (primitives). A more specific description of grammars and languages is given in Section 7.4.

Predicate logic

Predicate logic plays a very important role in knowledge representation – it introduces a mathematical formalism to derive new knowledge from old knowledge by applying mathematical deduction. Predicate logic works with combinations of logic variables, quantifiers $(\exists, \forall)$, and logic operators (*and*, *or*, *not*, *implies*, *equivalent*). The logic variables are binary (*true*, *false*). The idea of proof and rules of inference such as **modus ponens** and **resolution** are the main building blocks of predicate logic [Pospesel 76].

Predicate logic forms the essence of the programming language PROLOG that is widely used if objects are described by logic variables. Requirements of 'pure truth' represent the main weakness of predicate logic in knowledge representation since it does not allow work with uncertain or incomplete information. Predicate logic incorporates logic conditions and constraints into knowledge processing (see Section 8.5), [Hayes 77, Kowalski 79, Clocksin and Mellish 81].

Production rules

Production rules represent a wide variety of knowledge representations that are based on **condition-action** pairs. The essential model of behavior of a system based on production rules (a production system) can be described as follows:

if condition X holds *then* action Y is appropriate

Information about what action is appropriate at what time represents knowledge. The procedural character of knowledge represented by production rules is another important property – not all the information about objects must be listed as an object property. Consider a simple knowledge base where the following knowledge is present

$$if \quad \text{ball} \quad then \quad \text{circular} \tag{7.1}$$

Let the knowledge base also include the statements

$$\begin{array}{lll} \text{object A} & is\_a & \text{ball} \\ \text{object B} & is\_a & \text{ball} \\ \text{object C} & is\_a & \text{shoe} \\ & \text{etc.} & \end{array} \tag{7.2}$$

To answer the question *how many objects are circular?*, if enumerative knowledge representation is used, the knowledge must be listed as

object A *is_a* (ball, circular)

object B *is_a* (ball, circular)

etc.

If procedural knowledge is used, the knowledge base (7.2) together with the knowledge (7.1) gives the same information in a significantly more efficient manner.

Both production rule knowledge representation and production systems appear frequently in computer vision and image understanding problems. Furthermore, production systems together with a mechanism for handling uncertainty information, form a basis of expert systems.

Fuzzy logic

Fuzzy logic has been developed [Zadeh 65, Zimmermann et al. 84] to overcome the obvious limitations of numerical or crisp representation of information. Consider the use of knowledge represented by equation (7.1) for recognition of balls; using the production rule, the knowledge about balls may be represented as

$$if \quad \text{circular} \quad then \quad \text{ball} \tag{7.3}$$

If the object in a two-dimensional image is considered circular then it may represent a ball. Our experience with balls, however, says that they are usually close to, but not perfectly, circular. Thus, it is necessary to define some circularity threshold so that all *reasonably* circular objects from our set of objects are labeled as balls. Here is the fundamental problem of crisp descriptions; how circular must an object be to be considered circular?

If humans represent such knowledge, the rule for ball circularity may look like

$$if \text{ circularity is HIGH} \quad then \quad \text{object is a ball with HIGH confidence} \tag{7.4}$$

Clearly, high circularity is a preferred property of balls. Such knowledge representation is very close to common sense representation of knowledge, with no need for exact specification of the circularity/non-circularity threshold. **Fuzzy rules** are of the form

$$if \quad X \text{ is } A \quad then \quad Y \text{ is } B \tag{7.5}$$

where X and Y represent some properties and A and B are **linguistic variables**. Fuzzy logic can be used to solve object recognition and other decision making tasks, among others; this is further discussed in Section 7.7.

Semantic nets

Semantic nets are a special variation of relational data structures (see Chapter 3). The semantics distinguish them from general nets – semantic nets consist of objects, their description, and a description of relations between objects (often just relations between neighbors). Logical forms of knowledge can be included in semantic nets, and predicate logic can be used to represent and/or evaluate the local information and local knowledge. Semantic nets can also represent common sense knowledge that is often imprecise and needs to be treated in a probabilistic way. Semantic nets have a hierarchical structure; complex representations

consist of less complex representations, which can in turn be divided into simpler ones, etc. Relations between partial representations are described at all appropriate hierarchical levels.

Evaluated graphs are used as a semantic net data structure; nodes represent objects and arcs represent relations between objects. The following definition of a human face is an example of a simple semantic net:

- A *face* is a circular part of the human body that consists of two eyes, one nose, and one mouth.
- One eye is positioned left of the other eye.
- The nose is between and below the eyes.
- The mouth is below the nose.
- An eye is approximately circular.
- The nose is vertically elongated.
- The mouth is horizontally elongated.

The semantic net representing this knowledge is shown in Figure 7.3.

It is clear that the descriptive structures found in real images match the knowledge represented by a semantic net with varying degrees of closeness. The question of whether the described structure is similar to that represented by the semantic net is discussed in Section 7.5 and in Chapter 8.

A detailed discussion of semantic nets related to image information can be found in [Niemann 90], and more general properties of semantic nets are described in [Michalski et al. 83, Sharples et al. 89].

Frames, scripts

Frames provide a very general method for knowledge representation which may contain all the knowledge representation principles discussed so far. They are sometimes called **scripts** because of their similarity to film scripts. Frames are suitable for representing common sense knowledge under specific circumstances. Consider a frame called *plane_start*; this frame may consist of the following sequence of actions:

1. Start the engines
2. Taxi to the runway
3. Increase RPMs of engines to maximum
4. Travel along runway increasing speed
5. Fly

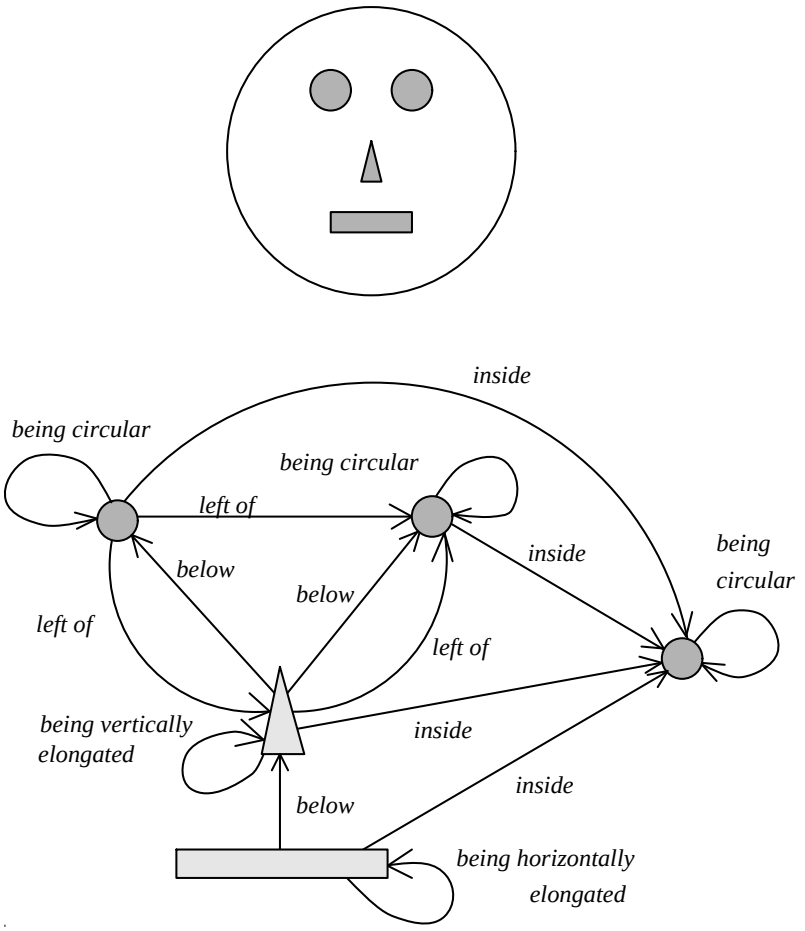

Figure 7.3: *Semantic nets: A human face model and its net.*

Assuming this frame represents knowledge of how planes *usually* start, the situation of a plane standing on a runway with engines running causes the prediction that the plane will start in a short time. The frame can be used as a substitute for missing information which may be extremely important in vision-related problems.

Assuming that one part of the runway is not visible from the observation point, using the *plane_start* frame, a computer vision system can overcome the lack of continuous information between the plane moving at the beginning of the runway and flying when it next appears. If it is a passenger plane, the frame may have additional items like *time of departure, time of arrival, departure city, arrival city, airline, flight number, etc.* because in a majority of cases it makes sense to be interested in this information if we identify a passenger plane.

From a formal point of view, a frame is represented by a general semantic net accompanied by a list of relevant variables, concepts, and concatenation of situations. No standard form of frame exists. Frames represent a tool for organizing knowledge in prototypical objects, and for description of mutual influences of objects using stereotypes of behavior in specific situations. Examples of frames can be found elsewhere [Michalski et al. 83, Winston 84, Schutzer 87, Sharples et al. 89]. Frames are considered high-level knowledge representations.

7.2 Statistical pattern recognition

An object is a physical unit, in image analysis and computer vision usually represented by a region in a segmented image. The set of objects can be divided into disjoint subsets, that, from the classification point of view, have some common features and are called **classes**. The definition of how the objects are divided into classes is ambiguous and depends on the classification goal.

Object recognition is based on assigning classes to objects and the device that does these assignments is called the **classifier**. The number of classes is usually known beforehand, and typically can be derived from the problem specification. Nevertheless, there are approaches in which the number of classes may not be known (see Section 7.2.4).

The classifier (similarly to a human) does not decide about the class from the object itself – rather, sensed object properties are used to serve this purpose. For example, to distinguish steel from sandstone, we do not have to determine their molecular structures, although this would describe these materials well. Properties like texture, specific weight, hardness, etc. are used instead. This sensed object is called the **pattern**, and the classifier does not actually recognize objects, but recognizes their patterns. Object recognition and pattern recognition are considered synonymous.

The main pattern recognition steps are shown in Figure 7.4. The block 'Construction of formal description' is based on the experience and intuition of the designer. A set of elementary properties is chosen which describe some characteristics of the object; these properties are measured in an appropriate way, and form the description pattern of the object. These properties can be either quantitative or qualitative in character and their form can vary (numerical vectors, chains, etc.). The theory of recognition deals with the problem of designing the classifier for the specific (chosen) set of elementary object descriptions.

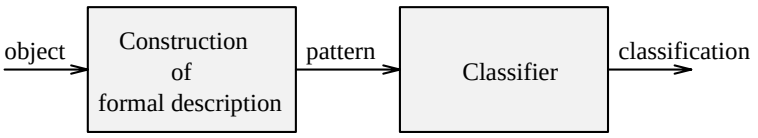

Figure 7.4: *Main pattern recognition steps.*

Statistical object description uses elementary numerical descriptions called **features**, $x_1, x_2, \ldots, x_n$; in image analysis, the features result from object description as discussed in Chapter 6. The pattern (also referred to as pattern vector, or feature vector) $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ that describes an object is a vector of elementary descriptions, and the set of all possible patterns forms the **pattern space** X (also called **feature space**). If the elementary descriptions were appropriately chosen, similarity of objects in each class results in the proximity of their patterns in pattern space. The classes form clusters in the feature space, which can be separated by a discrimination curve (or hypersurface in a multi-dimensional feature space) – see Figure 7.5.

If a discrimination hypersurface exists which separates the feature space such that only objects from one class are in each separated region, the problem is called a recognition task with **separable classes**. If the discrimination hypersurfaces are hyperplanes, it is called a **linearly separable** task. If the task has separable classes, each pattern will represent only

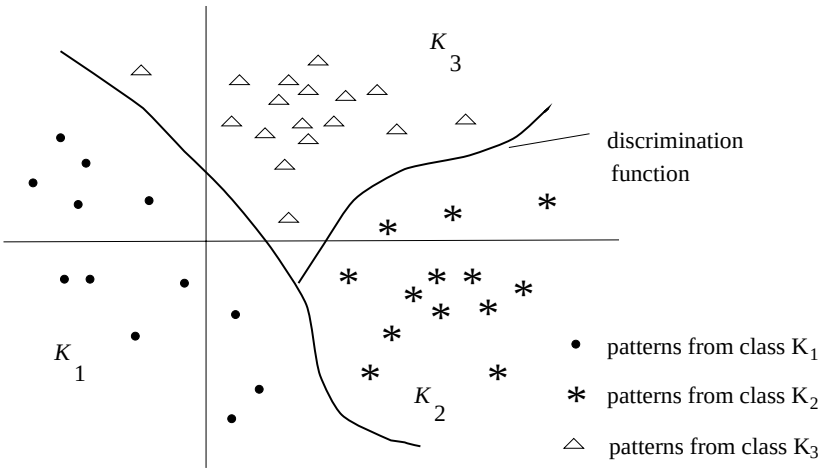

Figure 7.5: *General discrimination functions.*

objects from one class. Intuitively, we may expect that separable classes can be recognized without errors.

The majority of object recognition problems do not have separable classes, in which case the locations of the discrimination hypersurfaces in the feature space can never separate the classes correctly and some objects will always be misclassified.

7.2.1 Classification principles

A statistical classifier is a device with n inputs and 1 output. Each input is used to enter the information about one of n features $x_1, x_2, \ldots, x_n$ that are measured from an object to be classified. An R-class classifier will generate one of R symbols $\omega_1, \omega_2, \ldots, \omega_R$ as an output, and the user interprets this output as a decision about the class of the processed object. The generated symbols ω_r are the **class identifiers**.

The function $d(\mathbf{x}) = \omega_r$ describes relations between the classifier inputs and the output; this function is called the **decision rule**. The decision rule divides the feature space into R disjoint subsets K_r, $r = 1, \ldots, R$ each of which includes all the feature representation vectors $\mathbf{x}'$ of objects for which $d(\mathbf{x}') = \omega_r$. The borders between subsets K_r, $r = 1, \ldots, R$ form the discrimination hypersurfaces mentioned earlier. The determination of discrimination hypersurfaces (or definition of the decision rule) is the goal of classifier design.

The discrimination hypersurfaces can be defined by R scalar functions $g_1(\mathbf{x})$, $g_2(\mathbf{x})$, ..., $g_R(\mathbf{x})$ called **discrimination functions**. The design of discrimination functions must satisfy the following formula for all $\mathbf{x} \in K_r$ and for any $s \in \{1, \ldots, R\}$, $s \neq r$

$$g_r(\mathbf{x}) \geq g_s(\mathbf{x}) \tag{7.6}$$

Therefore, the discrimination hypersurface between class regions K_r and K_s is defined by

$$g_r(\mathbf{x}) - g_s(\mathbf{x}) = 0 \tag{7.7}$$

The decision rule results from this definition. The object pattern $\mathbf{x}$ will be classified into the

class whose discrimination function gives a maximum of all the discrimination functions:

$$d(\mathbf{x}) = \omega_r \iff g_r(\mathbf{x}) = \max_{s=1,\ldots,R} g_s(\mathbf{x}) \tag{7.8}$$

Linear discrimination functions are the simplest and are widely used. Their general form is

$$g_r(\mathbf{x}) = q_{r0} + q_{r1}x_1 + \ldots + q_{rn}x_n \tag{7.9}$$

for all $r = 1, \ldots, R$. If all the discrimination functions of the classifier are linear, it is called a **linear classifier**.

Another possibility is to construct classifiers based on the **minimum distance** principle. The resulting classifier is just a special case of classifiers with discrimination functions, however they have computational advantages and may easily be implemented on digital computers. Assume that R points are defined in the feature space, $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_R$ that represent **exemplars** (sample patterns) of classes $\omega_1, \omega_2, \ldots, \omega_R$. A minimum distance classifier classifies a pattern $\mathbf{x}$ into the class to whose exemplar it is closest.

$$d(\mathbf{x}) = \omega_r \iff |\mathbf{v}_r - \mathbf{x}| = \min_{s=1,\ldots,R} |\mathbf{v}_s - \mathbf{x}| \tag{7.10}$$

Each discrimination hyperplane is perpendicular to the line segment $\mathbf{v}_s\mathbf{v}_r$ and bisects it (Figure 7.6).

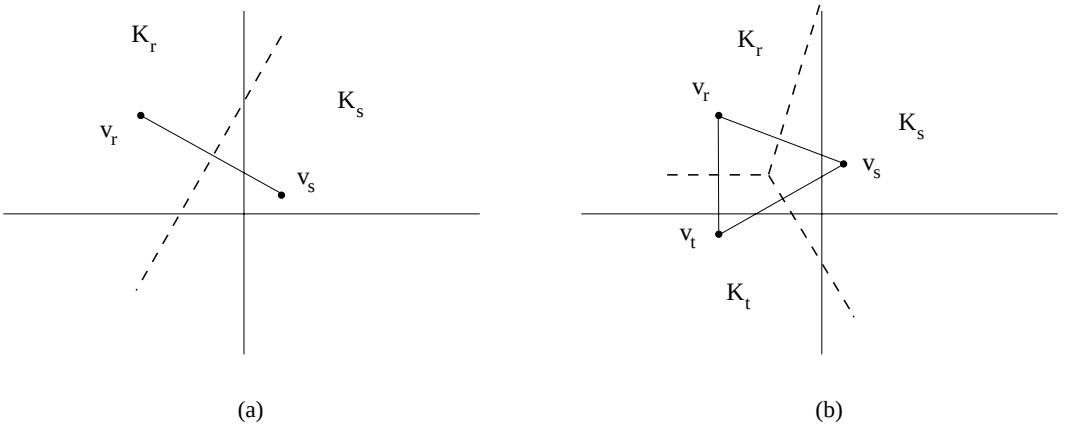

Figure 7.6: *Minimum distance discrimination functions. (a) 2-class problem, (b) 3-class problem.*

If each class is represented by just one exemplar, a linear classifier results. If more than one exemplar represents some class, the classifier results in piecewise linear discrimination hyperplanes. An algorithm for learning and classification using a minimum distance classifier can be found in Section 7.2.3, Algorithm 7.2.

Nonlinear classifiers usually transform the original feature space X^n into a new feature space X^m applying some appropriate nonlinear function $\mathbf{\Phi}$ where the superscripts n, m refer to the space dimensionality.

$$\mathbf{\Phi} = (\phi_1, \phi_2, \ldots, \phi_m) : X^n \to X^m \tag{7.11}$$

After the nonlinear transformation, a linear classifier is applied in the new feature space – the role of the function $\mathbf{\Phi}$ is to 'straighten' the nonlinear discrimination hypersurfaces of the original feature space into hyperplanes in the transformed feature space. This approach to feature space transformation is called a **Φ-classifier**.

The discrimination functions of a Φ-classifier are

$$g_r(\mathbf{x}) = q_{r0} + q_{r1}\phi_1(\mathbf{x}) + \ldots + q_{rm}\phi_m(\mathbf{x}) \tag{7.12}$$

where $r = 1, \ldots, R$. We may rewrite the formula in vector representation

$$g_r(\mathbf{x}) = \mathbf{q}_r \cdot \mathbf{\Phi}(\mathbf{x}) \tag{7.13}$$

where $\mathbf{q}_r$, $\mathbf{\Phi}(\mathbf{x})$ are vectors consisting of $q_{r0}, \ldots, q_{rm}$ and $\phi_0(\mathbf{x}), \ldots, \phi_m(\mathbf{x})$, respectively, $\phi_0(\mathbf{x}) \equiv 1$. Nonlinear classifiers are described in detail in [Sklansky 81, Devijver and Kittler 82].

7.2.2 Classifier setting

A classifier based on discrimination functions is a deterministic machine – one pattern $\mathbf{x}$ will always be classified into the same class. Note that the pattern $\mathbf{x}$ may represent objects from different classes, meaning that the classifier decision may be correct for some objects and incorrect for others. Therefore, setting of the optimal classifier should be probabilistic. Incorrect classifier decisions cause some losses to the user, and according to the definition of loss, different criteria for optimal classifier settings will be obtained. Discussing these optimality criteria from the mathematical point of view, criteria represent the value of the mean loss caused by classification.

Let the classifier be considered a universal machine that can be set to represent any decision rule from the rule set D. The set D may be ordered by a parameter vector $\mathbf{q}$ that refers to particular discrimination rules. The value of the mean loss $J(\mathbf{q})$ depends on the decision rule that is applied $\omega = d(\mathbf{x}, \mathbf{q})$. In comparison with the definition of decision rule used in the previous section, the parameter vector $\mathbf{q}$ has been added to represent the specific decision rule used by the classifier. The decision rule

$$\omega = d(\mathbf{x}, \mathbf{q}^*) \tag{7.14}$$

that gives the minimum mean loss $J(\mathbf{q})$ is called the optimum decision rule, and $\mathbf{q}^*$ is called the vector of optimal parameters

$$J(\mathbf{q}^*) = \min_{\mathbf{q}} J(\mathbf{q}), \qquad d(\mathbf{x}, \mathbf{q}) \in D \tag{7.15}$$

The **minimum error criterion** (Bayes criterion, maximum likelihood) uses loss functions of the form $\lambda(\omega_r|\omega_s)$, where $\lambda(.)$ is the number that describes quantitatively the loss incurred if a pattern $\mathbf{x}$ which should be classified into the class ω_s is incorrectly classified into the class ω_r

$$\omega_r = d(\mathbf{x}, \mathbf{q}) \tag{7.16}$$

The mean loss is

$$J(\mathbf{q}) = \int_X \sum_{s=1}^{R} \lambda(d(\mathbf{x}, \mathbf{q})|\omega_s) p(\mathbf{x}|\omega_s) P(\omega_s) \mathbf{dx} \tag{7.17}$$

where $P(\omega_s)$, $s = 1, \ldots, R$ are the a priori probabilities of classes, and $p(\mathbf{x}|\omega_s)$, $s = 1, \ldots, R$ are the conditional probability densities of objects $\mathbf{x}$ in the class ω_s.

A classifier that has been set according to the minimum loss optimality criterion is easy to construct using discrimination functions; usually, unit loss functions are considered

$$\begin{aligned} \lambda(\omega_r|\omega_s) &= 0 \text{ for } r = s \\ &= 1 \text{ for } r \neq s \end{aligned} \tag{7.18}$$

and the discrimination functions are

$$g_r(\mathbf{x}) = p(\mathbf{x}|\omega_r)P(\omega_r), \qquad r = 1, \ldots, R \tag{7.19}$$

where $g_r(\mathbf{x})$ corresponds (up to a multiplicative constant) to the value of the a posteriori probability $P(\omega_r|\mathbf{x})$.

This probability describes how often a pattern $\mathbf{x}$ is from the class ω_r. Clearly, the optimal decision is to classify a pattern $\mathbf{x}$ to a class ω_r if the a posteriori probability $P(\omega_r|\mathbf{x})$ is the highest of all possible a posteriori probabilities

$$P(\omega_r|\mathbf{x}) = \max_{s=1,\ldots,R} P(\omega_s|\mathbf{x}) \tag{7.20}$$

A posteriori probability may be computed from a priori probabilities using the Bayes formula

$$P(\omega_s|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_s)P(\omega_s)}{p(\mathbf{x})} \tag{7.21}$$

where $p(\mathbf{x})$ is the mixture density. The mean loss is equal to the probability of an incorrect decision and represents a theoretical optimum – no other classifier setting can give a lower probability of the decision loss. Plots of a posteriori probabilities are shown in Figure 7.7, and corresponding discrimination hypersurfaces for a 3-class classifier can be seen in Figure 7.8.

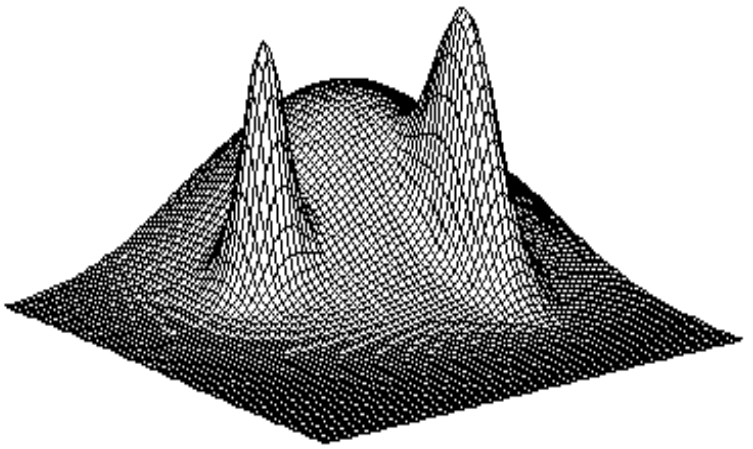

Figure 7.7: *Minimum error classifier: A posteriori probabilities.*

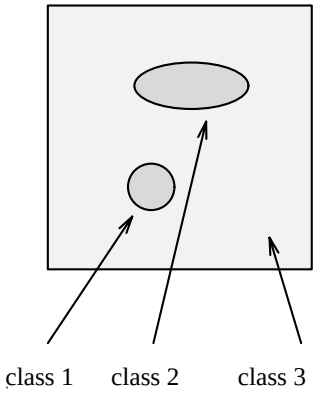

Figure 7.8: *Minimum error classifier: Discrimination hypersurfaces and resulting classes.*

Another criterion is the **best approximation criterion** which is based on the best approximation of discrimination functions by linear combinations of predetermined functions $\phi_i(\mathbf{x})$, $i = 1, \ldots, n$. The classifier is then constructed as a Φ-classifier.

Analytic minimization of the extrema problem (7.14) is in many practical cases impossible because the multi-dimensional probability densities are not available. Criteria for loss function evaluation can be found in [Sklansky 81, Devijver and Kittler 82]. The requirements for classification correctness, and the set of objects accompanied by information about their classes are usually available in practical applications – very often this is all the information that can be used for the classifier design and setting.

The ability to set classification parameters from a set of examples is very important and is called **classifier learning**. The classifier setting is based on a set of objects (represented by their feature vectors), each object being accompanied by information about its proper classification – this set of patterns and their classes is called the **training set**. Clearly, the quality of the classifier setting depends on the quality and size of the training set, which is always finite. Therefore, to design and set a classifier, it is not possible to use all the objects which will later need classifying; that is, the patterns that were not used for classifier design and setting will also enter the classifier, not merely the patterns contained in the training set. The classifier setting methods must be **inductive** in the sense that the information obtained from the elements of the training set must be generalized to cover the whole feature space, implying that the classifier setting should be (near) optimal for all feasible patterns, not only for those patterns that were present in the training set. In other words, the classifier should be able to recognize even those objects that it had never 'seen' before.

It may be that a solution for a given problem does not exist. If the requirements for classification correctness together with the set of training examples are given, it may be impossible to give an immediate answer as to whether the assignment can be fulfilled. The larger the training set, the better the guarantee that the classifier may be set correctly – classification correctness and the size of the training set are closely related. If the statistical properties of patterns are known, the necessary sizes of the training sets can be estimated, but the problem is that in reality they are not usually known. The training set is actually supposed to substitute this missing statistical information. Only after processing of the training set can the designer know whether it was sufficient, and whether an increase in the

training set size is necessary.

The training set size will typically be increased several times until the correct classification setting is achieved. The problem, which originally could not be solved, uses more and more information as the training set size increases until the problem specifications can be met.

The general idea of sequential increase in training set size can be understood as presenting small portions of a large training set to the classifier whose performance is checked after each portion. The smallest portion size is one element of the training set. Sequential processing of information (which cannot be avoided in principle) has some substantial consequences in the classifier setting process.

All the properties of the classifier setting methods given have analogies in the learning process of living organisms. The basic properties of learning can be listed as;

- **Learning** is the process of automated system optimization based on the sequential presentation of examples.

- The **goal of learning** is to minimize the optimality criterion. The criterion may be represented by the mean loss caused by incorrect decisions.

- The finite size of the training set requires the **inductive** character of learning. The goal of learning must be achieved by generalizing the information from examples, before all feasible examples have been presented. The examples may be chosen at random.

- The unavoidable requirements of sequential information presentation and the finite size of system memory necessitate the **sequential character of learning**. Therefore, learning is not a one-step process, but rather a step by step process of improvement.

The learning process searches out the optimal classifier setting from examples. The classifier system is constructed as a universal machine that becomes optimal after processing the training set examples (supervised learning), meaning that it is not necessary to repeat the difficult optimal system design if a new application appears. Learning methods do not depend on the application; the same learning algorithm can be applied if a medical diagnostics classifier is set just as if an object recognition classifier for a robot is set.

The quality of classifier decisions is closely related to the quality and amount of information that is available. From this point of view, the patterns should represent as complex a description as possible. On the other hand, a large number of description features would result. Therefore, the object description is always a trade-off between the permissible classification error, the complexity of the classifier construction, and the time required for classification. This results in a question of how to choose the best features from a set of available features, and how to detect the features with the highest contribution to the recognition success. Methods of determination of **informativity** and **discriminativity** of measured features can be found in [Fu 68, Young and Calvert 74, Devijver and Kittler 82, Pudil et al. 94a, Pudil et al. 94b].

7.2.3 Classifier learning

Two common learning strategies will be presented in this section:

- **Probability density estimation** estimates the probability densities $p(\mathbf{x}|\omega_r)$ and probabilities $P(\omega_r)$, $r = 1, \ldots, R$. The discrimination functions are computed according to the minimum error criterion (equation (7.19)).

- **Direct loss minimization** finds the decision rule $\omega = d(\mathbf{x}, \mathbf{q}^*)$ by direct minimization of losses $J(\mathbf{q})$ without estimation of probability densities and probabilities. The criterion of the best approximation is applied.

Probability density estimation methods differ in computational difficulty according to the amount of prior information available about them. If some prior information is available, it usually describes the shape of probability density functions $p(\mathbf{x}|\omega_r)$. The parameters describing the distribution are not usually known, and learning must find the estimate of these parameters. Therefore, this class of learning methods is sometimes called **parametric learning**.

Assume the patterns in the r-th class can be described by a normal distribution. The probability density for the normal distribution $N(\boldsymbol{\mu}_r, \boldsymbol{\Psi}_r)$ can be computed for patterns from the class ω_r

$$p(\mathbf{x}|\omega_r) = \frac{1}{(2\pi)^{\frac{n}{2}} \sqrt{(det\ \boldsymbol{\Psi}_r)}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_r)^T\ \boldsymbol{\Psi}_r^{-1}\ (\mathbf{x} - \boldsymbol{\mu}_r)\right] \tag{7.22}$$

where $\boldsymbol{\Psi}_r$ is the dispersion matrix (and we recall that $\mathbf{x}$, $\boldsymbol{\mu}_i$ are column vectors). Details about multivariate probability density function estimation may be found in [Rao 65, Johnson and Wichern 90]. The computation process depends on additional information about the vector of values $\boldsymbol{\mu}_r$ and $\boldsymbol{\Psi}_r$; three cases can be distinguished:

1. The dispersion matrix $\boldsymbol{\Psi}_r$ is known, but the mean value vector $\boldsymbol{\mu}_r$ is unknown. One of the feasible estimates of the mean value may be the average

$$\tilde{\boldsymbol{\mu}}_r = \overline{\mathbf{x}} \tag{7.23}$$

which can be computed recursively

$$\overline{\mathbf{x}}(k+1) = \frac{1}{k+1}(k\ \ \overline{\mathbf{x}}(k) + \mathbf{x}_{k+1}) \tag{7.24}$$

where $\overline{\mathbf{x}}(k)$ is the average computed from k samples, and $\mathbf{x}_{k+1}$ is the $(k+1)$-st pattern from the class r from the training set. This estimate is unbiased, consistent, efficient, and linear.

Alternatively, if the a priori estimate of the mean $\tilde{\boldsymbol{\mu}}_r(0)$ is available, the Bayes approach to estimation of the normal distribution parameters can be used. Then, the estimate can be computed recursively

$$\tilde{\boldsymbol{\mu}}_r(k+1) = \frac{a+k}{a+k+1}\tilde{\boldsymbol{\mu}}_r(k) + \frac{1}{a+k+1}\mathbf{x}_{k+1} \tag{7.25}$$

The parameter a represents the confidence in the a priori estimate $\tilde{\boldsymbol{\mu}}_r(0)$. In training, a specifies the number of steps during which the designer believes more in the a priori estimate than in the mean value so far determined by training. Note that for $a = 0$, the Bayes estimate is identical to that given in equation (7.24).

2. The dispersion matrix Ψ_r is unknown, but the mean value vector $\boldsymbol{\mu}_r$ is known. The estimate of the dispersion matrix Ψ_r if the mean value $\boldsymbol{\mu}_r$ is known is usually taken as

$$\tilde{\Psi}_r = \frac{1}{K}\sum_{k=1}^{K}(\mathbf{x}_k - \boldsymbol{\mu}_r)\ \ (\mathbf{x}_k - \boldsymbol{\mu}_r)^T \tag{7.26}$$

or in recursive form

$$\tilde{\Psi}_r(k+1) = \frac{1}{k+1}[k\ \ \tilde{\Psi}_r(k) + (\mathbf{x}_{k+1} - \boldsymbol{\mu}_r)\ \ (\mathbf{x}_{k+1} - \boldsymbol{\mu}_r)^T] \tag{7.27}$$

This estimate is unbiased and consistent.

As another option, if the a priori estimate $\tilde{\Phi}_r(0)$ of the dispersion matrix Ψ_r is known, the Bayes estimation approach can be applied. Let K be the number of samples in the training set, and $\tilde{\Psi}_r(K)$ be calculated as in equation (7.27). Then,

$$\tilde{\Phi}_r(K) = \frac{b\ \ \tilde{\Phi}_r(0) + K\ \ \tilde{\Psi}_r(K)}{b+K} \tag{7.28}$$

and $\tilde{\Phi}_r(K)$ is considered the Bayes estimate of the dispersion matrix Ψ_r. Parameter b represents the confidence in the a priori estimate $\tilde{\Phi}_r(0)$.

3. Both the dispersion matrix Ψ_r and the mean value vector $\boldsymbol{\mu}_r$ are unknown. The following estimates can be used

$$\tilde{\boldsymbol{\mu}}_r = \overline{\mathbf{x}} \tag{7.29}$$

$$\tilde{\Psi}_r = \mathbf{S} = \frac{1}{K-1}\sum_{k=1}^{K}(\mathbf{x}_k - \overline{\mathbf{x}})\ \ (\mathbf{x}_k - \overline{\mathbf{x}})^T \tag{7.30}$$

or in the recursive form

$$\begin{aligned}\mathbf{S}(k+1) = \quad & \tfrac{1}{k}[(k-1)\mathbf{S}(k) + \\ & (\mathbf{x}_{k+1} - \overline{\mathbf{x}}(k+1))\ \ (\mathbf{x}_{k+1} - \overline{\mathbf{x}}(k+1))^T + \\ & k(\overline{\mathbf{x}}(k) - \overline{\mathbf{x}}(k+1))(\overline{\mathbf{x}}(k) - \overline{\mathbf{x}}(k+1))^T]\end{aligned} \tag{7.31}$$

Alternatively, if the a priori estimate $\tilde{\Phi}_r(0)$ of the dispersion matrix Ψ_r and the a priori estimate $\tilde{\boldsymbol{\nu}}_r(0)$ of the mean vector for class r are known, the Bayes estimates can be determined as follows:

$$\tilde{\boldsymbol{\nu}}_r(K) = \frac{a\tilde{\boldsymbol{\mu}}_r(0) + K\tilde{\boldsymbol{\mu}}_r(K)}{a+K} \tag{7.32}$$

where K is the number of samples in the training set and $\tilde{\boldsymbol{\mu}}_r(K)$ is determined using equations (7.23) and (7.24). The dispersion matrix estimate is calculated as

$$\begin{aligned}\tilde{\Phi}_r(K) = \quad & \tfrac{b}{b+K}\ \ \tilde{\Phi}_r(0) + a\tilde{\boldsymbol{\nu}}_r(0)\tilde{\boldsymbol{\nu}}_r(0)^T + (K-1)\tilde{\Psi}_r(K) \\ & + K\tilde{\boldsymbol{\mu}}_r(K)\tilde{\boldsymbol{\mu}}_r(K)^T - (a+K)\tilde{\boldsymbol{\nu}}_r(K)\tilde{\boldsymbol{\nu}}_r(K)^T\end{aligned} \tag{7.33}$$

where $\tilde{\Psi}_r(K)$ is calculated as given in equation 7.27. Then, $\tilde{\boldsymbol{\nu}}_r(K)$ and $\tilde{\Phi}_r(K)$ are considered the Bayes estimates of the mean vector and the dispersion matrix for class r, respectively. Again, parameters a, b represent the confidence in the a priori estimates of $\tilde{\Phi}_r(0)$ and $\tilde{\boldsymbol{\nu}}_r(0)$.

The a priori probabilities of classes $P(\omega_r)$ are estimated as relative frequencies

$$P(\omega_r) = \frac{K_r}{K} \tag{7.34}$$

where K is the total number of objects in the training set; K_r is the number of objects from the class r in the training set.

Algorithm 7.1: Learning and classification based on estimates of probability densities assuming the normal distribution

1. Learning: Compute the estimates of the mean value vector $\boldsymbol{\mu}_r$ and the dispersion matrix $\boldsymbol{\Psi}_r$, equations (7.24) and/or (7.27), (7.31).

2. Compute the estimates of the a priori probability densities $p(\mathbf{x}|\omega_r)$, equation (7.22).

3. Compute the estimates of the a priori probabilities of classes, equation (7.34).

4. Classification: Classify all patterns into the class r if

$$\omega_r = \max_{i=1,\ldots,s} (p(\mathbf{x}|\omega_i)\, P(\omega_i))$$

(equations (7.19) and (7.8)).

If no prior information is available (i.e. even the distribution type is not known), the computation is more complex. In such cases, if it is not necessary to use the minimum error criterion, it is advantageous to use a direct loss minimization method.

No probability densities or probabilities are estimated in the second group of methods based on direct minimization of losses. The minimization process can be compared to gradient optimization methods, however pure gradient methods cannot be used because of unknown probability densities, so the gradient cannot be evaluated. Nevertheless, the minimum can be found using methods of **stochastic approximations** that are discussed in [Sklansky 81].

The most important conclusion is that the learning algorithms can be represented by recursive formulae in both groups of learning methods and it is easy to implement them.

We have noted that the most common and easily implementable classifier is the minimum distance classifier. Its learning and classification algorithm is;

Algorithm 7.2: Minimum distance classifier learning and classification

1. Learning: For all classes, compute class exemplars $\mathbf{v}_i$ based on the training set

$$\mathbf{v}_i(k_i + 1) = \frac{1}{k_i + 1}(k_i \mathbf{v}_i(k_i) + \mathbf{x}_i(k_i + 1)) \tag{7.35}$$

where $\mathbf{x}_i(k_i + 1)$ are objects from the class i and k_i denotes the number of objects from class i used thus far for learning.

2. Classification: For an object description vector $\mathbf{x}$ determine the distance of $\mathbf{x}$ from the class exemplars $\mathbf{v}_i$. Classify the object into the class j if the distance of $\mathbf{x}$ from $\mathbf{v}_j$ is the minimum such (equation (7.10)).

7.2.4 Cluster analysis

We noted earlier that classification methods exist which do not need training sets for learning. In particular, they do not need information about the class of objects in the learning stage, but learn them without a teacher (unsupervised learning). One such group of classification methods is called **cluster analysis**. Cluster analysis can be applied in classification if for any reason the training set cannot be prepared, or if examples with known class evaluation are not available.

Cluster analysis methods divide the set of processed patterns into subsets (clusters) based on the mutual similarity of subset elements. Each cluster contains patterns representing objects that are similar according to the selected object description and similarity criteria. Objects that are not similar reside in different clusters.

There are two main groups of cluster analysis methods – the first is hierarchical and the second non-hierarchical. Hierarchical methods construct a clustering tree; the set of patterns is divided into the two most dissimilar subsets, and each subset is divided into other different subsets, etc. Non-hierarchical methods sequentially assign each pattern to one cluster. Methods and algorithms for cluster analysis can be found in [Duda and Hart 73, Dubes and Jain 80, Devijver and Kittler 82, Blashfield et al. 82, Romesburg 84, McQuitty 87, Kaufman and Rousseeuw 90, Schalkoff 92, Everitt and Brian 93, Arabie et al. 96].

Non-hierarchical cluster analysis methods are either parametric or non-parametric. Parametric approaches are based on known class-conditioned distributions and require distribution parameter estimation that is similar to that used in minimum error classification described in Section 7.2.3. Parametric clustering approaches used for threshold-based image segmentation were also described in Section 5.1.2.

Non-parametric cluster analysis is a popular, simple, and practically useful non-hierarchical approach to cluster analysis. The **MacQueen k-means** cluster analysis method is a well-known example of this approach [MacQueen 67]. We need to assume that the number of clusters k is known – if it is not, it can be determined as the number of classes that gives the maximum confidence in results, or some more complex clustering method can be applied that does not need this information. The starting cluster points are constructed in the first step, represented by k points in the n-dimensional feature space. These points can either be selected at random from the clustered set of patterns, or the first k patterns from the set can be chosen. If there are exemplars of clusters available, even if these exemplars are unreliable, it is worthwhile using them as the starting cluster points. The method has two main stages; patterns are allocated to one of the existing clusters in the first stage according to their distance from the cluster exemplars, choosing the closest. Then the exemplar is recomputed as the center of gravity of all patterns in that cluster. If all the patterns from the set have been processed, the current exemplars of clusters are considered final; all the patterns are assigned to one of the clusters, represented by the exemplars determined in the first stage. Then the

patterns are (re-)assigned to clusters according to their distance from the exemplars, patterns being assigned to the closest cluster. The exemplars are not recomputed in the second stage. It should be clear that elements that were used for the starting cluster point definitions need not be members of the same clusters at the end.

Algorithm 7.3: MacQueen k-means cluster analysis

1. Define the number of clusters.
2. Initialize the cluster starting points (exemplars, initial guesses) $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$. Usually some patterns are chosen to serve as cluster starting points, perhaps chosen at random.
3. First Pass: Decide to which cluster each pattern belongs, choosing the closest (do not process those patterns that were used to initialize clusters). Recompute the relevant exemplar after an object is added to a cluster, possibly using equation (7.35).
4. Second Pass: Let the final exemplars be exemplars of resulting clusters. Classify all objects (including those used to form starting exemplars) using the final exemplars from the first pass. Use the same distance criterion as in the first pass.

Because of its simplicity, the MacQueen method has its limitations. There are many variations on this algorithm; one is to repeat the second stage until convergence. The ISODATA cluster analysis method [Dubes and Jain 76, Kaufman and Rousseeuw 90] may solve a complex clustering problem better. ISODATA uses two parameter sets, one which does not change during the clustering, and another which can be interactively adjusted until an acceptable clustering result is obtained. ISODATA represents a set of non-hierarchical cluster analysis methods from which the best can be picked.

Determining the number of clusters has not been mentioned; for example, what metric is the most suitable in n-dimensional space, etc. Answers to these and many other questions can be found in [Romesburg 84, McQuitty 87, Kaufman and Rousseeuw 90].

Note that statistical pattern recognition and cluster analysis can be combined. For instance, the minimum distance classifier can be taught using cluster analysis methods, cluster exemplars can be considered class exemplars, these exemplars can be assigned appropriate names and other patterns can be recognized using the resulting classifier [Sonka 86]. Additionally, fuzzy clustering approaches were reported with good results [Bezdek 81] (Section 7.7).

7.3 Neural nets

Neural nets have seen an explosion of interest since their rediscovery as a pattern recognition paradigm in the early 1980s. The value of some of the applications for which they are used may be arguable, but there is no doubt that they represent a tool of great value in various areas generally regarded as 'difficult', particularly speech and visual pattern recognition.

Most neural approaches are based on combinations of elementary processors (neurons) each of which takes a number of inputs and generates a single output. Associated with each input is a weight, and the output (in most cases) is then a function of the weighted sum

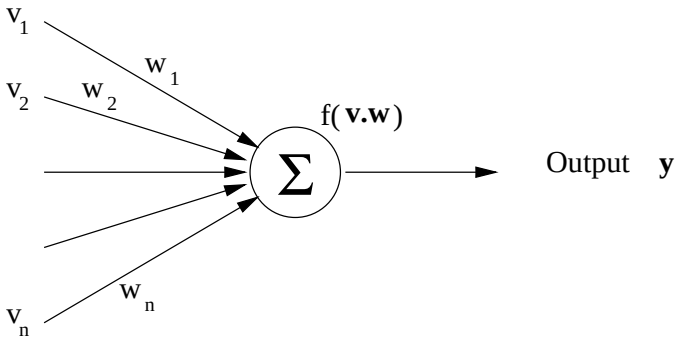

Figure 7.9: *A simple (McCulloch Pitts) neuron*

of inputs; this output function may be discrete or continuous, depending on the variety of network in use. A simple neuron is shown in Figure 7.9 – this model is derived from pioneering work on neural simulation conducted over 50 years ago [McCulloch and Pitts 43]. The inputs are denoted by $v_1, v_2, \ldots$, and the weights by $w_1, w_2, \ldots$; the total input to the neuron is then

$$x = \sum_{i=1}^{n} v_i w_i \tag{7.36}$$

or, more generally,

$$x = \sum_{i=1}^{n} v_i w_i - \theta \tag{7.37}$$

where θ is a threshold associated with this neuron. Also associated with the neuron is a **transfer function** $f(x)$ which provides the output; common examples are

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{7.38}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{7.39}$$

This model saw a lot of enthusiastic use during an early phase, culminating in Rosenblatt's **perceptron** [Rosenblatt 62].

The general idea of collections (networks) of these neurons is that they are interconnected (so the output of one becomes the input of another, or others) – this idea mimics the high-level of interconnection of elementary neurons found in brains which is thought to explain the damage resistance and recall capabilities of humans. Such an interconnection may then take some number of external inputs and deliver up some (possibly different) number of external outputs – see Figure 7.10. What lies between then specifies the network: This may mean a large number of heavily interconnected neurons, or some highly structured (e.g. layered) interconnection, or, pathologically, nothing (so that inputs are connected straight to outputs).

There are many uses to which such a structure may be put; the general task being performed is vector association. Examples may be;

- Classification: If the output vector (m-dimensional) is binary and contains only a single one, the position of the one classifies the input pattern into one of m categories.

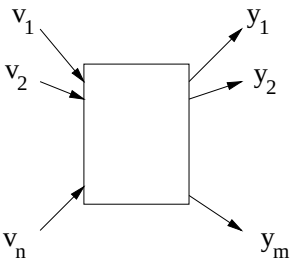

Figure 7.10: *A neural network as a vector associator*

- Auto-association: Some uses of neural networks cause them to regenerate the input pattern at the outputs (so $m = n$ and $v_i = y_i$); the purpose of this may be to derive a more compact vector representation from within the network internals.

- General association: At their most interesting, the vectors $\mathbf{v}$ and $\mathbf{y}$ represent patterns in different domains, and the network is forming a correspondence between them. One of the most quoted examples of this is NetTalk [Sejnowski and Rosenberg 87], in which the inputs represent a stream of written text and the outputs are phonemes – thus the network is a speech generator.

7.3.1 Feedforward networks

The first neural networks involved no 'internals' (so the box in Figure 7.10 was empty); these early perceptrons had a training algorithm developed which was shown to converge *if a solution to the problem at hands exists* [Minsky 88]; unfortunately, this caveat proved very restrictive, requiring that the classification being performed be linearly separable (vector clusters of interest lay in distinct half-spaces). This restriction was overcome by the now very popular **back-propagation** algorithm [Rumelhart and McClelland 86] which trains strictly layered networks in which it is assumed that at least one layer exists between input and output (it fact, it can be shown that two such 'hidden' layers always suffice [Kolmogorov 63, Hecht-Nielson 87]). Such a network is shown in Figure 7.11, and is an example of a **feed-forward** network, in which data are admitted at the inputs and travel in one direction toward the outputs, at which the 'answer' may be read.

The standard approach to use of such networks is to obtain a training set of data – a set of vectors for which the 'answer' is already known. This is used to teach a network with some training algorithm, such that the network can perform the association accurately. Then, in classification (or 'live') mode, unknown patterns are fed into the net and it produces answers based on generalizing what it has learned.

Back-propagation proceeds by comparing the output of the network to that expected, and computing an error measure based on sum of square differences. This is then minimized using gradient descent by altering the weights of the network. Denoting a member of the training set by $\mathbf{v}^i$, the actual outputs by $\mathbf{y}^i$ and the *desired* outputs by ω^i, the error is

$$E = \sum_i \sum_j (y_j^i - \omega_j^i)^2$$

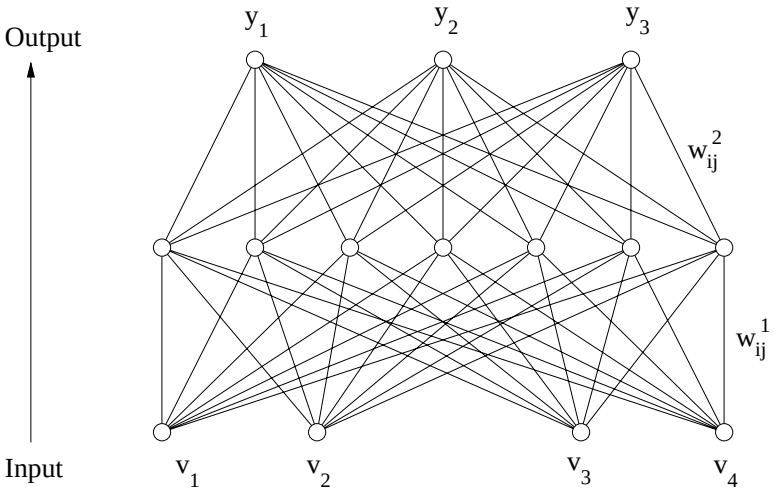

Figure 7.11: *A three-layered neural net structure.*

(thus summing square difference over the entire training set) and the algorithm performs the updates

$$w_{ij}(k+1) = w_{ij}(k) - \epsilon \frac{\partial E}{\partial w_{ij}} \tag{7.40}$$

iteratively until 'good' performance is seen (k here counts the iterations of the updates).

The literature on back-propagation is large and thorough and we present here a summary of the algorithm only;

Algorithm 7.4: Back-propagation learning

1. Assign small random numbers to the weights w_{ij}.

2. Input a pattern $\mathbf{v}$ from the training set and evaluate the neural net output $\mathbf{y}$.

3. Learning: If $\mathbf{y}$ does not match the required output vector ω, the weights must be adjusted

$$w_{ij}(k+1) = w_{ij}(k) + \epsilon \delta_j z_i(k) \tag{7.41}$$

where ϵ is called the **learning constant** or **learning rate**, $z_i(k)$ is the output of the node i, k is the iteration number, δ_j is an error associated with the node j in the adjacent upper level

$$\delta_j = \begin{cases} y_j(1-y_j)(\omega_j - y_j) & \text{for output node } j \\ z_j(1-z_j)(\sum_l \delta_l w_{jl}) & \text{for hidden node } j \end{cases} \tag{7.42}$$

4. Go to 2 and fetch the next input pattern.

5. Repeat steps 2 to 4 until each training pattern outputs a suitably good approximation to that expected. Each circuit of this loop is termed an **epoch**.

The convergence process can be very slow and there is an extensive literature on speeding the algorithm (see, for example, [Haykin 94]). The best known of these techniques is the introduction of **momentum** , which accelerates convergence across plateaux of the cost surface, and controls behavior in steep ravines. This approach rewrites equation (7.40) as

$$\Delta w_{ij} = \epsilon \frac{\partial E}{\partial w_{ij}}$$

and updates it to

$$\Delta w_{ij} := \epsilon \frac{\partial E}{\partial w_{ij}} + \epsilon \Delta w_{ij}$$

which updates equation (7.41) to

$$w_{ij}(k+1) = w_{ij}(k) + \epsilon \delta_j z_i(k) + \alpha(w_{ij}(k) - w_{ij}(k-1)) \tag{7.43}$$

α is called the **momentum constant**, and is chosen to be between 0 and 1, having the effect of contributing a proportion of the update of the previous iteration into the current one. Thus, in areas of very low gradient, some movement continues.

7.3.2 Unsupervised learning

A different class of networks exists which are self-teaching – that is, they do not depend on the net being exposed to a a training set with known information about classes, but are able to self-organize themselves to recognize patterns automatically. Various types of network exist under this general heading, of which the best known are Kohonen feature maps.

Kohonen maps will take as input n-dimensional data vectors and generate an n dimensional output that, within the domain of the problem at hand, 'best represents' the particular input given. More precisely, the network has a layer of neurons each of which is connected to all n input vector components; each neuron calculates its input (equation 7.36), and that with the largest input is regarded as the 'winner'; the n weights associated with the input arcs to this node then represent the output. Figure 7.12 illustrates this. The weights are

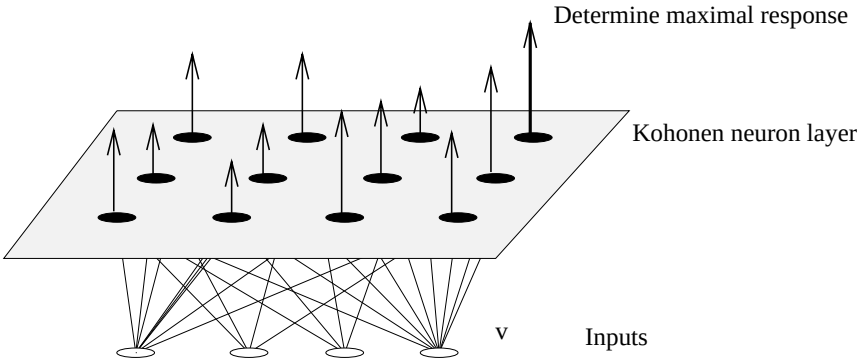

Figure 7.12: *Kohonen self-organizing neural net.*

updated using a learning algorithm that finds for itself the data structure (that is, no prior classification is needed or indeed known). It may be clear that such a network is performing the role of clustering – similar inputs will generate the same output.

The theory underlying Kohonen networks is derived from the operation of biological neurons which are known to exist in locally 2D layers, and in which neural responses are known to cluster. The derivation of the algorithm may be found in various standard texts [Kohonen 88, Kohonen 95], but may be summarized as;

Algorithm 7.5: Unsupervised learning of the Kohonen feature map

1. Assign random numbers with a small variance around the average values of the input feature vector elements to the weights w_{ij}.
2. Collect a sample of vectors $V = \{\mathbf{v}\}$ from the set to be analyzed.
3. Select a new vector $\mathbf{v} \in V$, determine the neuron with the biggest input;
$$j^* = argmax_j \sum_i w_{ij} v_i$$
4. For all neurons n_j within a neighborhood of radius r of n_{j^*}, perform the weight update with step size $\alpha > 0$ (learning rate)
$$w_{ij} := w_{ij} + \alpha(v_i - w_{ij}) \tag{7.44}$$
5. Go to 3
6. Reduce r and α, and go to 3.

Kohonen networks have enjoyed considerable use, often as components of larger system which may include other varieties of neural network.

Several other varieties of self-teaching net exist, of which the best known is perhaps ART (Adaptive Resonance Theory) [Carpenter and Grossberg 87b, Carpenter and Grossberg 87a]. More specialized texts will provide ample detail.

7.3.3 Hopfield neural nets

Hopfield nets are mostly used in optimization problems [Hopfield and Tank 85, Hopfield and Tank 86]; however it is possible to represent recognition as an optimization task – find the maximum similarity between a pattern $\mathbf{x}$ and one of the existing exemplars $\mathbf{v}$.

In the Hopfield neural model, the network does not have designated inputs and outputs, but rather the current configuration represents its state. The neurons, which are fully interconnected, have discrete (0/1 or -1/1) outputs, calculated from equation 7.38. Weights between neurons do not evolve (learn), but are computed from a set of known exemplars at initialization;
$$w_{ij} = \sum_r (v_i^r v_j^r) \qquad (i \neq j) \tag{7.45}$$
where w_{ij} is the interconnection weight between nodes i and j; and v_i^r is the i^{th} element of the r^{th} exemplar; $w_{ii} = 0$ for any i.

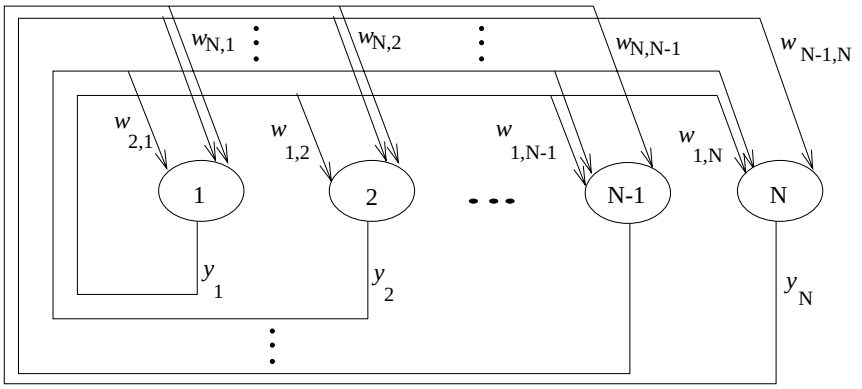

Figure 7.13: *Hopfield recurrent neural net.*

The Hopfield net acts as an associative memory where the exemplars are stored; its architecture is shown in Figure 7.13. When used for recognition, the feature vector to be classified enters the net in the form of initial values of node outputs. The Hopfield net then recurrently iterates using existing interconnections with fixed weights until a stable state is found – that such a state is reached can be proved under certain conditions (for which equation (7.45) is sufficient). The resulting stable state should be equal to the values of the exemplar that is closest to the processed feature vector in the Hamming metric sense. Supposing these class exemplars $\mathbf{v}^r$ are known, the recognition algorithm is:

Algorithm 7.6: Recognition using a Hopfield net

1. Based on existing exemplars $\mathbf{v}^i$ of r classes, compute the interconnection weights w_{ij} (equation (7.45)).
2. Apply an unknown feature vector $\mathbf{x}$ as initial outputs $\mathbf{y}(0)$ of the net.
3. Iterate until the net converges (output $\mathbf{y}$ does not change):

$$y_j(k+1) = f(\sum_{i=1}^{N}(w_{ij}y_i(k))) \tag{7.46}$$

The final output vector $\mathbf{y}$ represents the exemplar of the class into which the processed feature vector $\mathbf{x}$ is classified. In other words, a Hopfield neural net transforms a non-ideal representation of an object (fuzzy, noisy, incomplete, etc.) to the ideal exemplar representation. The transformation of a noisy binary image of characters to a clear letter is one vision-related application; binary image recognition examples can be found in [Kosko 91, Rogers and Kabrisky 91].

The Hopfield neural net converges by seeking a minimum of a particular function – this is usually a *local* minimum, which may mean that the correct exemplar (the *global* minimum) is not found. Moreover, the number of local minima grows rapidly with the number of exemplars

stored in the associative network. It can be shown that the minimum number of nodes N required is about seven times the number of memories M to be stored (this is known as the $0.15N \geq M$ rule) [McEliece et al. 87, Amit 89], causing a rapid increase in the number of necessary nodes.

This overview has shown only the main principles of neural nets and their connections to conventional statistical pattern recognition, and we have not discussed many alternative neural net techniques, methods, and implementations. The state of the art and many references may be found in a set of selected papers [Carpenter and Grossberg 91] and in [Amit 89, Hecht-Nielsen 90, Judd 90, Simpson 90, Mozer 91, Zhou 92, Masters 95], and various useful introductory texts [Wasserman 89, Carling 92, Nigrin 93, Fausett 94, Braspenning et al. 95]. Some interesting applications, many in the visual domain, may be found in [Linggard et al. 92].

7.4 Syntactic pattern recognition

Quantitative description of objects using numeric parameters (the feature vector) is used in statistical pattern recognition, while **qualitative** description of an object is a characteristic of syntactic pattern recognition. The object structure is contained in the syntactic description. Syntactic object description should be used whenever feature description is not able to represent the complexity of the described object and/or when the object can be represented as a hierarchical structure consisting of simpler parts. The elementary properties of the syntactically described objects are called **primitives** (Section 6.2.4 covered the syntactic description of object borders using border primitives, these border primitives representing parts of borders with a specific shape). Graphical or relational descriptions of objects where primitives represent subregions of specific shape is another example (see Sections 6.3.3 to 6.3.5). After each primitive has been assigned a symbol, relations between primitives in the object are described, and a **relational structure** results (Chapters 3 and 6). As in statistical recognition, the design of description primitives and their relation is not algorithmic. The design is based on the analysis of the problem, designer experience and abilities. However, there are some principles that are worth following:

1. The number of primitive types should be small.
2. The primitives chosen must be able to form an appropriate object representation.
3. Primitives should be easily segmentable from the image.
4. Primitives should be easily recognizable using some statistical pattern recognition method.
5. Primitives should correspond with significant natural elements of the object (image) structure being described.

For example, if technical drawings are described, primitives are line and curve segments, binary relations describe relations such as *to be adjacent*, *to be left of*, *to be above*, etc. This description structure can be compared with the structure of a natural language. The text consists of sentences, sentences consist of words, words are constructed by concatenation of letters. Letters are considered primitives in this example; the set of all letters is called the

alphabet. The set of all words in the alphabet that can be used to describe objects from one class (the set of all feasible descriptions) is named the **description language** which represents descriptions of all objects in the specific class. In addition, a **grammar** exists to represent a set of rules that must be followed when words of the specific language are constructed from letters (of the alphabet). Grammars can describe infinite languages as well. These definitions will be considered in more detail in Section 7.4.1.

Assume that the object is appropriately described by some primitives and their relations. Moreover, assume that the grammar is known for each class that generates descriptions of all objects of the specified class. Syntactic recognition decides whether the description word is or is not syntactically correct according to the particular class grammars, meaning that each class consists only of objects whose syntactic description can be generated by the particular grammar. Syntactic recognition is a process that looks for the grammar that can generate the syntactic word that describes an object.

We mentioned relational structure in the correspondence with the syntactic description of objects. Each relational structure with multiple relations can be transformed to a relational structure with at most binary relations; the image object is then represented by a **graph** which is **planar** if relations with adjacent regions only are considered. A graphical description is very natural, especially in the description of segmented images – examples were given in Section 6.3. Each planar graph can be represented either by a graph grammar or by a sequence of symbols (chain, word, etc.) over an alphabet. Sequential representation is not always advantageous in image object recognition because the valuable correspondence between the syntactic description and the object may be lost. Nevertheless, work with chain grammars is more straightforward and understandable and all the main features of more complex grammars are included in chain grammars. Therefore, we will principally discuss sequential syntactic descriptions and chain grammars. More precise and detailed discussion of grammars, languages, and syntactic recognition methods can be found in [Fu 74, Fu 77, Chen 76, Pavlidis 77, Rosenfeld 79b, Fu 80, Pavlidis 80].

The syntactic recognition process is described by the following algorithm.

Algorithm 7.7: Syntactic recognition

1. Learning: Based on the problem analysis, define the primitives and their possible relations.

2. Construct a description grammar for each class of objects using either hand analysis of syntactic descriptions or automated grammar inference (see Section 7.4.3).

3. Recognition: For each object, extract its primitives first; recognize the primitives' classes and describe the relations between them. Construct a description word representing an object.

4. Based on the results of the syntactic analysis of the description word, classify an object into that class for which its grammar (constructed in step (2)) can generate the description word.

It can be seen that the main difference between statistical and syntactic recognition is in the learning process. Grammar construction can rarely be algorithmic using today's approaches, requiring significant human interaction. It is usually found that the more complex the primitives are, the simpler is the grammar, and the simpler and faster is the syntactic analysis. More complex description primitives on the other hand make step (3) of the algorithm more difficult and more time consuming; also, primitive extraction and evaluation of relations may not be simple.

7.4.1 Grammars and languages

Assuming that the primitives have been successfully extracted, all the inter-primitive relations can then be syntactically described as n-ary relations; these relations form structures (chains, trees, graphs) called **words** that represent the object or the pattern. Each pattern is therefore described by a word. Primitive classes can be understood as letters from the alphabet of symbols called **terminal symbols**. Let the alphabet of terminal symbols be V_t.

The set of patterns from a particular class corresponds to a set of words. This set of words is called the **formal language** which is described by a **grammar**. The grammar is a mathematical model of a generator of syntactically correct words (words from the particular language); it is a quadruple

$$G = [V_n, V_t, P, S] \tag{7.47}$$

where V_n and V_t are disjoint alphabets, elements of V_n are called **non-terminal symbols**, and elements of V_t are terminal symbols. Define V^* to be the set of all empty or non-empty words built from the terminal and/or non-terminal symbols. The symbol S is the grammar axiom or the *start* symbol. The set P is a non-empty finite subset of the set $V^* \times V^*$; elements of P are called the substitution rules. The set of all words that can be generated by the grammar G is called the **language** $L(G)$. Grammars that generate the same language are called **equivalent**.

A simple example will illustrate this terminology. Let the words generated by the grammar be squares of arbitrary size with sides parallel to the co-ordinate axes, and let the squares be represented by the Freeman chain code of the border in 4-connectivity (see Section 6.2.1). There are four terminal symbols (primitives) of the grammar in this case, $V_t = \{0, 1, 2, 3\}$. Let the non-terminal symbols be $V_n = \{s, a, b, c, d\}$. Note that the terminal symbols correspond to natural primitives of the 4-connectivity Freeman code; the non-terminal symbols were chosen from an infinite set of feasible symbols. The set of substitution rules P demonstrates how the start symbol $S = s$ can be transformed to words corresponding to the Freeman chain code description of squares:

$$\begin{array}{lllll} P: & (1) & s & \rightarrow & abcd \\ & (2) & aAbBcCdD & \rightarrow & a1Ab2Bc3Cd0D \\ & (3) & aAbBcCdD & \rightarrow & ABCD \end{array} \tag{7.48}$$

where A (B, C, D, respectively) is a variable representing any chain (including an empty one) consisting only of terminal symbols 1 ($2, 3, 0$). Rule (3) stops the word generating process. For example, a square with a side length $l = 2$ with the Freeman chain description 11223300 is generated by the following sequence of substitution rules (see Figure 7.14)

$$s \rightarrow^1 abcd \rightarrow^2 a1b2c3d0 \rightarrow^2 a11b22c33d00 \rightarrow^3 11223300$$

where the arrow superscript refers to the appropriate substitution rule. The simple analysis of generated words shows that the language generated consists only of Freeman chain code representations of squares with sides parallel to the plane co-ordinates.

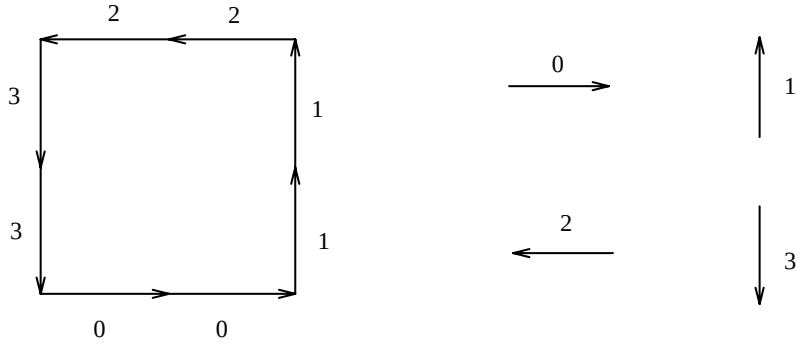

Figure 7.14: *Square shape description.*

Grammars can be divided into four main groups ordered from the general to the specific [Chomsky 66, Chomsky et al. 71]:

1. Type 0 – **General Grammars**
   There are no limitations for the substitution rules.

2. Type 1 – **Context-Sensitive Grammars**
   Substitution rules can be of the form

$$W_1 \alpha W_2 \rightarrow W_1 U W_2 \tag{7.49}$$

   that can contain the substitution rule $S \rightarrow e$ where e is an empty word; words W_1, W_2, U consist of elements of V^*, $U \neq e$, $\alpha \in V_n$. This means that the non-terminal symbols can be substituted by the word U in the context of words W_1 and W_2.

3. Type 2 – **Context-Free Grammars**
   Substitution rules have the form

$$\alpha \rightarrow U \tag{7.50}$$

   where $U \in V^*$, $U \neq e$, $\alpha \in V_n$. Grammars can contain the rule $S \rightarrow e$. This means that the non-terminal symbol can be substituted by a word U independently of the context of α.

4. Type 3 – **Regular Grammars**
   The substitution rules of regular grammars are of the form

$$\alpha \rightarrow x\beta \qquad or \qquad \alpha \rightarrow x \tag{7.51}$$

   where $\alpha, \beta \in V_n$, $x \in V_t$. The substitution rule $S \rightarrow e$ may be included.

All the grammars discussed so far have been **non-deterministic**. The same left hand side might appear in several substitution rules with different right hand sides, and no rule exists that specifies which rule should be chosen. A non-deterministic grammar generates

a language in which no words are 'preferred'. If it is advantageous to generate some words (those more probable) more often than others, substitution rules can be accompanied by numbers (for instance by probabilities) that specify how often the substitution rule should be applied. If the substitution rules are accompanied by probabilities, the grammar is called **stochastic**. If the accompanying numbers do not satisfy the properties of probability (unit sum of probabilities for all rules with the same left hand side), the grammar is called **fuzzy** [Zimmermann et al. 84].

Note, that the evaluation of the frequency with which each substitution rule should be used can substantially increase the efficiency of syntactic analysis in the recognition stage [Fu 74].

7.4.2 Syntactic analysis, syntactic classifier

If appropriate grammars exist that can be used for representation of all patterns in their classes, the last step is to design a syntactic classifier which assigns the pattern (the word) to an appropriate class. It is obvious that the simplest way is to construct a separate grammar for each class; an unknown pattern x enters a parallel structure of blocks that can decide if $x \in L(G_j)$ where $j = 1, 2, \ldots R$ and R is the number of classes; $L(G_j)$ is the language generated by the j^{th} grammar. If the j^{th} block's decision is positive, the pattern is accepted as a pattern from the j^{th} class and the classifier assigns the pattern to the class j. Note that generally more than one grammar can accept a pattern as belonging to its class.

The decision of whether or not the word can be generated by a particular grammar is made during **syntactic analysis**. Moreover, syntactic analysis can construct the pattern derivation tree which can represent the structural information about the pattern.

If a language is finite (and of a reasonable size), the syntactic classifier can search for a match between the word being analyzed and all the words of the language. Another simple syntactic classifier can be based on comparisons of the chain word descriptions with typical representatives of classes comparing primitive type presence only. This method is very fast and easily implemented, though it does not produce reliable results since the syntactic information is not used at all. However, impossible classes can be rejected in this step which can speed up the syntactic analysis process.

Syntactic analysis is based on efforts to construct the tested pattern by the application of some appropriate sequence of substitution rules to the start symbol. If the substitution process is successful, the analysis process stops and the tested pattern can be generated by the grammar. The pattern can be classified into the class represented by the grammar. If the substitution process is unsuccessful, the pattern is not accepted as representing an object of this class.

If the class description grammar is regular (type 3), syntactic analysis is very simple. The grammar can be substituted with a finite non-deterministic automaton and it is easy to decide if the pattern word is accepted or rejected by the automaton [Fu 82]. If the grammar is context-free (type 2), the syntactic analysis is more difficult. Nevertheless, it can be designed using stack automata.

Generally, which process of pattern word construction is chosen is not important; the transformation process can be done in top-down or bottom-up manner.

A top-down process begins with the start symbol and substitution rules are applied in

the appropriate way to obtain the same pattern word as that under analysis. The final goal of syntactic analysis is to generate the same word as the analyzed word; every partial substitution creates a set of subgoals, just as new branches are created in the generation tree. Effort is always devoted to fulfill the current subgoal. If the analysis is not successful in fulfilling the subgoal, it indicates an incorrect choice of the substitution rule somewhere in the previous substitutions, and backtracking is invoked to get back to the nearest higher tree level (closer to the root), and to pick another applicable rule. The process of rule applications and backtracking is repeated until the required pattern word results. If the whole generating process ends unsuccessfully, the grammar does not generate the word, and the analyzed pattern does not belong to the class.

This top-down process is a series of expansions starting with the start symbol S. A bottom-up process starts with the analyzed word, which is **reduced** by applying reverse substitutions, the final goal being to reduce the word to the start symbol S. The main principle of bottom-up analysis is to detect subwords in the analyzed word that match the pattern on the right hand side of some substitution rule, then the reduction process substitutes the former right hand side with the left hand side of the rule in the analyzed word. The bottom-up method follows no subgoals, all the effort is devoted to obtaining a reduced and simplified word pattern until the start symbol is obtained. Again, if the process is not successful, the grammar does not generate the analyzed word.

The pure top-down approach is not very efficient since too many incorrect paths are generated. The number of misleading paths can be decreased by application of consistency tests. For example, if the word starts with a non-terminal symbol I, only rules with the right hand side starting with I should be considered. Many more consistency tests can be designed that take advantage of prior knowledge. This approach is also called **tree pruning** (See Figure 7.15) [Nilsson 71, Nilsson 82].

Tree pruning is often used if an exhaustive search cannot be completed because the search effort would exceed any reasonable bounds. Note that pruning can mean that the final solution is not optimal or may not be found at all (especially if tree search is used to find the best path through the graph, Section 5.2.4). This depends on the quality of the a priori information that is applied during the pruning process.

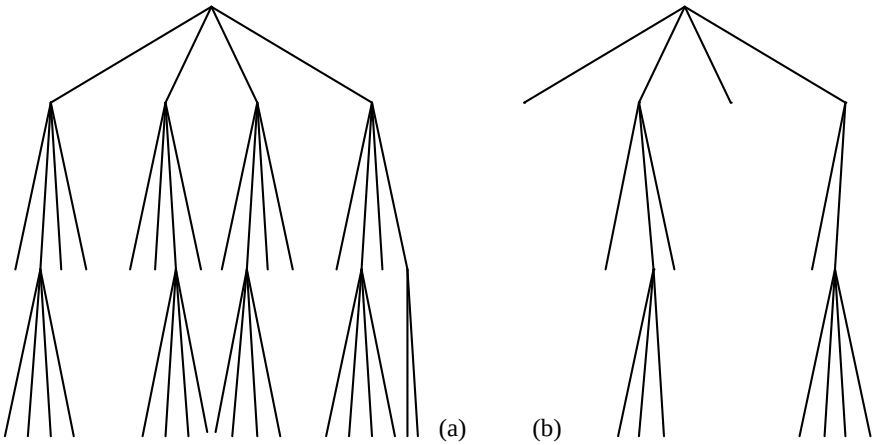

Figure 7.15: *Tree pruning: (a) Original tree, (b) pruning decreases size of the searched tree.*

There are two main principles for recovery from following a wrong path. The first one is represented by the backtracking mechanism already mentioned, meaning that the generation of words returns to the nearest point in the tree where another substitution rule can be applied which has not yet been applied. This approach requires the ability to reconstruct the former appearances of generated subwords and/or remove some branches of the derivation tree completely.

The second approach does not include backtracking. All possible combinations of the substitution rules are applied in parallel and several generation trees are constructed simultaneously. If any tree succeeds in generating the analyzed word, the generation process ends. If any tree generation ends with a non-successful word, this tree is abandoned. The latter approach uses more brute force, but the algorithm is simplified by avoiding backtracking.

It is difficult to compare the efficiency of these two and the choice depends on the application; Bottom-up analysis is more efficient for some grammars, and top-down is more efficient for others. As a general observation, the majority of syntactic analyzers which produce all generated words is based on the top-down principle. This approach is appropriate for most grammars but is usually less efficient.

Another approach to syntactic analysis uses example relational structures of classes. The syntactic analysis consists of matching the relational structure that represents the analyzed object with the example relational structure. The main goal is to find an **isomorphism** of both relational structures. These methods can be applied to n-ary relational structures as well. Relational structure matching is a perspective approach to syntactic recognition, a perspective way of image understanding (see Section 7.5). A simple example of relational structure matching is shown in Figure 7.16. A detailed description of relational structure matching approaches can be found in [Barrow and Popplestone 71, Pavlidis 77, Ballard and Brown 82, Baird 84].

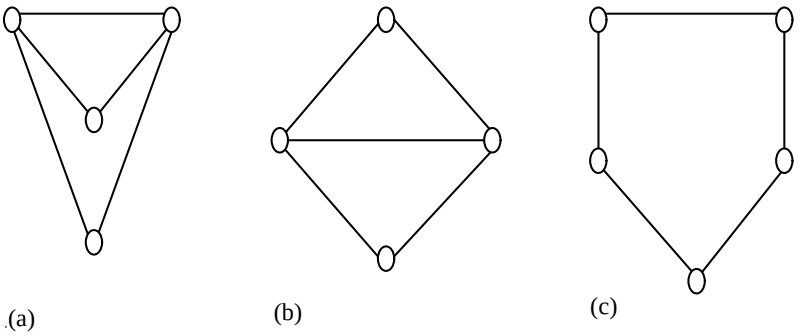

Figure 7.16: *Matching relational structures: (a) and (b) match assuming nodes and relations of the same type, (c) does not match either (a) or (b).*

7.4.3 Syntactic classifier learning, grammar inference

To model a language of any class of patterns as closely as possible, the grammar rules should be extracted from a training set of example words. This process of grammar construction from examples is known as **grammar inference**, the essence of which can be seen in Figure 7.17.

The source of words generates finite example words consisting of the terminal symbols. Assume that these examples include structural features that should be represented by a grammar G which will serve as a model of this source. All the words that can be generated by the source are included in the language $L(G)$ and the words that cannot be generated by the source represent a residuum of this set $L^C(G)$. This information enters the inference algorithm whose goal is to find and describe the grammar G. Words that are included in the language $L(G)$ can be acquired simply from the source of examples. However, the elements of $L^C(G)$ must be presented by a teacher that has additional information about the grammar properties [Gonzalez and Thomason 74, Barrero 91, Schalkoff 92].

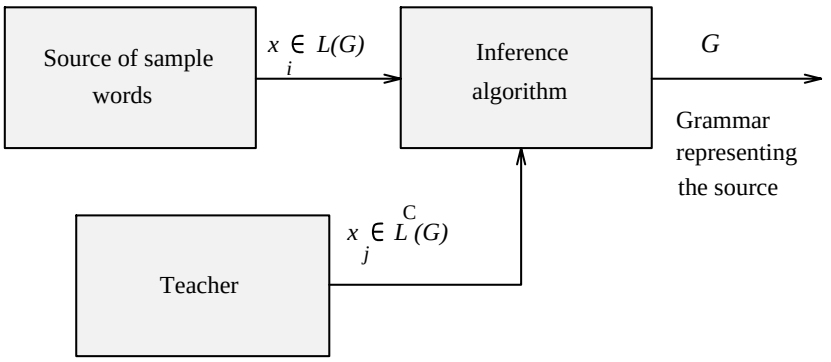

Figure 7.17: *Grammar inference.*

Note that the number of example words generated by the source is finite and it is therefore not sufficient to define the possibly infinite language $L(G)$ unambiguously. Any finite set of examples can be represented by an infinite set of languages, making it impossible to identify unambiguously the grammar that generated the examples. Grammar inference is expected to construct the grammar that describes the training set of examples, plus another set of words that in some sense have the same structure as the examples.

The inference methods can be divided into two groups, based on **enumeration** and **induction**. Enumeration detects the grammar G from the finite set M of grammars that can generate the entire training set of examples or its main part. The difficulty is in the definition of the set M of grammars and in the procedure to search for the grammar G. Induction based methods start with the analysis of words from the training set; the substitution rules are derived from these examples using patterns of similar words.

There is no general method for grammar inference that constructs a grammar from a training set. Existing methods can be used to infer regular and context-free grammars, and may furthermore be successful in some other special cases. Even if simple grammars are considered, the inferred grammar usually generates a language that is much larger than the minimum language that can be used for appropriate representation of the class. This property of grammar inference is extremely unsuitable for syntactic analysis because of the computational complexity. Therefore, the main role in syntactic analyzer learning is still left to a human analyst, and the grammar construction is based on heuristics, intuition, experience, and prior information about the problem.

If the recognition is based on sample relational structures, the main problem is in its automated construction. The conventional method for the sample relational structure con-

struction is described in [Winston 75] where the relational descriptions of objects from the training set are used. The training set consists of examples and counter-examples. The counter-examples should be chosen to have only one typical difference in comparison with a pattern that is a representative of the class.

7.5 Recognition as graph matching

The following section is devoted to recognition methods based on graph comparisons. Graphs with evaluated nodes and evaluated arcs will be considered as they appear in the image description using relational structures. The aim is to decide whether the reality represented by an image matches prior knowledge about the image incorporated into the graphical models. An example of a typical graph matching task is in Figure 7.18.

If this task is presented as an object recognition problem, the object graph must match the object model graph exactly. If the problem is to find an object (represented by a model graph) in the graphical representation of the image, the model must match a subgraph in the image graph exactly. An exact match of graphs is called graph **isomorphism** – for example, the graphs in Figure 7.18 are isomorphic.

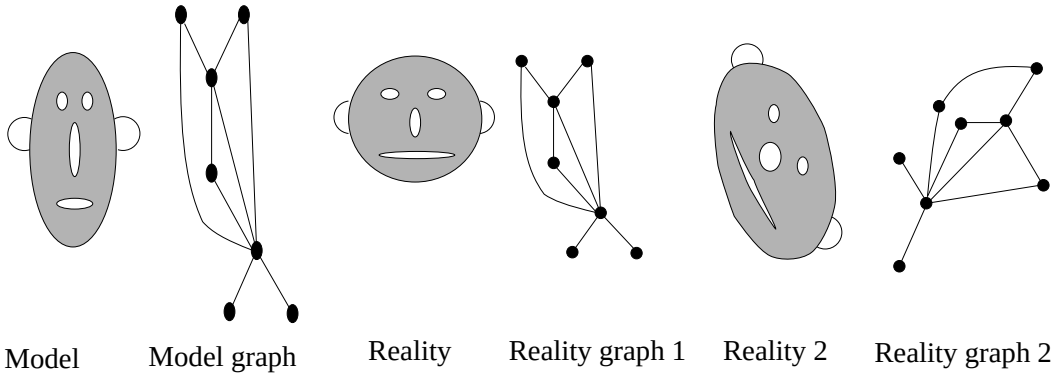

Figure 7.18: *Graph matching problem.*

Graph isomorphism and subgraph isomorphism evaluation is a classical problem in graph theory and is important from both practical and theoretical points of view. Graph theory is covered in [Harary 69, Berge 76, Mohring 91, Nagl 90, Tucker 95], and graph theoretical algorithms can be studied in [Even 79, Lau 89, McHugh 90]. The problem is actually more complex in reality, since the requirement of an exact match is very often too strict in recognition problems.

Because of imprecise object descriptions, image noise, overlapping objects, lighting conditions, etc., the object graph usually does not match the model graph exactly. Graph matching is a difficult problem and evaluation of graph **similarity** is not any easier. An important problem in evaluation of graph similarity is to design a metric which determines how similar two graphs are.

7.5.1 Isomorphism of graphs and subgraphs

Regardless of whether graph or subgraph isomorphism is required, the problems can be divided into three main classes [Harary 69, Berge 76, Ballard and Brown 82]

1. **Graph isomorphism**. Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, find a *one-to-one* and *onto* mapping (an isomorphism) f between V_1 and V_2 such that for each edge of E_1 connecting any pair of nodes $v, v' \in V_1$, there is an edge of E_2 connecting $f(v)$ and $f(v')$; further, if $f(v)$ and $f(v')$ are connected by an edge in G_2, v and v' are connected in G_1.

2. **Subgraph isomorphism**. Find an isomorphism between a graph G_1 and subgraphs of another graph G_2. This problem is more difficult than the previous one.

3. **Double subgraph isomorphism**. Find all isomorphisms between subgraphs of a graph G_1 and subgraphs of another graph G_2. This problem is of the same order of difficulty as number 2.

The subgraph isomorphism and the double subgraph isomorphism problems are NP-complete, meaning that using known algorithms the solution can only be found in time proportional to an exponential function of the length of the input. It is still not known whether the graph isomorphism problem is NP-complete (see [Even 79, Sedgewick 84, Blum and Rivest 88] for details and examples). Despite extensive effort, there is neither an algorithm that can test for graph isomorphism in polynomial time, nor is there a proof that such an algorithm cannot exist. However, non-deterministic algorithms for graph isomorphism that use heuristics and look for suboptimal solutions give a solution in polynomial time in both graph and subgraph isomorphism testing.

Isomorphism testing is computationally expensive for both non-evaluated and evaluated graphs. Evaluated graphs are more common in recognition and image understanding, where nodes are evaluated by properties of regions they represent, and graph arcs are evaluated by relations between nodes they connect (see Section 7.1).

The evaluations can simplify the isomorphism testing. More precisely, the evaluation may make disproof of isomorphism easier. Isomorphic evaluated graphs have the same number of nodes with the same evaluation, and the same number of arcs with the same evaluation. An isomorphism test of two evaluated graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ can be based on partitioning the node sets V_1 and V_2 in a consistent manner looking for inconsistencies in the resulting set partitions. The goal of the partitioning is to achieve a one-to-one correspondence between nodes from sets V_1 and V_2 for all nodes of the graphs G_1 and G_2. The algorithm consists of repeated node set partitioning steps, and the necessary conditions of isomorphism are tested after each step (the same number of nodes of equivalent properties in corresponding sets of both graphs). The node set partitioning may, for example, be based on the following properties:

- Node attributes (evaluations)
- The number of adjacent nodes (connectivity)
- The number of edges of a node (node degree)

- Types of edges of a node
- The number of edges leading from a node back to itself (node order)
- The attributes of adjacent nodes

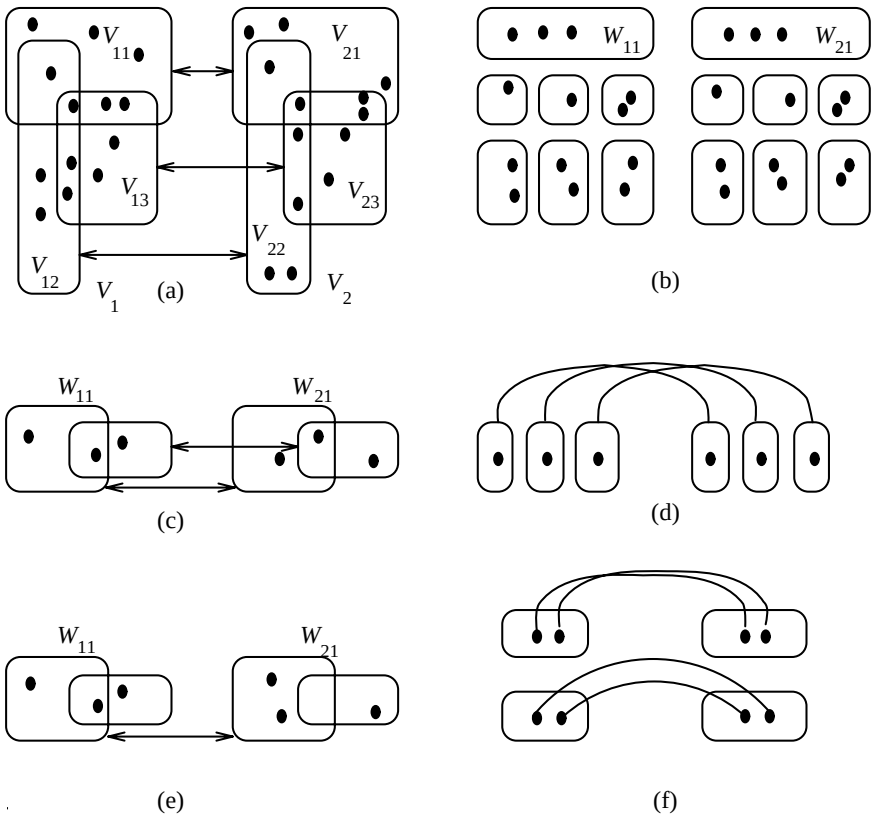

Figure 7.19: *Graph isomorphism: (a) Testing cardinality in corresponding subsets, (b) partitioning node subsets, (c) generating new subsets, (d) subset isomorphism found, (e) graph isomorphism disproof, (f) situation when arbitrary search is necessary.*

After the new subsets are generated based on one of the listed criteria, the cardinality of corresponding subsets of nodes in graphs G_1 and G_2 are tested, see Figure 7.19a. Obviously, if v_{1i} is in several subsets V_{1j} then the corresponding node v_{2i} must also be in the corresponding subsets V_{2j}, or the isomorphism is disproved

$$v_{2i} \in \bigcap_{j|v_{1i} \in V_{1j}} V_{2j} \tag{7.52}$$

If all the generated subsets satisfy the necessary conditions of isomorphism in step i, the subsets are split into new sets of nodes W_{1n}, W_{2n} (Figure 7.19b)

$$\begin{aligned} W_{1i} \bigcap W_{1j} &= \emptyset \quad \text{for } i \neq j \\ W_{2i} \bigcap W_{2j} &= \emptyset \quad \text{for } i \neq j \end{aligned} \tag{7.53}$$

Clearly, if $V_{1j} = V_{2j}$ and if $v_{1i} \notin V_{1k}$ then $v_{2i} \in V_{2k}^C$, where V^C is the set complement. Therefore, by equation (7.52), corresponding elements v_{1i}, v_{2i} of W_{1n}, W_{2n} must satisfy [Niemann 90]

$$v_{2i} \in \{ \bigcap_{\{j|v_{1i} \in W_{1j}\}} W_{2j} \} \cap \{ \bigcap_{\{k|v_{1i} \notin W_{1k} \wedge W_{1k} = W_{2k}\}} W_{2k}^C \} \tag{7.54}$$

The cardinality of all the corresponding sets W_{1n}, W_{2n} is tested to disprove the graph isomorphism.

The same process is repeated in the following steps applying different criteria for graph node subset generation. Note that the new subsets are generated independently in W_{1i}, W_{2i} (Figure 7.19c).

The process is repeated unless one of three cases occurs:

1. The set partitioning reaches the stage when all the corresponding sets W_{1i}, W_{2i} contain one node each. The isomorphism is found (Figure 7.19d).

2. The cardinality condition is not satisfied in at least one of the corresponding subsets. The isomorphism is disproved (Figure 7.19e).

3. No more new subsets can be generated before one of the previous cases occurs. In that situation, either the node set partitioning criteria are not sufficient to establish an isomorphism, or more than one isomorphism is possible. If this is the case, the systematic arbitrary assignment of nodes that have more than one possible corresponding node and cardinality testing after each assignment may provide the solution (Figure 7.19f).

The last part of the process based on systematic assignment of possibly corresponding nodes and isomorphism testing after each assignment may be based on backtracking principles. Note that the backtracking approach can be used from the very beginning of the isomorphism testing, however it is more efficient to start the test using all the available prior information about the matched graphs. The backtracking process is applied if more than one potential correspondence between nodes is encountered. Backtracking tests for directed graph isomorphism and a recursive algorithm is given in [Ballard and Brown 82] together with accompanying hints for improving the efficiency of backtrack searches [Bittner and Reingold 75, Haralick and Elliott 79, Nilsson 82]. The process presented above, graph isomorphism testing, is summarized in the following algorithm

Algorithm 7.8: Graph isomorphism

1. Take two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$.

2. Use a node property criterion to generate subsets V_{1i}, V_{2i} of the node sets V_1 and V_2. Test whether the cardinality conditions hold for corresponding subsets. If not, the isomorphism is disproved.

3. Partition the subsets V_{1i}, V_{2i} into subsets W_{1j}, W_{2j} satisfying the conditions given in equation (7.53) (no two subsets W_{1j} or W_{2j} contain the same node). Test whether the cardinality conditions hold for all the corresponding subsets W_{1j}, W_{2j}. If not, the isomorphism is disproved.

4. Repeat steps (2) and (3) using another node property criterion in all subsets W_{1j}, W_{2j} generated so far. Stop if one of the three above mentioned situations occurs.

5. Based on the situation that stopped the repetition process, the isomorphism either was proved, disproved, or some additional procedures (like backtracking) must be applied to complete the proof or disproof.

A classic approach to subgraph isomorphism can be found in [Ullmann 76]. A brute force enumeration process is described as a depth-first tree-search algorithm. As a way of improving the efficiency of the search, a refinement procedure is entered after each node is searched in the tree – the procedure reduces the number of node successors which yields a shorter execution time. An alternative approach testing isomorphism of graphs and subgraphs transforms the graph problem into a linear programming problem [Zdrahal 81].

The double subgraph isomorphism problem can be translated into a subgraph isomorphism problem using the **clique** – a complete (totally connected) subgraph – approach. A clique is said to be maximal if no other clique properly includes it. Note that a graph may have more than one maximal clique; however, it is often important to find the largest maximal clique (that with the largest number of elements). (Other definitions exist which consider a clique always to be maximal [Harary 69].)

The search for the maximal clique is a well-known problem in graph theory. An example algorithm for finding all cliques of an undirected graph can be found in [Bron and Kerbosch 73]. The maximal clique $G_{clique} = (V_{clique}, E_{clique})$ of the graph $G = (V, E)$ can be found as follows [Niemann 90];

Algorithm 7.9: Maximal clique location

1. Take an arbitrary node $v_j \in V$; construct a subset $V_{clique} = \{v_j\}$.

2. In the set V^C_{clique} search for a node v_k that is connected with all nodes in V_{clique}. Add the node v_k to a set V_{clique}.

3. Repeat step (2) as long as new nodes v_k can be found.

4. If no new node v_k can be found, V_{clique} represents the node set of the maximal clique subgraph G_{clique} (the maximal clique that contains the node v_j).

To find the largest maximal clique, an additional maximizing search is necessary. Other clique-finding algorithms are discussed in [Ballard and Brown 82, Yang et al. 89].

The search for isomorphism of two subgraphs (the double subgraph isomorphism) is transformed to a clique search using the **assignment graph** [Ambler 75]. A pair (v_1, v_2), $v_1 \in V_1$, $v_2 \in V_2$ is called an **assignment** if the nodes v_1 and v_2 have the same node property descriptions, and two assignments (v_1, v_2) and (v'_1, v'_2) are **compatible** if (in addition) all relations between v_1 and v'_1 also hold for v_2 and v'_2 (graph arcs between v_1,v'_1 and v_2,v'_2 must have the same evaluation, including the no-edge case). The set of assignments defines the set of nodes

V_a of the assignment graph G_a. Two nodes in V_a (two assignments) are connected by an arc in the assignment graph G_a if these two nodes are compatible. The search for the maximum matching subgraphs of graphs G_1 and G_2 is a search for the maximum totally connected subgraph in G_a (the maximum totally compatible subset of assignments).

The maximum totally connected subgraph is a maximal clique and the maximal clique finding algorithm can be applied to solve this problem.

7.5.2 Similarity of graphs

All the approaches mentioned above tested for a perfect match between graphs and/or subgraphs. This cannot be anticipated in real applications, and these algorithms are not able to distinguish between a small mismatch of two very similar graphs and the case when the graphs are not similar at all. Moreover, if graph similarity is tested, the main stress is given to the ability to quantify the similarity. Having three graphs G_1, G_2, G_3 the question as to which two are the most similar is a natural one [Buckley 90].

The similarity of two strings (chains) can be based on the **Levenshtein distance** which is defined as the smallest number of deletions, insertions and substitutions necessary to convert one string into the other. Transformations of string elements can be assigned a specific transition cost to make the computed similarity (distance) more flexible and more sensitive. This principle can be applied to graph similarity as well. The set of feasible transformations of nodes and arcs (insertion, deletion, substitution, relabeling) is defined, and these transformations are accompanied by transition costs. Any sequence of transformations is assigned a combination of single step costs (like the sum of individual costs). The set of transformations that has the minimum cost and transforms one graph to another graph defines a distance between them [Niemann 90, Shapiro and Haralick 80].

Note that similarity can be searched for in hierarchical graph structures. The graphs consist of a number of subgraphs in which isomorphism (or similarity) has already been proved. The next step is to detect, describe and evaluate relations between these subgraphs (Figure 7.20, cf. Figure 7.18).

To explain the principles, a physical analogy of templates and springs [Fischler and Elschlager 73] is usually considered. The templates (subgraphs) are connected by springs (relations between subgraphs). The quality of the match of two graphs relates to the quality of the local fit (in corresponding templates) and to the amount of energy used to stretch the springs to match one graph onto the second (reference) graph. To make the graph similarity measure more flexible, extra costs may be added for missing parts of the graph as well as for some extra ones. The spring energy penalty may be made highly nonlinear better to reflect the descriptive character in particular applications.

7.6 Optimization techniques in recognition

Optimization itself is much more flexible than is usually recognized. Considering image recognition and understanding, the best image representation is sought (the best matching between the image and the model is required, the best image understanding is the goal). Whenever 'the best' is considered, some objective function of *goodness* must be available, implying that

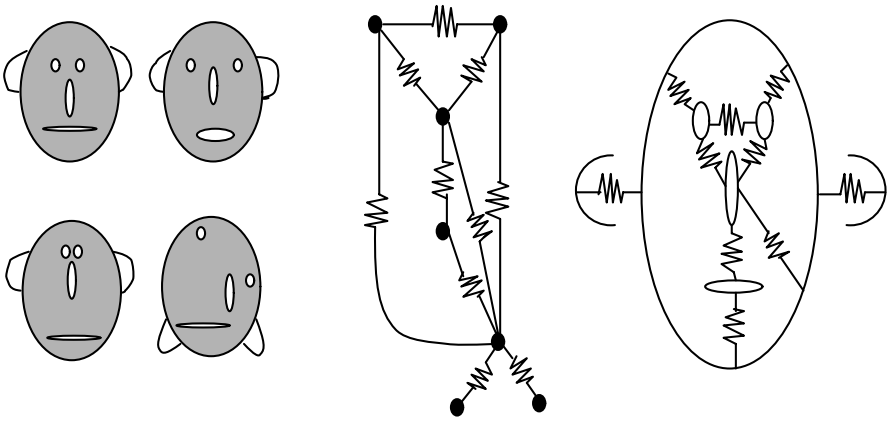

Figure 7.20: *Templates and springs principle: (a) Different objects having the same description graphs, (b),(c) nodes (templates) connected by springs, graph nodes may represent other graphs in finer resolution.*

an optimization technique can be applied which looks for the evaluation function maximum ...for *the best*.

A function optimization problem may be defined as follows: given some finite domain D and a function $f : D \rightarrow R$, R being the set of real numbers, find the *best* value in D under f. Finding the *best* value in D is understood as finding a value $\mathbf{x} \in D$ yielding either the minimum (function minimization) or the maximum (function maximization) of the function f:

$$f_{min}(\mathbf{x}) = \min_{\mathbf{x} \in D} f(\mathbf{x}), \qquad f_{max}(\mathbf{x}) = \max_{\mathbf{x} \in D} f(\mathbf{x}) \tag{7.55}$$

The function f is called the **objective** function. Maximization of the objective function will be considered here as it is typical in image interpretation applications, discussed in Chapter 8. However, optimization methods for seeking maxima and minima are logically equivalent, and optimization techniques can be equally useful if either an objective function maximum or function minimum is required.

It should be noted that no optimization algorithm can guarantee finding a good solution to the problem if the objective function does not reflect the *goodness* of the solution. Therefore, the design of the objective function is a key factor in the performance of any optimization algorithm (similarly, appropriate feature selection is necessary for the success of a classifier).

Most of the conventional approaches to optimization use calculus-based methods which can be compared to climbing a hill (in the case of maximization) – the gradient of the objective function gives the steepest direction to climb. The main limitation of calculus based methods is their local behavior; the search can easily end in a local maximum and the global maximum can be missed (see Figure 7.21).

There are several methods which improve the probability of finding the global maximum; to start the hill-climbing at several points in the search space, to apply enumerative searches like dynamic programming, to apply random searches, etc. Among these possibilities are genetic algorithms and simulated annealing.

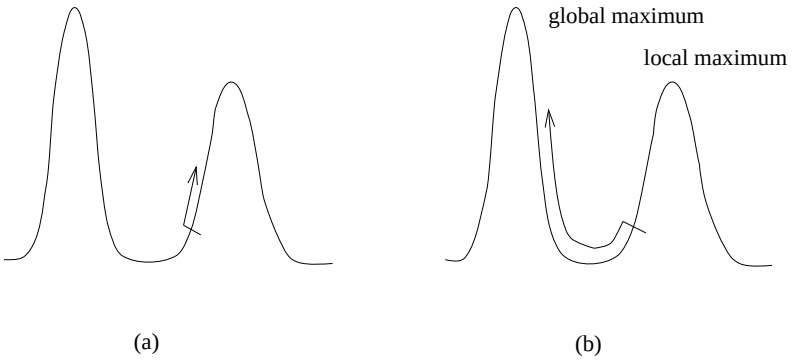

Figure 7.21: *Limitations of hill-climbing methods.*

7.6.1 Genetic algorithms

Genetic algorithms (GA) use natural evolution mechanisms to search for the maximum of an objective function; as with any optimization technique, they can be used in recognition and machine learning.

Genetic algorithms do not guarantee that the global optimum will be found, however empirical results from many applications show that the final solution is usually very close to it. This is very important in image understanding applications as will be seen in the next chapter. There are almost always several consistent (feasible) solutions that are locally optimal in image understanding, or matching, and only one of those possible solutions is the best one represented by the global maximum. The opportunity to find the (near) global optimum is very valuable in these tasks.

Genetic algorithms differ substantially from other optimization methods in the following ways [Goldberg 89];

1. GAs work with a coding of the parameter set, not the parameters themselves. Genetic algorithms require the natural parameter set of the optimization problem to be coded as a finite length string over some finite alphabet. This implies that any optimization problem representation must be transformed to a string representation; binary strings are often used (the alphabet consists of the symbols 0 and 1 only). The design of the problem representation as a string is an important part of the GA method.

2. GAs search from a population of points, not a single point. The population of solutions that is processed in each step is large, meaning that the search for the optimum is simultaneously driven from many places in the search space. This gives a better chance of finding the global optimum.

3. GAs use the objective function directly, not derivatives or other auxiliary knowledge. The search for new, better solutions depends on the values of the evaluation function only. Note that as in other recognition methods, the GAs find the (near) global optimum of the evaluation function but there is no guarantee at all that the evaluation function is relevant to the problem. The evaluation function describes the *goodness* of the particular string. The value of the evaluation function is called **fitness** in GAs.

4. GAs use probabilistic transition rules, not deterministic rules. Rules of transition from the current population of strings to a new and better population of strings are based on the natural idea of supporting good strings with higher fitness and removing poor strings with lower fitness. This is the key idea of genetic algorithms. The best strings representing the best solutions are allowed to survive the evolution process with a higher probability.

   The survival of the fittest and the death of the poor code strings is achieved by applying three basic operations: **reproduction, crossover**, and **mutation**.

The population of strings represents all the strings that are being processed in the current step of the GA. The sequence of reproduction, crossover, and mutation generates a new population of strings from the previous population.

Reproduction

The reproduction operator is responsible for the survival of the fittest and for the death of others based on a probabilistic treatment.

The reproduction mechanism copies strings with highest fitness into the next generation of strings. The selection process is usually probabilistic, the probability that a string is reproduced into the new population being given by its relative fitness in the current population – this is their mechanism of survival. The lower the fitness of the string the lower the chances for survival. This process results in a set of strings where some strings of higher fitness may be copied more than once into the next population. The total number of strings in the population usually remains unchanged, and the average fitness of the new generation is higher than it was before.

Crossover

There are many variations on the crossover. The basic idea is to mate the newly reproduced strings at random, randomly choosing a position for the border of each pair of strings, and to produce new strings by swapping all characters between the beginning of the string pairs and the border position, see Figure 7.22.

Not all newly reproduced strings are subject to the crossover. There is a probability parameter representing the number of pairs which will be processed by crossover; also, it may be performed such that the best reproduced strings are kept in an unchanged form.

The crossover operation together with reproduction represent the main power of GAs. However, there is one more idea in the crossover operation; blocks of characters can be detected in the strings that have locally correct structure even if the string as a whole does not represent a good solution. These blocks of characters in strings are called **schemata**. Schemata are substrings that can represent building blocks of the string, and can be understood as the local pattern of characters. Clearly, if schemata can be manipulated as locally correct blocks, the optimal solution can be located faster than if all the characters are handled independently. In every generation of n strings, about n^3 schemata are processed. This is called the **implicit parallelism** of genetic algorithms [Goldberg 89].

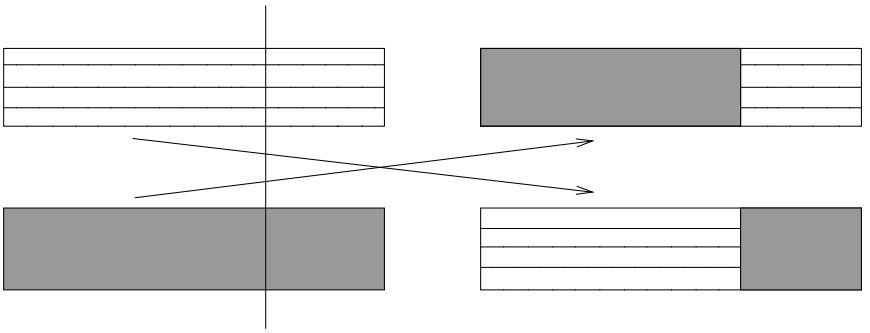

Figure 7.22: *Principle of crossover. Two strings before (left) and after the crossover (right).*

Mutation

The mutation operator plays only a secondary role in GAs. Its principle is randomly to change one character of some string of the population from time to time – it might, for example, take place approximately once per thousand bit transfers (i.e. one bit mutation per one thousand bits transferred from generation to generation). The main reason for mutation is the fact that some local configurations of characters in strings of the population can be totally lost as a result of reproduction and crossover operations. Mutation protects GAs against such irrecoverable loss of good solution features.

Population convergence in GAs is a serious question. For practical purposes this question becomes one of when to stop generating the new string populations. A common and practically proven criterion recommends that the population generating process be stopped when the maximum achieved fitness in the population has not improved substantially through several previous generations.

We have not yet discussed how to create the starting population, which usually consists of a large number of strings, the number depending on the application. The starting population can be generated at random, assuming the alphabet of characters and the desired length of strings are known. Nevertheless, as always, if there is some prior knowledge about the solution available (i.e. the probable local patterns of characters, the probable percentages of characters in strings, etc.), then it is advantageous to use this information to make the starting population as fit as possible. The better the starting population, the easier and faster the search for the optimum.

The simplified version of the genetic algorithm is as follows:

Algorithm 7.10: Genetic algorithm

1. Create a starting population of code strings, and find the value of their objective functions.

2. Probabilistically reproduce high fitness strings in the new population, remove poor fitness strings (reproduction).

3. Construct new strings combining reproduced code strings from the previous population (crossover).

4. From time to time change one character of some string at random (mutation).

5. Order code strings of the current population according to the value of their objective function (fitness).

6. If the maximum achieved string fitness does not increase over several steps, stop. The desired optimum is represented by the current string of maximum fitness. Otherwise, repeat the sequence of steps starting at step 2.

See Section 8.6.2 for an example of the algorithm. A more detailed and precise description of genetic algorithms can be found in [Goldberg 89, Rawlins 91, Michalewicz 94, Adeli and Hung 95, Chambers 95, Mitchell 96]. Many examples and descriptions of related techniques are included there as well, such as knowledge implementation into mutation and crossover, GA learning systems, hybrid techniques that combine good properties of conventional hill-climbing searches and GAs, etc.

7.6.2 Simulated annealing

Simulated annealing [Kirkpatrick et al. 83, Cerny 85] represents another group of robust optimization methods. Similarly to genetic algorithms, simulated annealing searches for a minimum of an objective function (cost function) that represents the goodness of some complex system. Searching for minima is considered in this section because it simplifies energy-related correspondences with the natural behavior of matter. Simulated annealing may be suitable for NP-complete optimization problems; simulated annealing does not guarantee that the global optimum is found, but the solution is usually near-optimal.

Cerny [Cerny 85] often uses the following example to explain the principle of simulated annealing optimization. Imagine a sugar bowl freshly filled with cube sugar. Usually, some cubes do not fit in the sugar bowl and the lid cannot be closed. From experience, everybody knows that shaking the sugar bowl will result in better placement of the cubes inside the bowl and the lid will close properly. In other words, considering the number of cubes that can be inside the bowl as an evaluation function, shaking the bowl results in a near-minimal solution (considering sugar space requirements). The degree of shaking is a parameter of this optimization process and corresponds to the heating and cooling process as described below.

Simulated annealing combines two basic optimization principles, **divide and conquer** and **iterative improvement** (hill-climbing). This combination avoids getting stuck in local optima. A strong connection between statistical mechanics or thermodynamics, and multivariate or combinatorial optimization is the basis for annealing optimization.

In statistical mechanics, only the most probable change of state of a system in thermal equilibrium at a given temperature is observed in experiments; each configuration (state) defined by the set of atomic positions $\{x_i\}$ of the system is weighted by its Boltzmann constant probability factor

$$\exp\left(\frac{-E(\{x_i\})}{k_B T}\right) \tag{7.56}$$

where $E(\{x_i\})$ is the energy of the state, k_B is the Boltzmann constant, and T is the temperature [Kirkpatrick et al. 83].

One of the main characteristics of the Boltzmann density is that at high temperature each state has an almost equal chance of becoming the new state, but at low temperature only states with low energies have a high probability of becoming current. The optimization can be compared with the ability of matter to form a crystalline structure that represents an energy minimum if the matter is melted and cooled down slowly. This minimum can be considered the optimization minimum for the energy function playing the role of the objective function. The crystalization process depends on the cooling speed of the molten liquid; if the cooling is too fast, the crystal includes many local defects and the global energy minimum is not reached.

Simulated annealing consists of downhill iteration steps combined with controlled uphill steps that make it possible to escape from local minima (see Figure 7.23).

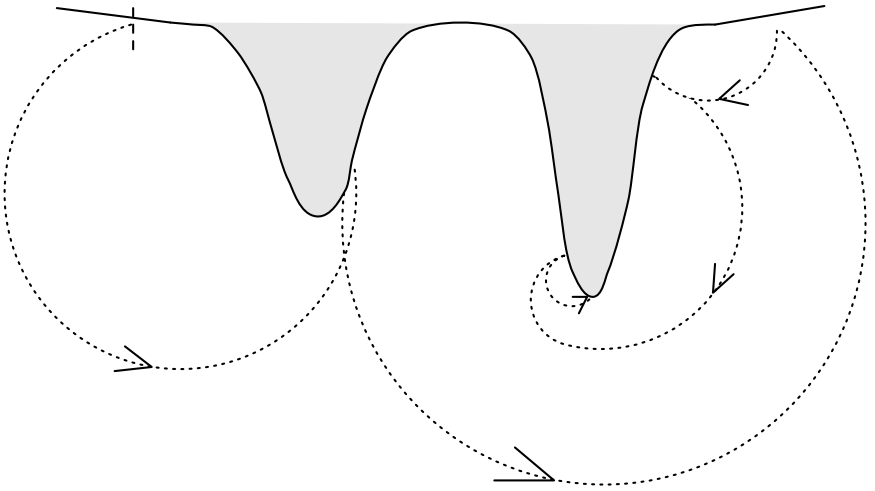

Figure 7.23: *Uphill steps make it possible to get out of local minima; the dotted line shows a possible convergence route.*

The physical model of the process starts with heating the matter until it melts; then the resulting liquid is cooled down slowly to keep the quasi-equilibrium. The cooling algorithm [Metropolis et al. 53] consists of repeated random displacements (state changes) of atoms in the matter, and the energy change ΔE is evaluated after each state change. If $\Delta E \leq 0$ (lower energy), the state change is accepted, and the new state is used as the starting state of the next step. If $\Delta E > 0$, the state is accepted with probability

$$P(\Delta E) = \exp(\frac{-\Delta E}{k_B T}) \tag{7.57}$$

To apply this physical model to an optimization problem, the temperature parameter T must be decreased in a controlled manner during optimization. The random part of the algorithm can be implemented by generating random numbers uniformly distributed in the interval (0,1); one such random number is selected and compared with $P(\Delta E)$.

Algorithm 7.11: Simulated annealing optimization

1. Let $\mathbf{x}$ be a vector of optimization parameters; compute the value of the objective function $J(\mathbf{x})$.

2. Repeat steps 3 and 4 $n(T)$ times.

3. Perturb the parameter vector $\mathbf{x}$ slightly, creating the vector $\mathbf{x}_{new}$, and compute the new value of the optimization function $J(\mathbf{x}_{new})$.

4. Generate a random number $r \in (0,1)$, from a uniform distribution in the interval (0,1). If
$$r < \exp\left(\frac{-(J(\mathbf{x}_{new}) - J(\mathbf{x}))}{k_B T}\right) \tag{7.58}$$
then assign $\mathbf{x} = \mathbf{x}_{new}$ and $J(\mathbf{x}) = J(\mathbf{x}_{new})$.

5. Repeat step 2 to 4 until a convergence criterion is met.

6. The parameter vector $\mathbf{x}$ now represents the solution of the optimization problem.

Note that nothing is known beforehand about how many steps n, what perturbations to the parameter (state changes), what choice of temperatures T, and what speed of cooling down should be applied to achieve the best (or even good) results, although some general guidelines exist and appropriate parameters can be found for particular problems. It is only known that the annealing process must continue long enough to reach a steady state for each temperature. Remember the sugar bowl example, the shaking is much stronger at the beginning and gradually decreases for the best results.

The sequence of temperatures and the number n of steps necessary to achieve equilibrium in each temperature is called the **annealing schedule**. Large values of n and small decrements of T yield low final values of the optimization function (the solution is close to the global minimum) but require long computation time. A small number of repetitions n and large decrements in T proceed faster but the results may not be close to the global minimum. The values T and n must be chosen to give a solution close to the minimum without wasting too much computation time. There is no known practically applicable way to design an optimal annealing schedule.

The annealing algorithm is easy to implement. Annealing has been applied to many optimization problems including pattern recognition, graph partitioning, and many others and has been demonstrated to be of great value (although examples of optimization problems exist in which it performs less well than standard algorithms and other heuristics). In the computer vision area, applications include stereo correspondence [Barnard 87], boundary detection [Geman et al. 90], texture segmentation [Bouman and Liu 91] and edge detection [Tan et al. 92]. Implementation details and annealing algorithm properties together with an extensive list of references can be found in [Aarts and van Laarhoven 86, van Laarhoven and Aarts 87, van Laarhoven 88, Otten and van Ginneken 89, Azencott 92].

7.7 Fuzzy systems

Fuzzy systems are capable of representing diverse, non-exact, uncertain, and inaccurate knowledge or information. They use qualifiers that are very close to the human way of expressing knowledge, like bright, medium dark, dark, etc. Fuzzy systems can represent complex knowledge and even knowledge from contradictory sources. They are based on fuzzy logic, which represents a powerful approach to decision-making [Zadeh 65, Kaufmann 75, Bezdek 81, Kandel 82, Pal and Majumder 86, Zimmermann 87, Pal 91, Zimmermann 91, Zadeh and Kacprzyk 92, Kosko 92, Cox 94, Furuhashi 95, Pedrycz 95, Adeli and Hung 95]. The fundamental principles of fuzzy logic were presented in Section 7.1; here, **fuzzy sets**, **fuzzy membership functions**, and **fuzzy systems** are introduced and fundamental fuzzy reasoning approaches are presented.

7.7.1 Fuzzy sets and fuzzy membership functions

If humans describe objects, they often use imprecise descriptors like *bright, large, rounded, elongated,* etc. For instance, fair-weather clouds may be described as small, medium dark or bright, somewhat rounded regions; thunderstorm clouds may be described as dark or very dark, large regions – people are quite comfortable with such descriptions. However, if the task is to recognize clouds from photographs of the sky automatically using pattern recognition approaches, it becomes obvious that crisp boundaries (discrimination functions) must be drawn that separate the cloud classes. It may be quite arbitrary to make a decision about the boundary location – a decision that a cloud region R_1 characterized by the average gray level g, roundness r and size s represents a thunderstorm cloud while the region R_2 characterized by the average gray level $g + 1$, the same roundness r and size s does not. It may be more appropriate to consider a region R_1 as belonging to the set of fair-weather clouds with some degree of membership and belonging to the set of thunderstorm clouds with another degree of membership. Similarly, another region R_2 might belong to both cloud sets with some other degrees of membership. Fuzzy logic thus facilitates simultaneous membership of regions in different fuzzy sets. Figures 7.24a,b demonstrate the difference between the crisp and fuzzy sets representing the average gray level of the cloud regions.

A **fuzzy set** S in a fuzzy space X is a set of ordered pairs

$$S = \{(x, \mu_S(x)) | x \in X\} \tag{7.59}$$

where $\mu_S(x)$ represents the grade of membership of x in S. The range of the membership function is a subset of non-negative real numbers whose supremum is finite. For convenience, a unit supremum is widely used

$$\sup_{x \in X} \mu_S(x) = 1 \tag{7.60}$$

The fuzzy sets are often denoted solely by its membership function.

The description of DARK regions presented in Figure 7.24b is a classical example of a fuzzy set and illustrates the properties of fuzzy spaces. The **domain** of the fuzzy set is depicted along the x- axis and ranges from black to white (0–255). The degree of membership $\mu(x)$ can be seen along the vertical axis. The membership is between zero and one, zero representing no membership and one representing the complete membership. Thus, a white region with

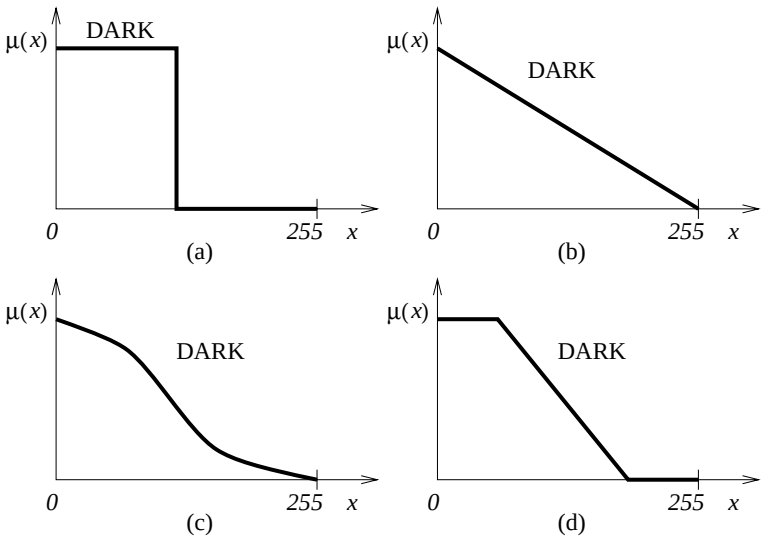

Figure 7.24: *Crisp and fuzzy sets representing cloud regions of the same size and roundness, varying average gray level g. (a) Crisp set showing the Boolean nature of the* DARK *set, (b) fuzzy set* DARK*, (c) another possible membership function associated with the fuzzy set* DARK*, (d) yet another possible membership function.*

average gray level of 255 has zero membership in the DARK fuzzy set while the black region (average gray level = 0) has complete membership in the DARK fuzzy set. As shown in Figure 7.24b, the membership function may be linear, however a variety of other curves may also be used (Figure 7.24c,d).

Consider average gray levels of fair-weather and thunderstorm clouds; Figure 7.25 shows possible membership functions associated with the fuzzy sets DARK, MEDIUM_DARK, BRIGHT. As the Figure shows, a region with a specific average gray level g may simultaneously belong to several fuzzy sets. Thus, the memberships $\mu_{DARK}(g)$, $\mu_{MEDIUM\_DARK}(g)$, $\mu_{BRIGHT}(g)$ represent the fuzziness of the description since they assess the degree of certainty about the membership of the region in the particular fuzzy set. The maximum membership value associated with any fuzzy set is called the **height** of the fuzzy set.

In fuzzy system design, normalized versions of membership functions are used. The **minimum normal form** requires at least one element of the fuzzy set domain to have a membership value of one, and the **maximum normal form** is such minimum normal forms for which at least one element of the domain has a membership value of zero.

In fuzzy reasoning systems, fuzzy membership functions are usually generated in the minimum normal form; a long list of possible fuzzy membership functions (linear, sigmoid, beta curve, triangular curve, trapezoidal curve, shouldered curve, arbitrary curve, etc.), fuzzy numbers, fuzzy quantities, and fuzzy counts can be found together with their definitions in [Cox 94].

Shape of fuzzy membership functions can be modified using **fuzzy set hedges**. Hedges may intensify, dilute, form a complement, narrowly or broadly approximate, etc. the membership of the fuzzy set elements. Zero or more hedges and the associated fuzzy set constitute a single semantic entity called a **linguistic variable**. Suppose $\mu_{DARK}(x)$ represents the

membership function of the fuzzy set DARK; then the intensified fuzzy set VERY_DARK will have the membership function (Figure 7.26a)

$$\mu_{VERY\_DARK}(x) = \mu^2_{DARK}(x) \tag{7.61}$$

Similarly, a diluting hedge creating a fuzzy set SOMEWHAT_DARK will have a membership function (Figure 7.26b)

$$\mu_{SOMEWHAT\_DARK}(x) = \sqrt{\mu_{DARK}(x)} \tag{7.62}$$

Multiple hedges can be applied to a single fuzzy membership function and a fuzzy set VERY_VERY_DARK can be created as

$$\mu_{VERY\_VERY\_DARK}(x) = \mu^2_{DARK}(x) \cdot \mu^2_{DARK}(x) = \mu^4_{DARK}(x) \tag{7.63}$$

There are no theoretically solid reasons for these hedge formulae, but they have the merit of success in practice – they simply 'seem to work' [Cox 94].

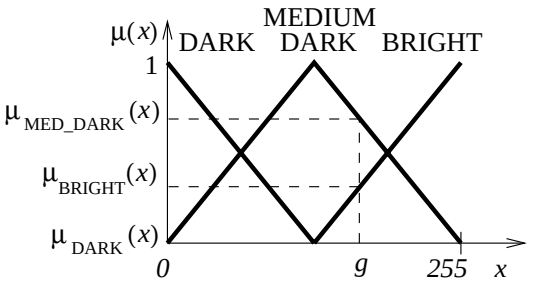

Figure 7.25: *Membership functions associated with fuzzy sets* DARK, MEDIUM_DARK, *and* BRIGHT. *Note, that several membership values may be associated with a specific average gray level g.*

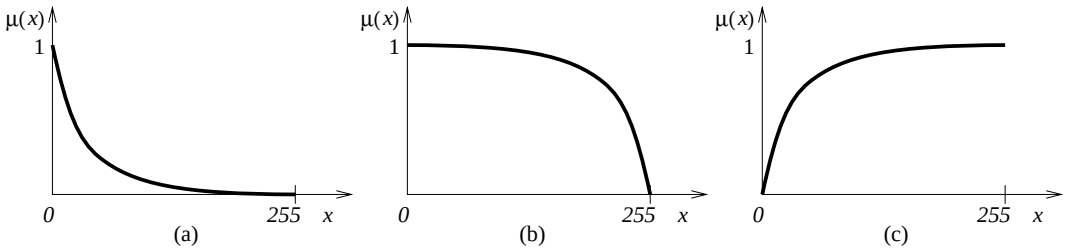

Figure 7.26: *Fuzzy set hedges. Fuzzy set* DARK *is shown in Figure 7.24b. (a) Fuzzy set* VERY_DARK, *(b) fuzzy set* SOMEWHAT_DARK, *(c) fuzzy set* NOT_VERY_DARK.

7.7.2 Fuzzy set operators

Rarely can a recognition problem be solved using a single fuzzy set and the associated single membership function. Therefore, tools must be made available that combine various fuzzy

sets and allow to determine membership functions of such combinations. In conventional logic, membership functions are either zero or one (Figure 7.24) and for any class set S, a rule of noncontradiction holds: An intersection of a set S with its complement S^c is an empty set.

$$S \cap S^c = \emptyset \tag{7.64}$$

Clearly, this rule does not hold in fuzzy logic since domain elements may simultaneously belong to fuzzy sets and their complements; there are three basic **Zadeh operators** on fuzzy sets: **fuzzy intersection**, **fuzzy union**, and **fuzzy complement**. Let $\mu_A(x)$ and $\mu_B(y)$ be two membership functions associated with two fuzzy sets A and B with domains X and Y. Then, the intersection, union, and complement are pointwise defined for all $x \in X, y \in Y$ (note that other definitions also exist)

$$\begin{aligned} \text{Intersection} \quad A \cap B: \qquad & \mu_{A\cap B}(x,y) = \quad \min(\mu_A(x), \mu_B(y)) \\ \text{Union} \quad A \cup B: \qquad & \mu_{A\cup B}(x,y) = \quad \max(\mu_A(x), \mu_B(y)) \\ \text{Complement} \quad A^c: \qquad & \mu_{A^c}(x) = \quad 1 - \mu_A(x) \end{aligned} \tag{7.65}$$

Note that the fuzzy set operators may be combined with the hedges and new fuzzy sets may be constructed, e.g. a fuzzy set NOT_VERY_DARK would be constructed as NOT (VERY (DARK))

$$\mu_{NOT\_VERY\_DARK}(x) = 1 - \mu^2_{DARK}(x)$$

(see Figure 7.26).

7.7.3 Fuzzy reasoning

In fuzzy reasoning, information carried in individual fuzzy sets is combined to make a decision. The functional relationship determining the degree of membership in related fuzzy membership functions is called **method of composition** (method of implication) and results in the definition of a **fuzzy solution space**. To arrive at the decision, a **defuzzification** (decomposition) process determines a functional relationship between the fuzzy solution space and the decision. Processes of composition and defuzzification form the basis of fuzzy reasoning (Figure 7.27), which is performed in the context of a **fuzzy system model** that consists of control, solution, and working data variables; fuzzy sets; hedges; fuzzy rules; and a control mechanism. Fuzzy models use a series of unconditional and conditional propositions called fuzzy rules. Unconditional fuzzy rules are of the form

$$x \text{ is } A \tag{7.66}$$

and conditional fuzzy rules have the form

$$if \quad x \text{ is } A \quad then \quad w \text{ is } B \tag{7.67}$$

where A and B are linguistic variables and x and w represent scalars from their respective domains. The degree of membership associated with an unconditional fuzzy rule is simply $\mu_A(x)$. Unconditional fuzzy propositions are used either to restrict the solution space or to define a default solution space. Since these rules are unconditional, they are directly applied to the solution space by applying fuzzy set operators.

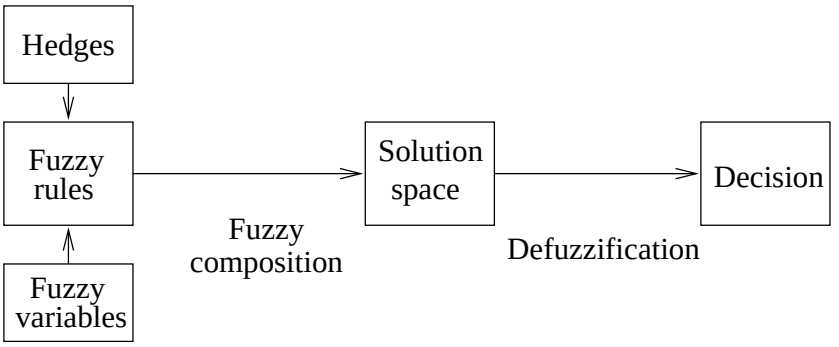

Figure 7.27: *Fuzzy reasoning – composition and defuzzification.*

Considering conditional fuzzy rules, there are several approaches to arrive at the decision. **Monotonic fuzzy reasoning** is the simplest approach that can produce a solution directly without composition and defuzzification. Let again x represent a scalar gray level describing darkness of a cloud, and w the severity of a thunderstorm. The following fuzzy rule may represent our knowledge of thunderstorm severity

$$\text{if } x \text{ is DARK then } w \text{ is SEVERE} \tag{7.68}$$

The algorithm for monotonic fuzzy reasoning is shown in Figure 7.28. Based on determination of the cloud gray level (x=80 in our case), the membership value $\mu_{DARK}(80) = 0.35$ is determined. This value is used to represent the membership value $\mu_{SEVERE}(w) = \mu_{DARK}(x)$ and the decision is made about the expected severity of the thunderstorm; in our case severity $w = 4.8$ on a scale between 0 and 10. This approach may also be applied to complex predicates of the form

$$if \quad (x \text{ is } A) \bullet (y \text{ is } B) \bullet \ldots \bullet (u \text{ is } F) \qquad then \qquad w \text{ is } Z \tag{7.69}$$

where $\bullet$ represents the conjunctive AND or disjunctive OR operations. Fuzzy intersection and union operators can be used to combine the complex predicates; AND corresponds to fuzzy intersection and OR corresponds to fuzzy union. While the monotonic approach shows the fundamental concept of fuzzy reasoning, it can only be used for a monotonic single fuzzy variable controlled by a single fuzzy rule (possibly with a complex predicate). As the complexity of the predicate proposition increases, the validity of the decision tends to decrease.

Fuzzy Composition

Knowledge related to the decision-making process is usually contained in more than one fuzzy rule. A large number of fuzzy rules may take part in the decision making process and all fuzzy rules are fired in parallel during that process. Clearly, not all fuzzy rules contribute equally to the final solution and rules that have no degree of truth in their premises do not contribute to the outcome at all. Several composition mechanisms exist that facilitate rule combination; the most frequently used approach, called the **min–max rule**, will be discussed.

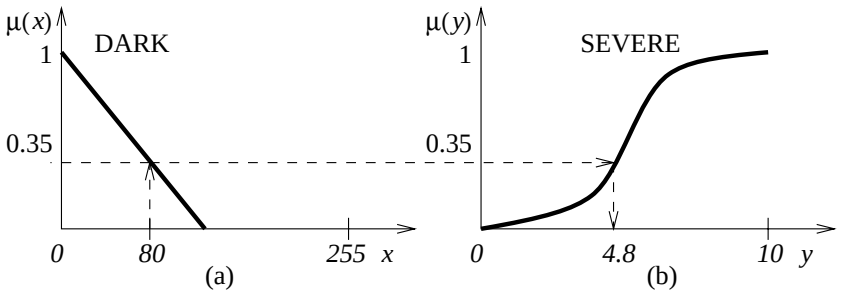

Figure 7.28: *Monotonic fuzzy reasoning based on a single fuzzy rule: If the gray level of the cloud is* DARK *then the thunderstorm will be* SEVERE

In the min–max composition approach, a sequence of minimizations and maximizations is applied. First, the minimum of the predicate truth (**correlation minimum**) $\mu_{A_i}(x)$ is used to restrict the consequent fuzzy membership function $\mu_{B_i}(w)$. Let the rules be in the form specified in equation (7.67), and let i represent the i-th rule. Then, the consequent fuzzy membership functions B_i are updated in a pointwise fashion and the fuzzy membership functions B_i^+ are formed (Figure 7.29).

$$\mu_{B_i^+}(w) = \min(\mu_{B_i}(w), \mu_{A_i}(x)) \tag{7.70}$$

Second, the pointwise maxima of these minimized fuzzy sets form the solution fuzzy membership function.

$$\mu_S(w) = \max_i(\mu_{B_i^+}(w)) \tag{7.71}$$

Figure 7.29 demonstrates the min–max composition process; again, complex predicates may be considered.

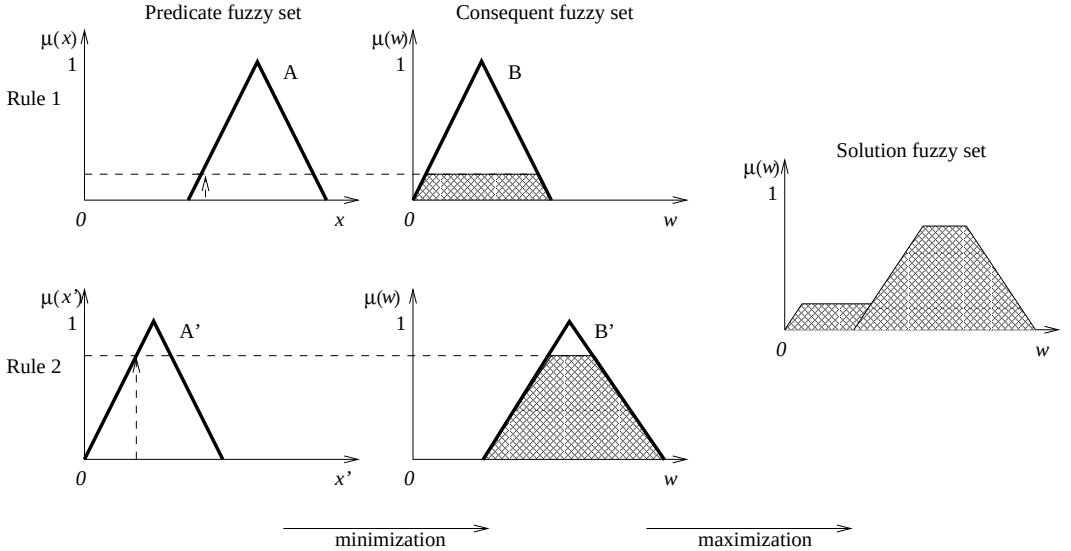

Figure 7.29: *Fuzzy min–max composition using correlation minimum.*

The correlation minimum described above is the most common approach to performing the first step of the min–max composition. An alternative approach called **correlation product** exists that scales the original consequent fuzzy membership functions instead of truncating them. While correlation minimum is computationally less demanding and easier to defuzzify, correlation product represents in many ways a better method of minimization since the original shape of the fuzzy set is retained (Figure 7.30).

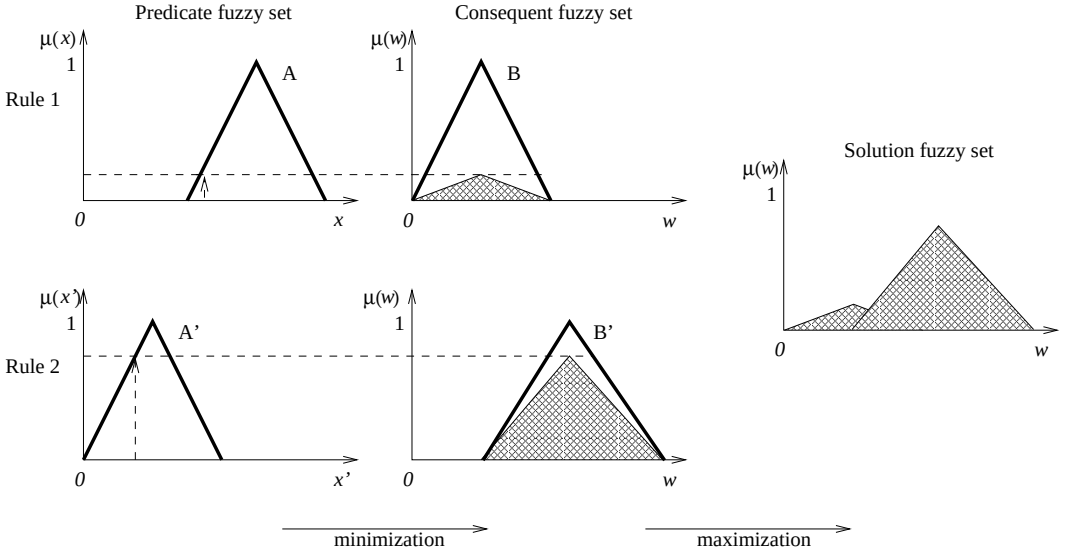

Figure 7.30: *Fuzzy min–max composition using correlation product.*

Defuzzification

Fuzzy composition produces a single solution fuzzy membership function for each solution variable. To find the actual crisp solution that will be used for decision making, it is necessary to find a vector of scalar values (one value for each solution variable) that best represents the information contained in the solution fuzzy sets. This process is performed independently for each solution variable and is called defuzzification. Two defuzzification methods, called **composite moments** and **composite maximum**, are commonly used; many other varieties exist.

Composite moments look for the centroid c of the solution fuzzy membership function – Figure 7.31a shows how the centroid method converts the solution fuzzy membership function into a crisp solution variable c. Composite maximum identifies the domain point with the highest membership value in the solution fuzzy membership function. If this point is ambiguous (on a plateau or if there are two or more equal global maxima), the center of the plateau (or the point halfway between the leftmost and rightmost global maximum) provides the crisp solution c' (Figure 7.31b). The composite moments approach produces a result that is sensitive to all the rules, while solutions determined using the composite maximum method are sensitive to the membership function produced by the single rule that has the highest predicate truth. While composite moments are mostly used in control applications, recognition applications usually use the composite maximum method.

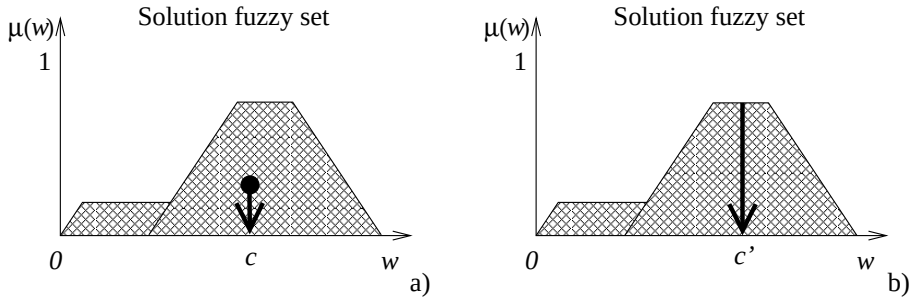

Figure 7.31: *Defuzzification. (a) Composite moments, (b) composite maximum.*

7.7.4 Fuzzy system design and training

Fuzzy system design consists of several main steps that are outlined in the following algorithm.

Algorithm 7.12: Fuzzy system design

1. Design functional and operational characteristics of the system – determine the system inputs, basic processing approaches, and system outputs. In object recognition, the inputs are patterns and the output represents the decision.

2. Define fuzzy sets by decomposing each input and output variable of the fuzzy system into a set of fuzzy membership functions. The number of fuzzy membership functions associated with each variable depends on the task at hand. Typically, an odd number of three to nine fuzzy membership functions are created for each variable. It is recommended that the neighboring fuzzy membership functions overlap by 10–50%. The sum of the membership values of the overlap are recommended to be less than one.

3. Convert problem-specific knowledge into the fuzzy *if–then* rules that represent a fuzzy associative memory. The number of designed rules is related to the number of input variables. For N variables each of which is divided in M fuzzy membership functions, M^N rules are required to cover all possible input combinations.

4. Perform fuzzy composition and defuzzification as described in Section 7.7.3.

5. Using a training set, determine the system's performance. If the fuzzy system's behavior does not meet the requirements, modify the fuzzy set descriptions and/or fuzzy rules and/or the fuzzy composition and/or defuzzification approaches. The speed and success of this fine-tuning step depend on the problem complexity, designer's level of understanding of the problem, and level of designer's experience.

As can be seen from the description of steps (3) and (5) of the previous algorithm, the design of fuzzy rules may be a tedious and time consuming process if the rules are to be designed from human experts as has been typical in most existing applications. Recently, several approaches

have been reported that generate fuzzy *if – then* rules automatically using a training set as the source of knowledge and/or for automated adjusting membership functions of fuzzy sets [Horikawa et al. 92, Simpson 92, Ishibuchi et al. 92, Ishibuchi et al. 95, Abe and Lan 95, Homaifar and McCormick 95]. Some of the approaches use neural networks or genetic algorithms to control the learning process. A genetic algorithm-based method for selecting a small number of significant rules from a training set of examples was given in [Ishibuchi et al. 95], in which the rule-selection problem is formulated as a combinatorial optimization problem and uses genetic algorithm optimization. The optimization process is designed to maximize the classification correctness and minimize the number of fuzzy *if – then* rules. An approach based on iterative splitting of the feature space that was introduced in [Park 96] generates a smaller number of fuzzy rules compared to the approach of Ishibuchi. While Ishibuchi's approach used equal spacing for partitioning of the feature space, Park proposed to use an adaptive grid defined by minimum and maximum values of individual features for each class. This adaptivity is mostly responsible for the more efficient feature space partitioning and is reflected in the smaller number of generated fuzzy rules.

Many applications of fuzzy systems exist in pattern recognition and image processing. In the field of pattern recognition, fuzzy logic has been used for supervised and nonsupervised recognition, sequential learning, fuzzy decision theoretic and syntactic classifiers, feature extraction, etc. [Bezdek 81, Kandel 82, Pal 91, Zadeh and Kacprzyk 92]. In image processing and vision, fuzzy logic was applied to image quality assessment, edge detection, image segmentation, color image segmentation, etc. [Pal 91]. Extensive work has been done in developing fuzzy geometry approaches [Rosenfeld 79a, Rosenfeld 83, Rosenfeld 84a, Rosenfeld 84b, Rosenfeld 85]. Recently, a fuzzy approach to object definition and connectedness and its application to image segmentation was presented in [Dellepiane and Fontana 95, Udupa and Samarasekera 96]; fuzzy connectivity in mathematical morphology is discussed in [Bloch 93]; good performance of fuzzy systems in medical image segmentation and interpretation applications was reported in [Udupa and Samarasekera 96, Park et al. 96].

7.8 Summary

- **Object recognition, pattern recognition**
  - Pattern recognition is used for region and object **classification**, and represents an important building block of complex machine vision processes.
  - No recognition is possible without **knowledge**. Specific knowledge about both the objects being processed and hierarchically higher and more general knowledge about object classes is required.
- **Knowledge representation**
  - Descriptions and features
  - Grammars and languages
  - Predicate logic
  - Production rules
  - Fuzzy logic

  - Semantic nets
  - Frames, scripts

- **Statistical pattern recognition**
  - **Object recognition** is based on assigning classes to objects and the device that does these assignments is called the **classifier**. The number of classes is usually known beforehand, and typically can be derived from the problem specification.
  - The classifier does not decide about the class from the object itself – rather, sensed object properties called **patterns** are used.
  - For statistical pattern recognition, **quantitative** description of objects is characteristic, elementary numerical descriptions – **features** are used. The set of all possible patterns forms the **pattern space** or **feature space**. The **classes** form clusters in the feature space, that can be separated by **discrimination hypersurfaces**.
  - A **statistical classifier** is a device with n inputs and 1 output. Each input is used to enter the information about one of n features measured from an object to be classified. An R-class classifier generates one of R symbols ω_r, the **class identifiers**.
  - Classification parameters are determined from a **training set** of examples during **classifier learning**. Two common learning strategies exist – **probability density estimation** and **direct loss minimization**.
  - Classification methods exist which do not need training sets for learning. **Cluster analysis** methods divide the set of processed patterns into subsets (clusters) based on the mutual similarity of subset elements.

- **Neural nets**
  - Most neural approaches are based on combinations of elementary processors (**neurons**), each of which takes a number of inputs and generates a single output. Associated with each input is a weight, and the output is a function of the weighted sum of inputs. Pattern recognition is one of many application areas of neural networks.
  - **Feedforward** networks are common in pattern recognition problems, their training uses a training set of examples and is often based on the **back-propagation** algorithm.
  - **Self-organizing** networks do not require a training set to cluster the processed patterns.
  - **Hopfield** neural networks do not have designated inputs and outputs, but rather the current configuration represents the state. The Hopfield net acts as an associative memory where the exemplars are stored.

- **Syntactic pattern recognition**

- For syntactic pattern recognition, **qualitative** description of objects is characteristic. The elementary properties of the syntactically described objects are called **primitives**. **Relational structures** are used to describe relations between the object primitives.
- The set of all primitives is called the **alphabet**. The set of all words in the alphabet that can describe objects from one class is named the **description language**. A **grammar** represents a set of rules that must be followed when words of the specific language are constructed from the alphabet.
- Grammar construction usually requires significant human interaction. In simple cases, an automated process of grammar construction from examples called **grammar inference** can be applied.
- The recognition decision of whether or not the word can be generated by a particular grammar is made during **syntactic analysis**.

- **Recognition as graph matching**
  - Matching of a model and an object graph description can be used for recognition. An exact match of graphs is called graph **isomorphism**. Determination of graph isomorphism is computationally expensive.
  - In the real world, the object graph usually does not match the model graph exactly. Graph isomorphism cannot assess the level of mismatch. To identify objects represented by similar graphs, **graph similarity** can be determined.

- **Optimization techniques in recognition**
  - Optimization problems seek minimization or maximization of an **objective function**. Design of the objective function is a key factor in the performance of optimization algorithms.
  - Most conventional approaches to optimization use calculus-based **hill-climbing** methods. For these, the search can easily end in a local maximum and the global maximum can be missed.
  - **Genetic algorithms** use natural evolution mechanisms of the survival of the fittest to search for the maximum of an objective function. Potential solutions are represented as strings. Genetic algorithms search from a population of potential solutions, not a single solution. The sequence of **reproduction**, **crossover**, and **mutation** generates a new population of strings from the previous population. The fittest string represents the final solution.
  - **Simulated annealing** combines two basic optimization principles, **divide and conquer** and **iterative improvement** (hill-climbing). This combination avoids getting stuck in local optima.

- **Fuzzy systems**
  - Fuzzy systems are capable of representing diverse, non-exact, uncertain, and inaccurate knowledge or information. They use qualifiers that are very close to the human way of expressing knowledge.

- Fuzzy reasoning is performed in the context of a **fuzzy system model** that consists of control, solution, and working data variables; fuzzy sets; hedges; fuzzy rules; and a control mechanism.
- **Fuzzy sets** represent properties of fuzzy spaces. **Membership functions** represent the fuzziness of the description and assess the degree of certainty about the membership of an element in the particular fuzzy set. Shape of fuzzy membership functions can be modified using **fuzzy set hedges**. A hedge and its fuzzy set constitute a single semantic entity called a **linguistic variable**.
- **Fuzzy** *if – then* **rules** represent fuzzy associative memory in which knowledge is stored.
- In fuzzy reasoning, information carried in individual fuzzy sets is combined to make a decision. The functional relationship determining the degree of membership in related fuzzy regions is called the **method of composition** and results in definition of a **fuzzy solution space**. To arrive at the decision, **defuzzification** is performed. Processes of composition and defuzzification form the basis of fuzzy reasoning.

7.9 Exercises

Short-answer questions

1. Define the *syntax* and *semantics* of knowledge representation.
2. Describe the following knowledge representations, giving for each one at least one example that is different from examples given in the text.
   (a) descriptions (features)
   (b) grammars
   (c) predicate logic
   (d) production rules
   (e) fuzzy logic
   (f) semantic nets
   (g) frames (scripts)
3. Define the following terms:
   (a) pattern
   (b) class
   (c) classifier
   (d) feature space
4. Describe the main steps of pattern recognition.
5. Define the following terms:
   (a) class identifier
   (b) decision rule

(c) discrimination function

6. Explain the main concepts and derive a mathematical representation of the discrimination functions for:
   (a) a minimum distance classifier
   (b) a minimum error classifier
7. What is a training set? How is it designed? What influences its desired size?
8. Explain why learning should be inductive and sequential.
9. Describe the conceptual differences between supervised and unsupervised learning.
10. Draw schematic diagrams of a feed-forward and Hopfield neural networks. Discuss their major architectural differences.
11. For what is the back-propagation algorithm used? Explain its main steps.
12. What is the reason for including the momentum constant in back-propagation learning?
13. Explain the functionality of Kohonen neural networks. How can they be used for unsupervised pattern recognition?
14. Explain how Hopfield networks can be used for pattern recognition.
15. Compare classification approaches used by statistical pattern recognition and neural networks.
16. Define the following terms:
   (a) primitive
   (b) alphabet
   (c) description language
   (d) grammar
17. Describe the main steps of syntactic pattern recognition.
18. Give a formal definition of a grammar.
19. When are two grammars equivalent?
20. True or false? A regular grammar is a context-free grammar.
21. Name the main approaches to syntactic analysis.
22. What is grammar inference? Give its block diagram.
23. Formally define:
   (a) a graph
   (b) graph isomorphism
   (c) subgraph isomorphism
   (d) double subgraph isomorphism
24. Define Levenshtein distance. Explain its application to assessing string similarity.
25. Explain why hill-climbing optimization approaches may converge to local instead of global optima.
26. Explain the concept and functionality of genetic algorithm optimization. What are the roles of reproduction, crossover, and mutation in genetic algorithms?

27. Explain the concept of optimization based on simulated annealing. What is the annealing schedule?

28. List the advantages and disadvantages of genetic algorithms and simulated annealing compared to optimization approaches based on derivatives.

29. Define the following terms:

    (a) fuzzy set

    (b) fuzzy membership function

    (c) minimum normal form of a fuzzy membership function

    (d) maximum normal form of a fuzzy membership function

    (e) fuzzy system

    (f) domain of a fuzzy set

    (g) hedge

    (h) linguistic variable

30. Use Zadeh's definitions to define formally:

    (a) fuzzy intersection

    (b) fuzzy union

    (c) fuzzy complement

31. Explain fuzzy reasoning based on composition and defuzzification. Draw a block diagram of fuzzy reasoning.

Problems

1. Let a minimum distance classifier be used to recognize 2-dimensional patterns from 3 classes K_1, K_2, K_3. The training set consists of 5 patterns from each class:

$$K_1 := \left\{ \begin{pmatrix} 0 \\ 6 \end{pmatrix}, \begin{pmatrix} 1 \\ 6 \end{pmatrix}, \begin{pmatrix} 2 \\ 6 \end{pmatrix}, \begin{pmatrix} 1 \\ 5 \end{pmatrix}, \begin{pmatrix} 1 \\ 7 \end{pmatrix} \right\}$$

$$K_2 := \left\{ \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \begin{pmatrix} 5 \\ 1 \end{pmatrix}, \begin{pmatrix} 6 \\ 1 \end{pmatrix}, \begin{pmatrix} 5 \\ 0 \end{pmatrix}, \begin{pmatrix} 5 \\ 2 \end{pmatrix} \right\}$$

$$K_3 := \left\{ \begin{pmatrix} 8 \\ 6 \end{pmatrix}, \begin{pmatrix} 9 \\ 6 \end{pmatrix}, \begin{pmatrix} 10 \\ 6 \end{pmatrix}, \begin{pmatrix} 9 \\ 5 \end{pmatrix}, \begin{pmatrix} 9 \\ 7 \end{pmatrix} \right\}$$

   Determine (sketch) the discrimination functions in the 2-dimensional feature space.

2. Let a minimum error classifier be used to recognize 2-dimensional patterns from 2 classes, each having a normal distribution $N(\boldsymbol{\mu}_r, \boldsymbol{\Psi}_r)$:

$$\boldsymbol{\mu}_1 = \begin{pmatrix} 2 \\ 5 \end{pmatrix}; \boldsymbol{\Psi}_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}; \quad \boldsymbol{\mu}_2 = \begin{pmatrix} 4 \\ 3 \end{pmatrix}; \boldsymbol{\Psi}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

   Assume unit loss functions and equal a priori probabilities of classes $P(\omega_1) = P(\omega_2) = 0.5$. Determine (sketch) the discrimination function in the 2-dimensional feature space.

3. Repeat Problem 7.2 with $P(\omega_1) = P$, $P(\omega_2) = 1 - P$. Show, how the discrimination function locations in the 2-dimensional feature space change as a function of P.

4. Repeat Problem 7.2 considering modified parameters of the normal distributions:

$$\boldsymbol{\mu}_1 = \begin{pmatrix} 2 \\ 5 \end{pmatrix}; \boldsymbol{\Psi}_1 = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}; \quad \boldsymbol{\mu}_2 = \begin{pmatrix} 4 \\ 3 \end{pmatrix}; \boldsymbol{\Psi}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}$$

5. Repeat Problem 7.4 with $P(\omega_1) = P$, $P(\omega_2) = (1 - P)$. Show, how the discrimination function locations in the 2-dimensional feature space change as a function of P.

6. Consider the training set specified in Problem 7.1. Assume that the three pattern classes have normal distributions and that a priori probabilities of classes are equal $P(\omega_1) = P(\omega_2) = P(\omega_3) = 1/3$. Determine (sketch) the discrimination functions of the minimum error classifier in the 2-dimensional feature space. Discuss under what circumstances the discrimination functions of a minimum distance classifier are identical to discrimination functions of the minimum error classifier if both were trained using the same training set.

7. Create the following training and testing sets of feature vectors named TRAIN1, TEST1; TRAIN2, TEST2. The training and testing sets will be used in the experiments below.

TRAIN1

ω_i	ω_1	ω_1	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_2	ω_2
x_1	2	4	3	3	4	10	9	8	9	10
x_2	3	2	3	2	3	7	6	6	7	6

TEST1

ω_i	ω_1	ω_1	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_2	ω_2
x_1	3	6	5	5	6	13	12	11	11	13
x_2	5	3	4	3	5	10	8	8	9	8

TRAIN2

ω_i	ω_1	ω_1	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_2	ω_2
x_1	2	6	-2	7	5	-2	-6	2	-4	-5
x_2	400	360	520	-80	180	-200	-200	-400	-600	-400

ω_i	ω_3	ω_3	ω_3	ω_3	ω_3
x_1	-10	-8	-15	-12	-14
x_2	200	140	100	50	300

TEST2

ω_i	ω_1	ω_1	ω_1	ω_1	ω_1	ω_2	ω_2	ω_2	ω_2	ω_2
x_1	4	8	-3	9	6	-1	-4	3	-2	-3
x_2	600	540	780	-120	270	-250	-250	-470	-690	-470

ω_i	ω_3	ω_3	ω_3	ω_3	ω_3
x_1	-15	-13	-6	-17	-16
x_2	230	170	130	80	450

8. Develop a program for training and classification using the minimum distance classifier. Assess classification correctness.

   (a) Train and test using data sets TRAIN1 and TEST1.

   (b) Train and test using data sets TRAIN2 and TEST2.

9. Develop a program for training and classification using the minimum error classifier, considering unit loss functions. Assume the training data have normal distribution. Assess classification correctness.

   (a) Train and test using data sets TRAIN1 and TEST1.

(b) Train and test using data sets TRAIN2 and TEST2.

10. Develop a program for cluster analysis using the k-means approach. Vary the initialization of cluster starting points and explore its influence on clustering results. Vary the number of classes and discuss the results.

    (a) Use a combined data set TRAIN1 and TEST1.

    (b) Use a combined data set TRAIN2 and TEST2.

11. Create a training set and a testing set of feature vectors using the shape description program developed in Problem 6.14 to determine shape feature vectors. Use simple shapes (e.g. triangles, squares, rectangles, circles, etc.) of different sizes. Select up to five discriminative features to form the feature vectors of analyzed shapes. The training as well as the testing sets should consist of at least 10 patterns from each class. The training and testing sets will be used in the experiments below.

12. Apply the program developed in Problem 7.8 to the training and testing sets created in Problem 7.11. Assess classification correctness.

13. Apply the program developed in Problem 7.9 to the training and testing sets created in Problem 7.11. Assume normal distributions, and that you have a sufficient number to determine representative dispersion matrices and mean values from the training set. Assess classification correctness in the testing set. Compare with the performance of the minimum distance classifier used in Problem 7.12.

14. Apply the program developed in Problem 7.10 to the testing set created in Problem 7.11. First, assume that the number of classes is known. Assess clustering correctness and compare it with that of the supervised methods used in Problems 7.12 and 7.13. Then, vary the initialization of cluster starting points and explore the influence on clustering results.

15. Develop a program for back-propagation training and classification using a three-layer feed-forward neural network. Train and test using artificial data from a two-dimensional feature space representing patterns from at least three separable classes.

16. Apply the program developed in Problem 7.15 to the testing set created in Problem 7.11. Assess classification correctness and compare it with that of the statistical classification methods assessed in Problems 7.12 and 7.13.

17. Choice of width of layers is often a problem. Repeat Problems 7.15 and 7.16, paying attention to the size of the hidden layer. Draw some conclusions about network performance and training time as the size of this layer varies.

18. Implement Algorithm 7.5. For some datasets devised by you, or extracted from some known application, run the algorithm. Compare its performance for different sizes and topologies of output layer, and various choices of parameters.

19. Implement a Hopfield network. Train it on digitised patterns of the digits 0 – 9; study its performance at pattern recall (e.g., of noisy examples of digits) for various resolutions of the patterns.

20. Design a grammar G that can generate a language $L(G)$ of equilateral triangles of any size; primitives with 0^o, 60^o, and 120^o orientation form the set of terminal symbols $V_t = \{a, b, c\}$.

21. Design three different grammars producing the language $L(G) = \{ab^n\}$ for $n = 1, 2, \ldots$

22. Design a grammar that generates all characters P or d of the following properties:

    - character P is represented by a square with an edge length equal to one, a vertical line is attached to the bottom left corner of the square and may have any length,

- character d is represented by a square with an edge length equal to one, a vertical line is attached to the top right corner of the square and may have any length.

Obviously, there are infinitely many such characters. Use the following set of terminal symbols $V_t = \{N, W, S, E\}$, terminal symbols correspond to directions of the chain code – north, west, south, east. Design your set of non-terminal symbols, use a start symbol s. Validate your grammar design on examples. Show all steps of generating at least two P and two d characters.

23. Using Algorithm 7.8 prove or disprove isomorphism of the graphs shown in Figure 7.32.

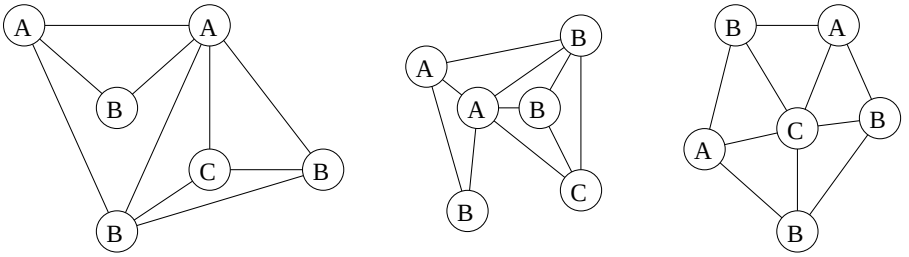

Figure 7.32: *Problem 7.23.*

24. Determine the Levenshtein distance for the following string pairs:

    (a) $S_1 = abadcdefacde \quad S_2 = abadddefacde$

    (b) $S_1 = abadcdefacde \quad S_3 = abadefaccde$

    (c) $S_1 = abadcdefacde \quad S_4 = cbadcacdae$

25. Using genetic algorithm optimization, determine the maximum of the following function (this function has several local maxima; its visualization is, e.g. available in Matlab by typing peaks, see Figure 7.33):

$$z(x,y) = 3(1-x)^2 \exp\left[-x^2-(y+1)^2\right] - 10\left(\tfrac{x}{5} - x^3 - y^5\right)\exp\left[-x^2-y^2\right] - \tfrac{1}{3}\exp\left[-(x+1)^2 - y^2\right]$$

    Develop a program for genetic algorithm-based optimization following Algorithm 7.10. (Alternatively, use one of the many genetic algorithm programs freely available on the World Wide Web.) Design the code string as consisting of n bits for the x value and n bits for the y value, the value of $z(x,y)$ represents the string fitness. Limit your search space to $x \in (-4,4)$ and $y \in (-4,4)$. Explore the role of the starting population, population size S, mutation rate M, and string bit length $2n$ on the speed of convergence and solution accuracy. For several values of S, M, n plot the function values of maximum string fitness, average population fitness, and minimum string fitness as a function of the generation number.

26. Using the definition of the fuzzy set SEVERE as given in Figure 7.28b, sketch the following fuzzy membership functions:

    (a) VERY_SEVERE

    (b) SOMEWHAT_SEVERE

    (c) SOMEWHAT_NOT_SEVERE

27. Using the fuzzy sets DARK and BRIGHT as given in Figure 7.25, sketch the single fuzzy membership function (NOT_VERY_DARK AND NOT_VERY_BRIGHT).

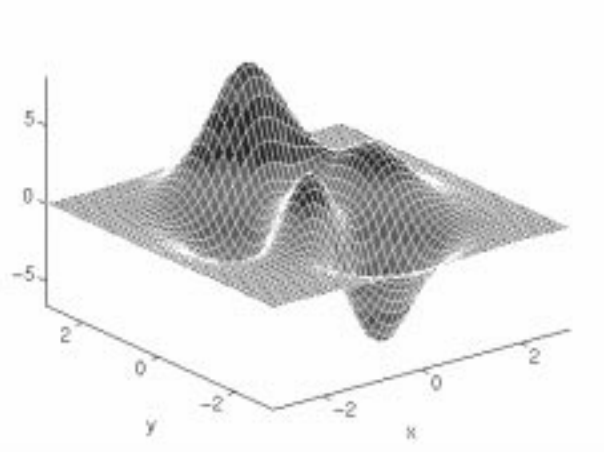

Figure 7.33: *Problem 7.25.*

28. Considering the fuzzy sets A and B given in Figure 7.34, derive intersection, union, and complement of the two fuzzy sets.

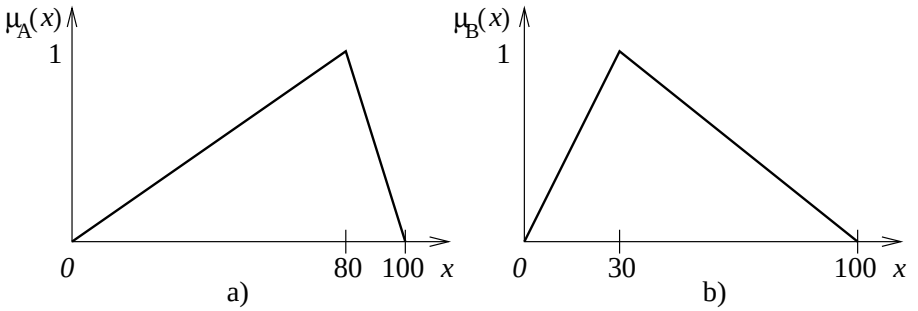

Figure 7.34: *Problem 7.28.*

29. Use the composite moments and composite maximum approaches to defuzzification to find the representative value of the fuzzy set provided in Figure 7.35.

30. Flash flood represents a potential danger in many areas, and its prediction is an important part of meteorological forecasting. Clearly, the following conditions increase the flood danger: 1) rain amount in the past three days, 2) water saturation of soil, and 3) the rainfall expected in the next 24 hours. Assuming the above specified information is available, design a fuzzy logic system to determine the expected flood danger within the next 24 hours.

31. Develop a program implementation of the fuzzy logic system designed in Problem 7.30. Explore, how different membership function shapes, and fuzzy logic composition and decomposition methods influence the achieved results.

7.10 References

[Aarts and van Laarhoven 86] E H L Aarts and P J M van Laarhoven. Simulated annealing: a pedestrian review of the theory and some applications. In P A Devijver and J Kittler, editors,

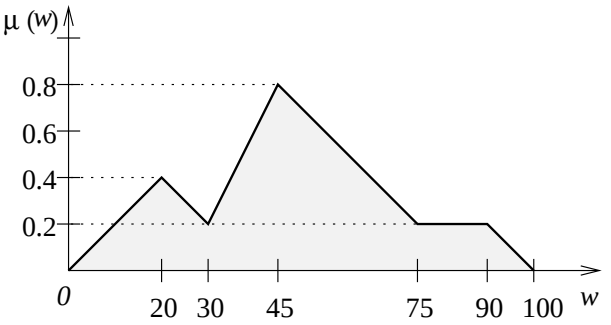

Figure 7.35: *Problem 7.29.*

Pattern Recognition Theory and Applications, pages 179–192. Springer Verlag, Berlin-New York-Tokyo, 1986.

[Abe and Lan 95] S Abe and M Lan. A method for fuzzy rules extraction directly from numerical data and its application to pattern classification. *IEEE Trans. on Fuzzy Systems*, 3(2):129–139, 1995.

[Adeli and Hung 95] H Adeli and S L Hung. *Machine Learning : Neural Networks, Genetic Algorithms, and Fuzzy Systems*. Wiley, New York, 1995.

[Ambler 75] A P H Ambler. A versatile system for computer controlled assembly. *Artificial Intelligence*, 6(2):129–156, 1975.

[Amit 89] D J Amit. *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge University Press, Cambridge, England; New York, 1989.

[Arabie et al. 96] P Arabie, L J Hubert, and G De Soete, editors. *Clustering and Classification*. World Scientific, River Edge, NJ, 1996.

[Azencott 92] R Azencott, editor. *Simulated Annealing : Parallelization Techniques*. Wiley, New York, 1992.

[Baird 84] H S Baird. *Model-Based Image Matching using Location*. MIT Press, Cambridge, Ma, 1984.

[Ballard and Brown 82] D H Ballard and C M Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Barnard 87] Barnard. Stereo matching by hierarchical microcanonical annealing. *Perception*, 1:832, 1987. Vision based application of Simulated Annealing.

[Barrero 91] A Barrero. Inference of tree grammars using negative samples. *Pattern Recognition*, 24(1):1–8, 1991.

[Barrow and Popplestone 71] H G Barrow and R J Popplestone. Relational descriptions in picture processing. *Machine Intelligence*, 6, 1971.

[Berge 76] C Berge. *Graphs and Hypergraphs*. American Elsevier, New York, 2nd edition, 1976.

[Bezdek 81] L C Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithm*. Plenum Press, New York, 1981.

[Bittner and Reingold 75] J R Bittner and E M Reingold. Backtrack programming techniques. *Communications of the ACM*, 18(11):651–656, 1975.

[Blashfield et al. 82] R K Blashfield, M S Aldenderfer, and L C Morey. Cluster analysis software. In P R Krishniah and L N Kanal, editors, *Handbook of Statistics*, pages 245–266. North Holland, Amsterdam, 1982.

[Bloch 93] I Bloch. Fuzzy connectivity and mathematical morphology. *Pattern Recognition Letters*, 14:483–488, 1993.

[Blum and Rivest 88] A Blum and R L Rivest. Training a three node neural network is np-complete. In *Proceedings of IEEE Conference on Neural Information Processing Systems*, page 494, 1988.

[Bouman and Liu 91] C Bouman and B Liu. Multiple resolution segmentation of textured images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):99–113, 1991.

[Braspenning et al. 95] P J Braspenning, F Thuijsman, and A J M M Weijters, editors. *Artificial Neural Networks : An Introduction to ANN Theory and Practice*. Springer Verlag, Berlin; New York, 1995.

[Bron and Kerbosch 73] C Bron and J Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[Buckley 90] F Buckley. *Distance in Graphs*. Addison-Wesley, Redwood City, Ca, 1990.

[Carling 92] A Carling. *Introducing Neural Networks*. Sigma, 1992.

[Carpenter and Grossberg 87a] G A Carpenter and S Grossberg. ART2: Self organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930, 1987.

[Carpenter and Grossberg 87b] G A Carpenter and S Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1987.

[Carpenter and Grossberg 91] G A Carpenter and S Grossberg. *Pattern Recognition by Self-organizing Neural Networks*. MIT Press, Cambridge, Ma, 1991.

[Cerny 85] V Cerny. Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[Chambers 95] L Chambers, editor. *Practical Handbook of Genetic Algorithms*. CRC Press, Boca Raton, FL, 1995.

[Chen 76] C H Chen, editor. *Pattern Recognition and Artificial Intelligence*. Academic Press, New York, 1976.

[Chen et al. 93] C H Chen, L F Pau, and P S P Wang, editors. *Handbook of Pattern Recognition and Computer Vision*. World Scientific, Singapore; River Edge, NJ, 1993.

[Cherkassky et al. 94] V Cherkassky, J H Friedman, and H Wechsler, editors. *From Statistics to Neural Networks*. Springer Verlag, Berlin; New York, 1994.

[Chomsky 66] N Chomsky. *Syntactic Structures*. Mouton, Hague, 6th edition, 1966.

[Chomsky et al. 71] N Chomsky, J P B Allen, and P Van Buren. *Chomsky: Selected Readings*. Oxford University Press, London-New York, 1971.

[Clocksin and Mellish 81] W F Clocksin and C S Mellish. *Programming in Prolog*. Springer Verlag, Berlin-New-York-Tokyo, 1981.

[Cox 94] E Cox. *The Fuzzy Systems Handbook*. AP Professional, Cambridge, 1994.

[Dasarathy 91] B V Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Comp. Society Press, Los Alamitos, Ca, 1991.

[Dellepiane and Fontana 95] S Dellepiane and F Fontana. Extraction of intensity connectedness for image processing. *Pattern Recognition Letters*, 16:313–324, 1995.

[Devijver and Kittler 82] P A Devijver and J Kittler. *Pattern Recognition: A Statistical Approach.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Devijver and Kittler 86] P A Devijver and J Kittler. *Pattern Recognition Theory and Applications.* Springer Verlag, Berlin-New York-Tokyo, 1986.

[Dubes and Jain 76] R C Dubes and A K Jain. Clustering techniques: The user's dilemma. *Pattern Recognition*, 8:247–260, 1976.

[Dubes and Jain 80] R C Dubes and A K Jain. Clustering methodologies in exploratory data analysis. In M Yovits, editor, *Advances in Computers*, pages 113–228. Academic Press, New York, 1980.

[Duda and Hart 73] R O Duda and P E Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1973.

[Even 79] S Even. *Graph Algorithms.* Computer Science Press, Rockville, Md, 1979.

[Everitt and Brian 93] B Everitt and S E Brian. *Cluster Analysis.* E Arnold, Halsted Press, New York; London, 3rd edition, 1993.

[Fausett 94] L Fausett. *Fundamentals of Neural Networks.* Prentice-Hall, 1994.

[Fischler and Elschlager 73] M A Fischler and R A Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, C-22(1):67–92, 1973.

[Fu 68] K S Fu. *Sequential Methods in Pattern Recognition and Machine Learning.* Academic Press, New York, 1968.

[Fu 74] K S Fu. *Syntactic Methods in Pattern Recognition.* Academic Press, New York, 1974.

[Fu 77] K S Fu. *Syntactic Pattern Recognition – Applications.* Springer Verlag, Berlin, 1977.

[Fu 80] K S Fu. Picture syntax. In S K Chang and K S Fu, editors, *Pictorial Information Systems*, pages 104–127. Springer Verlag, Berlin, 1980.

[Fu 82] K S Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Fukunaga 90] K Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, Boston, 2nd edition, 1990.

[Furuhashi 95] T Furuhashi, editor. *Advances in Fuzzy Logic, Neural Networks, and Genetic Algorithms.* Springer Verlag, Berlin ; New York, 1995.

[Geman et al. 90] D Geman, S Geman, C Graffigne, and P Dong. Boundary detection by constrained optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7), 1990.

[Goldberg 89] D E Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Ma, 1989.

[Gonzalez and Thomason 74] R C Gonzalez and M G Thomason. On the inference of tree grammars for pattern recognition. In *Proceedings of the IEEE International Conference on System, Man and Cybernetics*, pages 2–4. IEEE, 1974.

[Haralick and Elliott 79] R M Haralick and G L Elliott. Increasing tree search efficiency for constraint satisfaction problems. In *Proceedings of 6th IJCAI-79*, pages 356–364, 1979.

[Harary 69] F Harary. *Graph Theory.* Addison-Wesley, Reading, Ma, 1969.

[Hayes 77] P J Hayes. In defense of logic. In *Proceedings of 5th IJCAI*, Cambridge, Ma, 1977.

[Haykin 94] S Haykin. *Neural Networks.* Macmillan, 1994.

[Hecht-Nielsen 90] R Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, Reading, Ma, 1990.

[Hecht-Nielson 87] R Hecht-Nielson. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the First IEEE International Conference on Neural Networks*, volume 3, pages 11–14. IEEE, 1987.

[Hlavac and Sara 95] V Hlavac and R Sara, editors. *Computer Analysis of Images and Patterns.* Springer Verlag, Berlin; New York, 1995.

[Homaifar and McCormick 95] A Homaifar and E McCormick. Simultaneous design of memebership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE Trans. on Fuzzy Systems*, 3(2):129–139, 1995.

[Hopfield and Tank 85] J J Hopfield and D W Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[Hopfield and Tank 86] J J Hopfield and D W Tank. Computing with neural circuits: A model. *Science*, 233:625–633, 1986.

[Horikawa et al. 92] S Horikawa, T Furuhashi, and Y Uchikawa. On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. *IEEE Trans. on Neural Networks*, 3(5):801–806, 1992.

[Ishibuchi et al. 92] H Ishibuchi, K Nozaki, and H Tanaka. Distributed representation of fuzzy rules and its application to pattern classification. *Fuzzy Sets and Syst.*, 52:21–32, 1992.

[Ishibuchi et al. 95] H Ishibuchi, K Nozaki, N Yamamoto, and H Tanaka. Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3:260–270, 1995.

[Johnson and Wichern 90] R A Johnson and D W Wichern. *Applied Multivariate Statistical Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 1990.

[Judd 90] J S Judd. *Neural Network Design and the Complexity of Learning.* MIT Press, Cambridge, Ma, 1990.

[Kandel 82] A Kandel. *Fuzzy Techniques in Pattern Recognition.* Wiley, New York, 1982.

[Kaufman and Rousseeuw 90] L Kaufman and P J Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley and Sons, New York, 1990.

[Kaufmann 75] A Kaufmann. *Introduction to the Theory of Fuzzy Subsets–Fundamental Theoretical Elements, Vol 1.* Academic Press, New York, 1975.

[Kirkpatrick et al. 83] S Kirkpatrick, C D Gelatt, and M P Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Kohonen 88] T Kohonen. The "neural" phonetic typewriter. *Computer*, pages 11–22, March 1988.

[Kohonen 95] T Kohonen. *Self-organizing Maps.* Springer Verlag, Berlin; New York, 1995.

[Kolmogorov 63] A N Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 144:679–681, 1963. (AMS Translation, 28, 55-59).

[Kosko 91] B Kosko. Adaptive bidirectional associative memories. In G A Carpenter and S Grossberg, editors, *Pattern Recognition by Self-Organizing Neural Networks*, pages 425–450. MIT Press, Cambridge, Ma, 1991.

[Kosko 92] B Kosko. *Neural Networks and Fuzzy Systems.* Prentice Hall, Englewood Cliffs, NJ, 1992.

[Kowalski 79] R Kowalski. *Logic for Problem Solving.* North Holland, Amsterdam, 1979.

[Lakemeyer and Nebel 94] G Lakemeyer and B Nebel, editors. *Foundations of Knowledge Representation and Reasoning.* Springer Verlag, Berlin; New York, 1994.

[Lau 89] H T Lau. *Algorithms on Graphs.* TAB Professional and Reference Books, Blue Ridge Summit, Pa, 1989.

[Linggard et al. 92] R Linggard, C Nightingale, and D Myers, editors. *Neural networks for vision, speech and natural language.* Chapman and Hall, 1992.

[MacQueen 67] J MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium – 1*, pages 281–297, 1967.

[Masters 95] T Masters. *Advanced Algorithms for Neural Networks : A C++ Sourcebook.* Wiley, New York, 1995.

[Masuch and Polos 94] M Masuch and L Polos, editors. *Knowledge Representation and Reasoning under Uncertainty : Logic at Work.* Springer Verlag, Berlin; New York, 1994.

[McCulloch and Pitts 43] W S McCulloch and W Pitts. A logical calculus of ideas immanent in nervous activity. *Bull. Math. Biophysics*, 5:115–133, 1943.

[McEliece et al. 87] R J McEliece, E C Posner, E R Rodemich, and S S Venkatesh. The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, 33:461, 1987.

[McHugh 90] J A McHugh. *Algorithmic Graph Theory.* Prentice-Hall, Englewood Cliffs, NJ, 1990.

[McQuitty 87] L L McQuitty. *Pattern-Analytic Clustering: Theory, Method, Research, and Configural Findings.* University Press of America, Lanham, NY, 1987.

[Metropolis et al. 53] N Metropolis, A W Rosenbluth, M N Rosenbluth, A H Teller, and E Teller. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[Michalewicz 94] Z Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer Verlag, Berlin; New York, 2nd edition, 1994.

[Michalski et al. 83] R S Michalski, J G Carbonell, and T M Mitchell. *Machine Learning I, II.* Morgan Kaufmann Publishers, Los Altos, Ca, 1983.

[Minsky 88] M L Minsky. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, Ma, 2nd edition, 1988.

[Mitchell 96] M Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, 1996.

[Mohring 91] R H Mohring, editor. *Graph-Theoretic Concepts in Computer Science - 16th WG'90,* Berlin-New York-Tokyo, 1991. Springer Verlag.

[Mozer 91] M C Mozer. *The Perception of Multiple Objects: A Connectionist Approach.* MIT Press, Cambridge, Ma, 1991.

[Nagl 90] M Nagl, editor. *Graph-Theoretic Concepts in Computer Science - 15th WG'89*, Berlin-New York-Tokyo, 1990. Springer Verlag.

[Niemann 90] H Niemann. *Pattern Analysis and Understanding.* Springer Verlag, Berlin-New York-Tokyo, 2nd edition, 1990.

[Nigrin 93] A Nigrin. *Neural Networks for Pattern Recognition.* MIT Press, Cambridge, MA, 1993.

[Nilsson 71] N J Nilsson. *Problem Solving Methods in Artificial Intelligence.* McGraw Hill, New York, 1971.

[Nilsson 82] N J Nilsson. *Principles of Artificial Intelligence*. Springer Verlag, Berlin, 1982.

[Oja 83] E Oja. *Subspace Methods of Pattern Recognition*. Research Studies Press, Letchworth, England, 1983.

[Otten and van Ginneken 89] R H J M Otten and L P P P van Ginneken. *The Annealing Algorithm*. Kluwer Academic Publishers, Norwell, Ma, 1989.

[Pal 91] S K Pal. Fuzzy tools for the management of uncertainty in pattern recognition, image analysis, vision, and expert systems. *International Journal of System Science*, 22:511–548, 1991.

[Pal and Majumder 86] S K Pal and D D Majumder. *Fuzzy Mathematical Approach to Pattern Recognition*. Wiley, New York, 1986.

[Park 96] W Park. *Automated Determination of Fuzzy Rules and Membership Functions: Application to Analysis of Pulmonary CT Images*. PhD thesis, The University of Iowa, 1996.

[Park et al. 96] W Park, E A Hoffman, and M Sonka. Fuzzy logic approach to extraction of intrathoracic airway trees from three-dimensional CT images. In *Image Processing, Proceedings SPIE Vol. 2710*, pages 210–219, SPIE, Bellingham, WA, 1996.

[Patrick and Fattu 86] E A Patrick and J M Fattu. *Artificial Intelligence with Statistical Pattern Recognition*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[Pavel 93] M Pavel. *Fundamentals of Pattern Recognition*. M Dekker, New York, 2nd edition, 1993.

[Pavlidis 77] T Pavlidis. *Structural Pattern Recognition*. Springer Verlag, Berlin, 1977.

[Pavlidis 80] T Pavlidis. Structural descriptions and graph grammars. In S K Chang and K S Fu, editors, *Pictorial Information Systems*, pages 86–103, Springer Verlag, Berlin, 1980.

[Pedrycz 95] W Pedrycz. *Fuzzy Sets Engineering*. CRC Press, Boca Raton, FL, 1995.

[Pospesel 76] H Pospesel. *Predicate Logic*. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[Pudil et al. 94a] P Pudil, J Novovicova, and J Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15:1119–1125, 1994.

[Pudil et al. 94b] P Pudil, J Novovicova, and J Kittler. Simultaneous learning of decision rules and important atrtributes for classification problems in image analysis. *Image and Vision Computing*, 12:193–198, 1994.

[Rao 65] C R Rao. *Linear Statistical Inference and its Application*. John Wiley and Sons, New York, 1965.

[Rawlins 91] G J E Rawlins. *Foundations of Genetic Algorithms*. Morgan Kaufmann, San Mateo, Ca, 1991.

[Reichgelt 91] H Reichgelt. *Knowledge Representation: An AI Perspective*. Ablex Publishing Corporation, Norwood, NJ, 1991.

[Rogers and Kabrisky 91] S K Rogers and M Kabrisky. *An Introduction to Biological and Artificial Neural Networks for Pattern Recognition*. SPIE, Bellingham, Wa, 1991.

[Romesburg 84] H C Romesburg. *Cluster Analysis for Researchers*. Lifetime Learning Publications, Belmont, Ca, 1984.

[Rosenblatt 62] R Rosenblatt. *Principles of Neurodynamics*. Spartan books, Washington, D.C., 1962.

[Rosenfeld 79a] A Rosenfeld. Fuzzy digital topology. *Information Control*, 40:76–87, 1979.

[Rosenfeld 79b] A Rosenfeld. *Picture Languages – Formal Models for Picture Recognition*. Academic Press, New York, 1979.

[Rosenfeld 83] A Rosenfeld. On connectivity properties of grayscale pictures. *Information Control*, 16:47–50, 1983.

[Rosenfeld 84a] A Rosenfeld. The diameter of a fuzzy set. *Fuzzy Sets and Systems*, 13:241–246, 1984.

[Rosenfeld 84b] A Rosenfeld. The fuzzy geometry of image subsets. *Pattern Recognition Letters*, 2:311–317, 1984.

[Rosenfeld 85] A Rosenfeld. The perimeter of a fuzzy set. *Pattern Recognition*, 18:125–130, 1985.

[Rumelhart and McClelland 86] D Rumelhart and J McClelland. *Parallel Distributed Processing*. MIT Press, Cambridge, Ma, 1986.

[Schalkoff 92] R J Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches*. Wiley, New York, 1992.

[Schutzer 87] D Schutzer. *Artificial Intelligence, An Application-Oriented Approach*. Van Nostrand Reinhold, New York, 1987.

[Sedgewick 84] R Sedgewick. *Algorithms*. Addison-Wesley, Reading, Ma, 2nd edition, 1984.

[Sejnowski and Rosenberg 87] T J Sejnowski and C R Rosenberg. Parallel systems that learn to pronounce English text. *Complex Systems*, 1:145–168, 1987.

[Sethi and Jain 91] I K Sethi and A K Jain, editors. *Artificial Neural Networks and Statistical Pattern Recognition : Old and New Connections*. North-Holland, Amsterdam; New York, 1991.

[Shapiro and Haralick 80] L G Shapiro and R M Haralick. Algorithms for inexact matching. In *Proceedings 5th International Conference on Pattern Recognition*, pages 202–207, IEEE Comp. Society Press, Los Alamitos, Ca, 1980.

[Sharples et al. 89] M Sharples, D Hogg, C Hutchinson, S Torrance, and D Young. *Computers and Thought, A Practical Introduction to Artificial Intelligence*. The MIT Press, Cambridge, Ma, 1989.

[Simons 84] G L Simons. *Introducing Artificial Intelligence*. NCC Publications, Manchester, 1984.

[Simpson 90] P K Simpson. *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations*. Pergamon Press, New York, 1990.

[Simpson 92] P K Simpson. Fuzzy min-max neural networks – Part 1: Classification. *IEEE Trans. on Fuzzy Systems*, 3(2):129–139, 1992.

[Sklansky 81] J Sklansky. *Pattern Classifiers and Trainable Machines*. Springer Verlag, New York, 1981.

[Sonka 86] M Sonka. A new texture recognition method. *Computers and Artificial Intelligence*, 5(4):357–364, 1986.

[Tan et al. 92] H K Tan, S B Gelfand, and E J Delp. A cost minimization approach to edge detection using simulated annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1), 1992.

[Tucker 95] A Tucker. *Applied Combinatorics*. Wiley, New York, 3rd edition, 1995.

[Udupa and Samarasekera 96] J K Udupa and S Samarasekera. Fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation. *Graphical Models and Image Processing*, 58:246–261, 1996.

[Ullmann 76] J R Ullmann. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, 23(1):31–42, 1976.

[van Laarhoven 88] P J M van Laarhoven. *Theoretical and Computational Aspects of Simulated Annealing*. Centrum voor Wiskunde en Informatik, Amsterdam, 1988.

[van Laarhoven and Aarts 87] P J M van Laarhoven and E H L Aarts. *Simulated Annealing: Theory and Applications*. Dordrecht and Kluwer Academic Publishers, Norwell, Ma, 1987.

[Wasserman 89] P D Wasserman. *Neural Computing – Theory and Practice*. Van Nostrand Rheinhold, New York, 1989.

[Wechsler 90] H Wechsler. *Computational Vision*. Academic Press, London – San Diego, 1990.

[Winston 75] P H Winston, editor. *The Psychology of Computer Vision*. McGraw Hill, New York, 1975.

[Winston 84] P H Winston. *Artificial Intelligence*. Addison-Wesley, Reading, Ma, 2nd edition, 1984.

[Yang et al. 89] B Yang, W E Snyder, and G L Bilbro. Matching oversegmented 3D images to models using association graphs. *Image and Vision Computing*, 7(2):135–143, 1989.

[Young and Calvert 74] T Y Young and T W Calvert. *Classification, Estimation, and Pattern Recognition*. American Elsevier, New York-London-Amsterdam, 1974.

[Zadeh 65] L A Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[Zadeh and Kacprzyk 92] L A Zadeh and J Kacprzyk, editors. *Fuzzy Logic for the Management of Uncertainty*. Wiley, New York, 1992.

[Zdrahal 81] Z Zdrahal. A structural method of scene analysis. In *Proceedings of IJCAI-81*, pages 680–682, Vancouver, BC, Canada, 1981.

[Zhou 92] Y T Zhou. *Artificial Neural Networks for Computer Vision*. Springer Verlag, New York, 1992.

[Zimmermann 87] H J Zimmermann. *Fuzzy sets, decision making and expert systems*. Kluwer Academic Publishers, Boston, 1987.

[Zimmermann 91] H Zimmermann. *Fuzzy Set Theory and Its Applications*. Kluwer, Boston, MA, 1991.

[Zimmermann et al. 84] H J Zimmermann, L A Zadeh, and B R Gaines. *Fuzzy Sets and Decision Analysis*. North Holland, Amsterdam-New York, 1984.