

Pham, D. T. et al "Computational Intelligence for Manufacturing"
Computational Intelligence in Manufacturing Handbook
Edited by Jun Wang et al
Boca Raton: CRC Press LLC,2001

1

Computational Intelligence for Manufacturing

D. T. Pham
University of Wales
P. T. N. Pham
University of Wales

- 1.1 Introduction
- 1.2 Knowledge-Based Systems
- 1.3 Fuzzy Logic
- 1.4 Inductive Learning
- 1.5 Neural Networks
- 1.6 Genetic Algorithms
- 1.7 Some Applications in Engineering and Manufacture
- 1.8 Conclusion

1.1 Introduction

Computational intelligence refers to intelligence artificially realised through computation. Artificial intelligence emerged as a computer science discipline in the mid-1950s. Since then, it has produced a number of powerful tools, some of which are used in engineering to solve difficult problems normally requiring human intelligence. Five of these tools are reviewed in this chapter with examples of applications in engineering and manufacturing: knowledge-based systems, fuzzy logic, inductive learning, neural networks, and genetic algorithms. All of these tools have been in existence for more than 30 years and have found many practical applications.

1.2 Knowledge-Based Systems

Knowledge-based systems, or expert systems, are computer programs embodying knowledge about a narrow domain for solving problems related to that domain. An expert system usually comprises two main elements, a knowledge base and an inference mechanism. The knowledge base contains domain knowledge which may be expressed as any combination of “If-Then” rules, factual statements (or assertions), frames, objects, procedures, and cases. The inference mechanism is that part of an expert system which manipulates the stored knowledge to produce solutions to problems. Knowledge manipulation methods include the use of inheritance and constraints (in a frame-based or object-oriented expert system), the retrieval and adaptation of case examples (in a case-based expert system), and the application of inference rules such as *modus ponens* (If A Then B; A Therefore B) and *modus tollens* (If A Then B; Not B Therefore Not A) according to “forward chaining” or “backward chaining” control procedures and “depth-first” or “breadth-first” search strategies (in a rule-based expert system).

With *forward chaining* or data-driven inferencing, the system tries to match available facts with the If portion of the If-Then rules in the knowledge base. When matching rules are found, one of them is

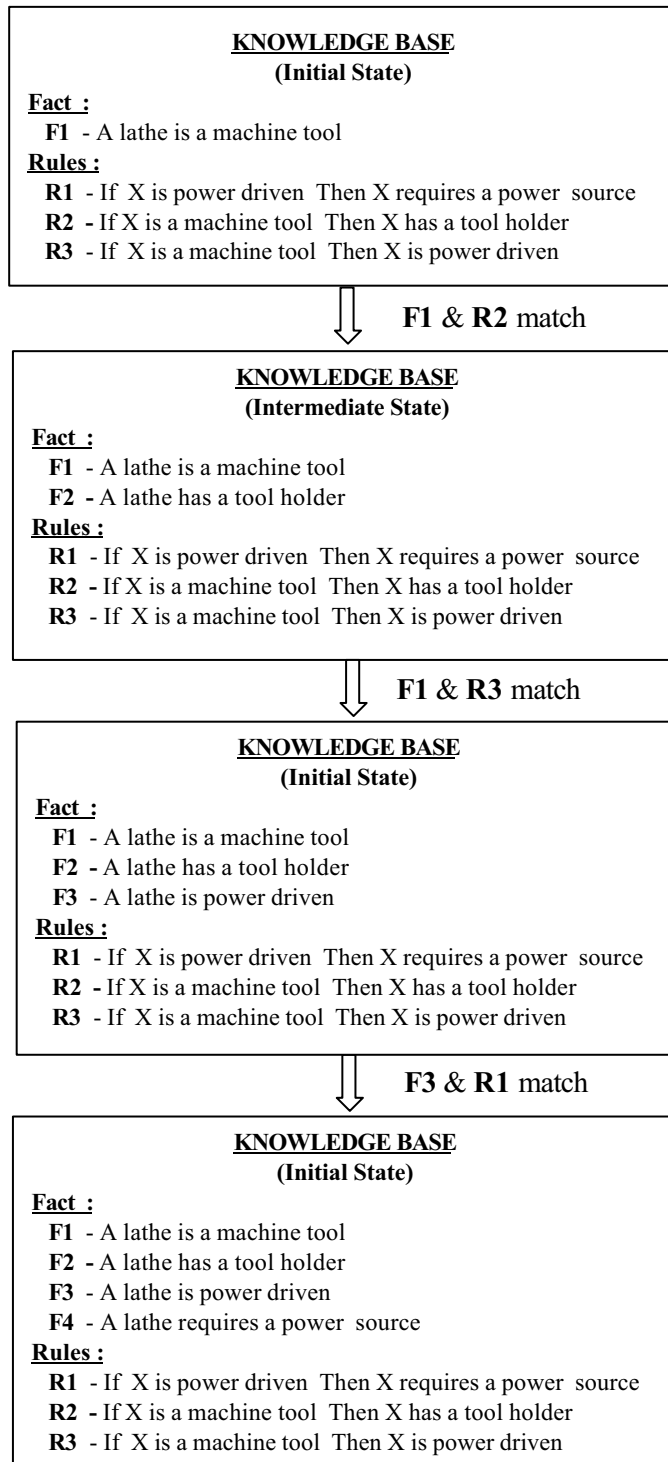


FIGURE 1.1(a) An example of forward chaining.

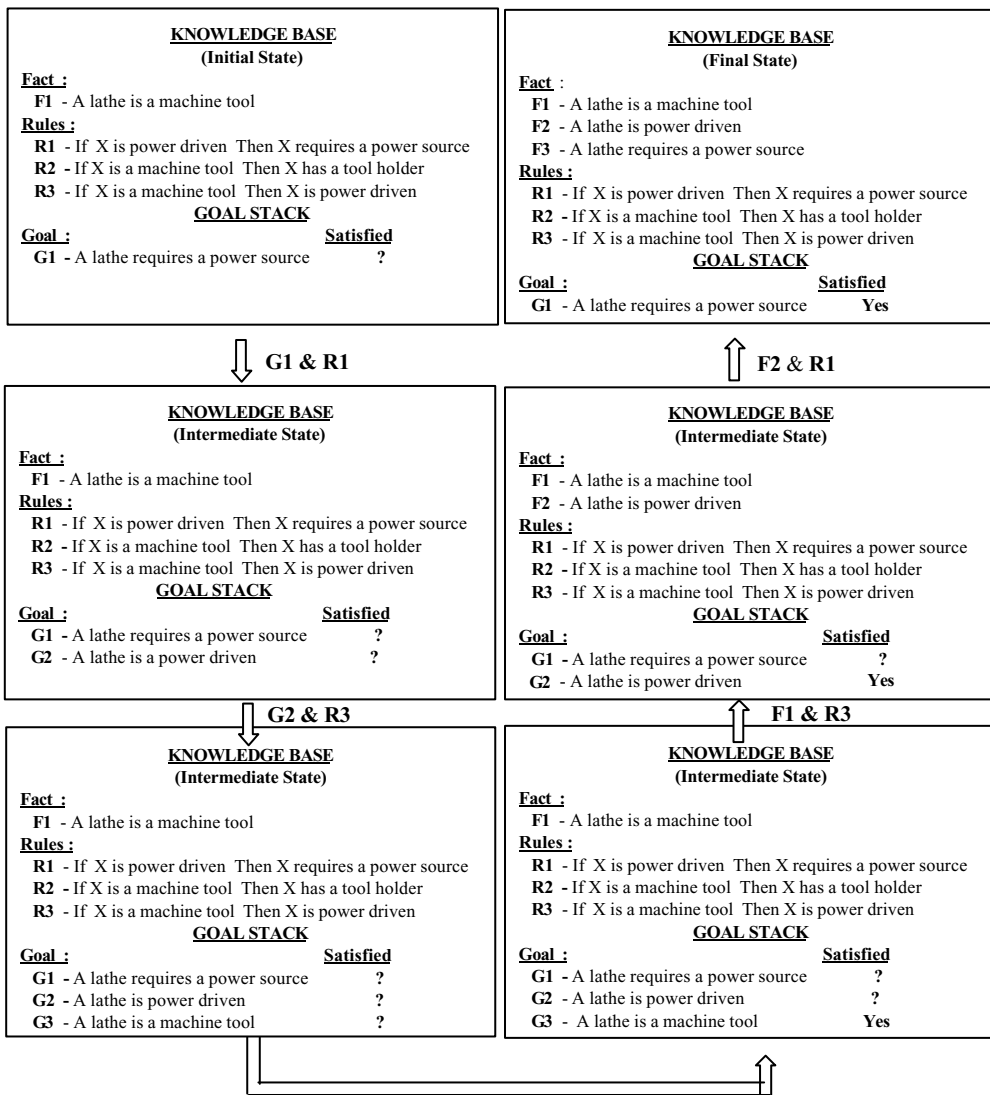


FIGURE 1.1(b) An example of backward chaining.

“fired,” i.e., its Then part is made true, generating new facts and data which in turn cause other rules to “fire.” Reasoning stops when no more new rules can fire. In *backward chaining* or goal-driven inferencing, a goal to be proved is specified. If the goal cannot be immediately satisfied by existing facts in the knowledge base, the system will examine the If-Then rules for rules with the goal in their Then portion. Next, the system will determine whether there are facts that can cause any of those rules to fire. If such facts are not available they are set up as subgoals. The process continues recursively until either all the required facts are found and the goal is proved or any one of the subgoals cannot be satisfied, in which case the original goal is disproved. Both control procedures are illustrated in Figure 1.1. Figure 1.1(a) shows how, given the assertion that a lathe is a machine tool and a set of rules concerning machine tools, a forward-chaining system will generate additional assertions such as “a lathe is power driven” and “a lathe has a tool holder.” Figure 1.1(b) details the backward-chaining sequence producing the answer to the query “does a lathe require a power source?”

In the forward-chaining example of Figure 1.1(a), both rules R2 and R3 simultaneously qualify for firing when inferencing starts as both their If parts match the presented fact F1. Conflict resolution has to be performed by the expert system to decide which rule should fire. The conflict resolution method adopted in this example is “first come, first served”: R2 fires, as it is the first qualifying rule encountered. Other conflict resolution methods include “priority,” “specificity,” and “recency.”

The search strategies can also be illustrated using the forward-chaining example of Figure 1.1(a). Suppose that, in addition to F1, the knowledge base also initially contains the assertion “a CNC turning centre is a machine tool.” Depth-first search involves firing rules R2 and R3 with X instantiated to “lathe” (as shown in Figure 1.1(a)) before firing them again with X instantiated to “CNC turning centre.” Breadth-first search will activate rule R2 with X instantiated to “lathe” and again with X instantiated to “CNC turning centre,” followed by rule R3 and the same sequence of instantiations. Breadth-first search finds the shortest line of inferencing between a start position and a solution if it exists. When guided by heuristics to select the correct search path, depth-first search might produce a solution more quickly, although the search might not terminate if the search space is infinite [Jackson, 1999].

For more information on the technology of expert systems, see [Pham and Pham, 1988; Durkin, 1994; Jackson, 1999].

Most expert systems are nowadays developed using programs known as “shells.” These are essentially ready-made expert systems complete with inferencing and knowledge storage facilities but without the domain knowledge. Some sophisticated expert systems are constructed with the help of “development environments.” The latter are more flexible than shells in that they also provide means for users to implement their own inferencing and knowledge representation methods. More details on expert systems shells and development environments can be found in [Price, 1990].

Among the five tools considered in this chapter, expert systems are probably the most mature, with many commercial shells and development tools available to facilitate their construction. Consequently, once the domain knowledge to be incorporated in an expert system has been extracted, the process of building the system is relatively simple. The ease with which expert systems can be developed has led to a large number of applications of the tool. In engineering, applications can be found for a variety of tasks including selection of materials, machine elements, tools, equipment and processes, signal interpreting, condition monitoring, fault diagnosis, machine and process control, machine design, process planning, production scheduling and system configuring. The following are recent examples of specific tasks undertaken by expert systems:

- Identifying and planning inspection schedules for critical components of an offshore structure [Peers et al., 1994]
- Training technical personnel in the design and evaluation of energy cogeneration plants [Lara Rosano et al., 1996]
- Configuring paper feeding mechanisms [Koo and Han, 1996]
- Carrying out automatic remeshing during a finite-elements analysis of forging deformation [Yano et al., 1997]
- Storing, retrieving, and adapting planar linkage designs [Bose et al., 1997]
- Designing additive formulae for engine oil products [Shi et al., 1997]

1.3 Fuzzy Logic

A disadvantage of ordinary rule-based expert systems is that they cannot handle new situations not covered explicitly in their knowledge bases (that is, situations not fitting exactly those described in the “If” parts of the rules). These rule-based systems are completely unable to produce conclusions when such situations are encountered. They are therefore regarded as shallow systems which fail in a “brittle” manner, rather than exhibit a gradual reduction in performance when faced with increasingly unfamiliar problems, as human experts would.

The use of fuzzy logic [Zadeh, 1965] that reflects the qualitative and inexact nature of human reasoning can enable expert systems to be more resilient. With fuzzy logic, the precise value of a variable is replaced by a linguistic description, the meaning of which is represented by a fuzzy set, and inferencing is carried out based on this representation. Fuzzy set theory may be considered an extension of classical set theory. While classical set theory is about “crisp” sets with sharp boundaries, fuzzy set theory is concerned with “fuzzy” sets whose boundaries are “gray.”

In classical set theory, an element u_i can either belong or not belong to a set A , i.e., the degree to which element u belongs to set A is either 1 or 0. However, in fuzzy set theory, the degree of belonging of an element u to a fuzzy set \underline{A} is a real number between 0 and 1. This is denoted by $\mu_{\underline{A}}(u_i)$, the grade of membership of u_i in A . Fuzzy set \underline{A} is a fuzzy set in U , the “universe of discourse” or “universe” which includes all objects to be discussed. $\mu_{\underline{A}}(u_i)$ is 1 when u_i is definitely a member of A and $\mu_{\underline{A}}(u_i)$ is 0 when u_i is definitely not a member of A . For instance, a fuzzy set defining the term “normal room temperature” might be as follows:

$$\text{normal room temperature} \quad \begin{array}{l} 0.0/ \text{below } 10^{\circ}\text{C} + 0.3/ 10^{\circ}\text{C} - 16^{\circ}\text{C} + 0.8/ 16^{\circ}\text{C} - 18^{\circ}\text{C} + \\ 1.0/ 18^{\circ}\text{C} - 22^{\circ}\text{C} + 0.8/ 22^{\circ}\text{C} - 24^{\circ}\text{C} + 0.3/ 24^{\circ}\text{C} - 30^{\circ}\text{C} + 0.0/ \text{above } 30^{\circ}\text{C} \end{array}$$

Equation (1.1)

The values 0.0, 0.3, 0.8, and 1.0 are the grades of membership to the given fuzzy set of temperature ranges below 10°C (above 30°C), between 10°C and 16°C (24°C to 30°C), between 16°C and 18°C (22°C to 24°C), and between 18°C and 22°C. Figure 1.2(a) shows a plot of the grades of membership for “normal room temperature.” For comparison, Figure 1.2(b) depicts the grades of membership for a crisp set defining room temperatures in the normal range.

Knowledge in an expert system employing fuzzy logic can be expressed as qualitative statements (or fuzzy rules) such as “If the room temperature is normal, Then set the heat input to normal,” where “normal room temperature” and “normal heat input” are both fuzzy sets.

A fuzzy rule relating two fuzzy sets \underline{A} and \underline{B} is effectively the Cartesian product $\underline{A} \times \underline{B}$, which can be represented by a relation matrix $[\underline{R}]$. Element R_{ij} of $[\underline{R}]$ is the membership to $\underline{A} \times \underline{B}$ of pair (u_i, v_j) , $u_i \in \underline{A}$ and $v_j \in \underline{B}$. R_{ij} is given by

$$R_{ij} = \min\left(\mu_{\underline{A}}(u_i), \mu_{\underline{B}}(v_j)\right) \quad \text{Equation (1.2)}$$

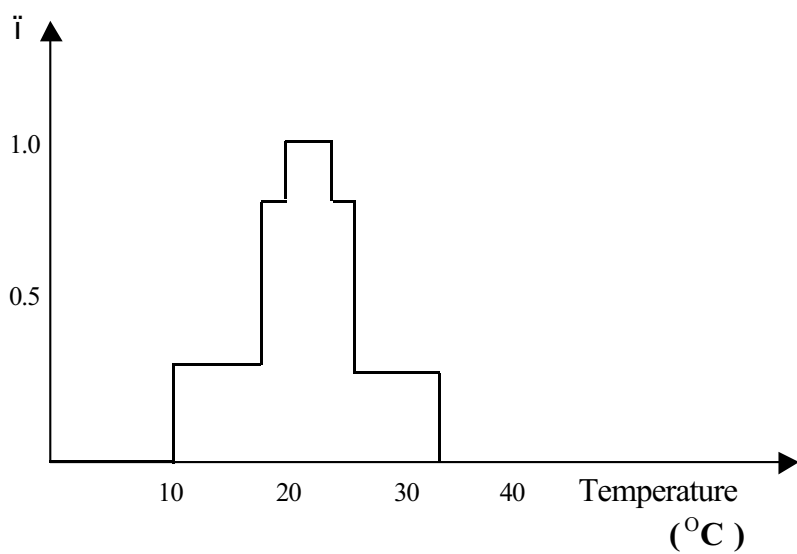
For example, with “normal room temperature” defined as before and “normal heat input” described by

$$\text{normal heat input} \equiv 0.2/ 1\text{kW} + 0.9/ 2\text{kW} + 0.2/ 3\text{kW} \quad \text{Equation (1.3)}$$

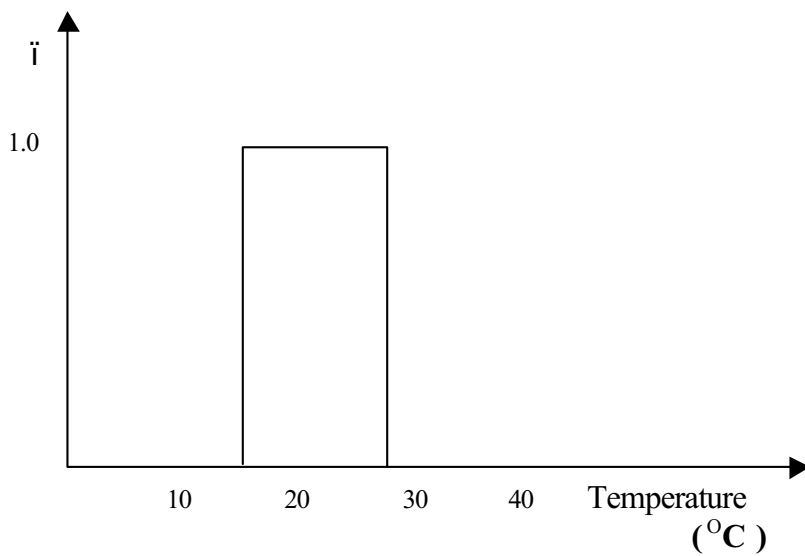
$[\underline{R}]$ can be computed as:

$$[\underline{R}] = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.2 & 0.3 & 0.2 \\ 0.2 & 0.8 & 0.2 \\ 0.2 & 0.9 & 0.2 \\ 0.2 & 0.8 & 0.2 \\ 0.2 & 0.3 & 0.2 \\ 0.0 & 0.0 & 0.0 \end{bmatrix} \quad \text{Equation (1.4)}$$

A reasoning procedure known as the *compositional rule of inference*, which is the equivalent of the *modus ponens* rule in rule-based expert systems, enables conclusions to be drawn by generalization



(a)



(b)

FIGURE 1.2 (a) Fuzzy set of "normal temperature." (b) Crisp set of "normal temperature."

(extrapolation or interpolation) from the qualitative information stored in the knowledge base. For instance, when the room temperature is detected to be "slightly below normal," a temperature-controlling fuzzy expert system might deduce that the heat input should be set to "slightly above normal." Note that

this conclusion might not be contained in any of the fuzzy rules stored in the system. A well-known compositional rule of inference is the *max-min rule*. Let $[R]$ represent the fuzzy rule “If A Then B ” and

$a \equiv \sum_i \mu_i / u_i$ a fuzzy assertion. A and a are fuzzy sets in the same universe of discourse. The max-

min rule enables a fuzzy conclusion $b \equiv \sum_j \lambda_j / v_j$ to be inferred from a and $[R]$ as follows:

$$b = a \circ [R] \quad \text{Equation (1.5)}$$

$$\lambda_j = \max_i \left[\min(\mu_i, R_{ij}) \right] \quad \text{Equation (1.6)}$$

For example, given the fuzzy rule “If the room temperature is normal, Then set the heat input to normal” where “normal room temperature” and “normal heat input” are as defined previously, and a fuzzy temperature measurement of

$$\begin{aligned} \text{temperature} &\equiv 0.0 / \text{below } 10^\circ\text{C} + 0.4 / 10^\circ\text{C} - 16^\circ\text{C} + 0.8 / 16^\circ\text{C} - 18^\circ\text{C} \\ &+ 0.8 / 18^\circ\text{C} - 22^\circ\text{C} + 0.2 / 22^\circ\text{C} - 24^\circ\text{C} + 0.0 / 24^\circ\text{C} - 30^\circ\text{C} + 0.0 / \text{above } 30^\circ\text{C} \end{aligned} \quad \text{Equation (1.7)}$$

the heat input will be deduced as

$$\begin{aligned} \text{heat input} &= \text{temperature} \circ [R] \\ &= 0.2 / 1kW + 0.8 / 2kW + 0.2 / 3kW \end{aligned} \quad \text{Equation (1.8)}$$

For further information on fuzzy logic, see [Kaufmann, 1975; Zimmermann, 1991].

Fuzzy logic potentially has many applications in engineering, where the domain knowledge is usually imprecise. Notable successes have been achieved in the area of process and machine control, although other sectors have also benefited from this tool. Recent examples of engineering applications include:

- Controlling the height of the arc in a welding process [Bigand et al., 1994]
- Controlling the rolling motion of an aircraft [Ferreiro Garcia, 1994]
- Controlling a multi-fingered robot hand [Bas and Erkmen, 1995]
- Analysing the chemical composition of minerals [Da Rocha Fernandes and Cid Bastos, 1996]
- Determining the optimal formation of manufacturing cells [Szwarc et al., 1997]
- Classifying discharge pulses in electrical discharge machining [Tarng et al., 1997]

1.4 Inductive Learning

The acquisition of domain knowledge to build into the knowledge base of an expert system is generally a major task. In some cases, it has proved a bottleneck in the construction of an expert system. Automatic knowledge acquisition techniques have been developed to address this problem. Inductive learning is an automatic technique for knowledge acquisition. The inductive approach produces a structured representation of knowledge as the outcome of learning. Induction involves generalising a set of examples to yield a selected representation that can be expressed in terms of a set of rules, concepts or logical inferences, or a decision tree.

TABLE 1.1 Training Set for the Cutting Tool Problem

Example	Sensor_1	Sensor_2	Sensor_3	Tool State
1	0	-1	0	Normal
2	1	0	0	Normal
3	1	-1	1	Worn
4	1	0	1	Normal
5	0	0	1	Normal
6	1	1	1	Worn
7	1	-1	0	Normal
8	0	-1	1	Worn

An inductive learning program usually requires as input a set of examples. Each example is characterised by the values of a number of attributes and the class to which it belongs. In one approach to inductive learning, through a process of “dividing and conquering” where attributes are chosen according to some strategy (for example, to maximise the information gain) to divide the original example set into subsets, the inductive learning program builds a decision tree that correctly classifies the given example set. The tree represents the knowledge generalised from the specific examples in the set. This can subsequently be used to handle situations not explicitly covered by the example set.

In another approach known as the “covering approach,” the inductive learning program attempts to find groups of attributes uniquely shared by examples in given classes and forms rules with the If part as conjunctions of those attributes and the Then part as the classes. The program removes correctly classified examples from consideration and stops when rules have been formed to classify all examples in the given set.

A new approach to inductive learning, *inductive logic programming*, is a combination of induction and logic programming. Unlike conventional inductive learning which uses propositional logic to describe examples and represent new concepts, inductive logic programming (ILP) employs the more powerful predicate logic to represent training examples and background knowledge and to express new concepts. Predicate logic permits the use of different forms of training examples and background knowledge. It enables the results of the induction process — that is, the induced concepts — to be described as general first-order clauses with variables and not just as zero-order propositional clauses made up of attribute–value pairs. There are two main types of ILP systems, the first based on the top-down generalisation/specialisation method, and the second on the principle of inverse resolution [Muggleton, 1992].

A number of inductive learning programs have been developed. Some of the well-known programs are ID3 [Quinlan, 1983], which is a divide-and-conquer program; the AQ program [Michalski, 1990], which follows the covering approach; the FOIL program [Muggleton, 1992], which is an ILP system adopting the generalisation/specialisation method; and the GOLEM program [Muggleton, 1992], which is an ILP system based on inverse resolution. Although most programs only generate crisp decision rules, algorithms have also been developed to produce fuzzy rules [Wang and Mendel, 1992].

Figure 1.3 shows the main steps in RULES-3 Plus, an induction algorithm in the covering category [Pham and Dimov, 1997] and belonging to the RULES family of rule extraction systems [Pham and Aksoy, 1994, 1995a, 1995b]. The simple problem of detecting the state of a metal cutting tool is used to explain the operation of RULES-3 Plus. Three sensors are employed to monitor the cutting process and, according to the signals obtained from them (1 or 0 for sensors 1 and 3; -1, 0, or 1 for sensor 2), the tool is inferred as being “normal” or “worn.” Thus, this problem involves three attributes which are the states of sensors 1, 2, and 3, and the signals that they emit constitute the values of those attributes. The example set for the problem is given in Table 1.1.

In Step 1, example 1 is used to form the attribute–value array SETAV which will contain the following attribute–value pairs: [Sensor_1 = 0], [Sensor_2 = -1] and [Sensor_3 = 0].

In Step 2, the partial rule set PRSET and T_PRSET, the temporary version of PRSET used for storing partial rules in the process of rule construction, are initialised. This creates for each of these sets three expressions having null conditions and zero H measures. The H measure for an expression is defined as

```

Step 1. Take an unclassified example and form array SETAV.
Step 2. Initialise arrays PRSET and T_PRSET (PRSET and T_PRSET will consist of  $m_{\text{PRSET}}$  expressions
with null conditions and zero H measures) and set  $n_{\text{CO}} = 0$ .
Step 3. IF  $n_{\text{CO}} < n_a$ 
      THEN  $n_{\text{CO}} = n_{\text{CO}} + 1$  and set  $m = 0$ ;
      ELSE the example itself is taken as a rule and STOP.
Step 4. DO
       $m = m + 1$ ;
      Specialise expression  $m$  in PRSET by appending to it a condition from SETAV
      that differs from the conditions already included in the expression;
      Compute the H measure for the expression;
      IF its H measure is higher than the H measure of any expression in T_PRSET
      THEN replace the expression having the lowest H measure with the newly
      formed expression;
      ELSE discard the new expression;
      WHILE  $m \leq m_{\text{PRSET}}$  .
Step 5. IF there are consistent expressions in T_PRSET
      THEN choose as a rule the expression that has the highest H measure and
      discard the others;
      ELSE copy T_PRSET into PRSET;
      initialise T_PRSET and go to Step 4.

```

n_{CO} - number of conditions; n_a - number of attributes;
 m_{PRSET} - number of expressions stored in PRSET (m_{PRSET} is user-provided);
 T_{PRSET} - a temporary array of partial rules of the same dimension as PRSET.

FIGURE 1.3 Rule-formatting procedure of RULES-3 Plus.

$$H = \sqrt{\frac{E^c}{E}} \left[2 - 2 \sqrt{\frac{E_i^c}{E^c} \frac{E_i}{E}} - 2 \sqrt{\left(1 - \frac{E_i^c}{E^c}\right) \left(1 - \frac{E_i}{E}\right)} \right] \quad \text{Equation (1.9)}$$

where E^c is the number of examples covered by the expression (the total number of examples correctly classified and misclassified by a given rule), E is the total number of examples, E_i^c is the number of examples covered by the expression and belonging to the target class i (the number of examples correctly classified by a given rule), and E_i is the number of examples in the training set belonging to the target class i . In Equation 1.1, the first term

$$G = \sqrt{\frac{E^c}{E}} \quad \text{Equation (1.10)}$$

relates to the generality of the rule and the second term

$$A = 2 - 2 \sqrt{\frac{E_i^c}{E^c} \frac{E_i}{E}} - 2 \sqrt{\left(1 - \frac{E_i^c}{E^c}\right) \left(1 - \frac{E_i}{E}\right)} \quad \text{Equation (1.11)}$$

indicates its accuracy.

In Steps 3 and 4, by specialising PRSET using the conditions stored in SETAV, the following expressions are formed and stored in T_PRSET:

- 1: [Sensor_3 = 0] \Rightarrow [Alarm = OFF] H = 0.2565
- 2: [Sensor_2 = -1] \Rightarrow [Alarm = OFF] H = 0.0113
- 3: [Sensor_1 = 0] \Rightarrow [Alarm = OFF] H = 0.0012

In Step 5, a rule is produced as the first expression in T_PRSET applies to only one class:

Rule 1: If [Sensor_3 = 0] Then [Alarm = OFF], H = 0.2565

Rule 1 can classify examples 2 and 7 in addition to example 1. Therefore, these examples are marked as classified and the induction proceeds.

In the second iteration, example 3 is considered. T_PRSET, formed in Step 4 after specialising the initial PRSET, now consists of the following expressions:

- 1: [Sensor_3 = 1] \Rightarrow [Alarm = ON] H = 0.0406
- 2: [Sensor_2 = -1] \Rightarrow [Alarm = ON] H = 0.0079
- 3: [Sensor_1 = 1] \Rightarrow [Alarm = ON] H = 0.0005

As none of the expressions cover only one class, T_PRSET is copied into PRSET (Step 5) and the new PRSET has to be specialised further by appending the existing expressions with conditions from SETAV. Therefore, the procedure returns to Step 3 for a new pass. The new T_PRSET formed at the end of Step 4 contains the following three expressions:

- 1: [Sensor_2 = -1] [Sensor_3 = 1] \Rightarrow [Alarm = ON] H = 0.3876
- 2: [Sensor_1 = 1] [Sensor_3 = 1] \Rightarrow [Alarm = ON] H = 0.0534
- 3: [Sensor_1 = 1] [Sensor_2 = -1] \Rightarrow [Alarm = ON] H = 0.0008

As the first expression applies to only one class, the following rule is obtained:

Rule 2: If [Sensor_2 = -1] AND [Sensor_3 = 1] Then [Alarm = ON], H = 0.3876.

Rule 2 can classify examples 3 and 8, which again are marked as classified.

In the third iteration, example 4 is used to obtain the next rule:

Rule 3: If [Sensor_2 = 0] Then [Alarm = OFF], H = 0.2565.

This rule can classify examples 4 and 5 and so they are also marked as classified.

In iteration 4, the last unclassified example 6 is employed for rule extraction, yielding

Rule 4: If [Sensor_2 = 1] Then [Alarm = ON], H = 0.2741.

There are no remaining unclassified examples in the example set and the procedure terminates at this point.

Due to its requirement for a set of examples in a rigid format (with known attributes and of known classes), inductive learning has found rather limited applications in engineering, as not many engineering problems can be described in terms of such a set of examples. Another reason for the paucity of applications is that inductive learning is generally more suitable for problems where attributes have

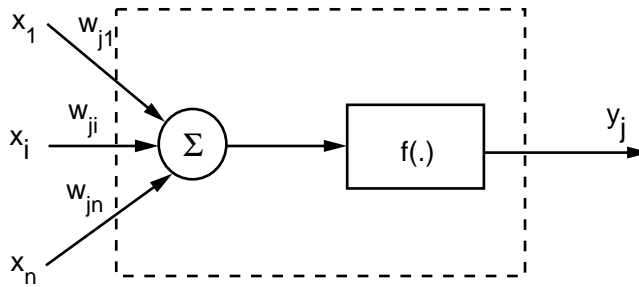


FIGURE 1.4 Model of a neuron.

discrete or symbolic values than for those with continuous-valued attributes as in many engineering problems. Some recent examples of applications of inductive learning are

- Controlling a laser cutting robot [Luzeaux, 1994]
- Controlling the functional electrical stimulation of spinally injured humans [Kostov et al., 1995]
- Classifying complex and noisy patterns [Pham and Aksoy, 1995a]
- Analysing the constructability of a beam in a reinforced-concrete frame [Skibniewski et al., 1997]

1.5 Neural Networks

Like inductive learning programs, neural networks can capture domain knowledge from examples. However, they do not archive the acquired knowledge in an explicit form such as rules or decision trees and they can readily handle both continuous and discrete data. They also have a good generalisation capability, as with fuzzy expert systems.

A neural network is a computational model of the brain. Neural network models usually assume that computation is distributed over several simple units called neurons that are interconnected and operate in parallel (hence, neural networks are also called parallel-distributed-processing systems or connectionist systems). Figure 1.4 illustrates a typical model of a neuron. Output signal y_j is a function f of the sum of weighted input signals x_i . The activation function f can be a linear, simple threshold, sigmoidal, hyperbolic tangent or radial basis function. Instead of being deterministic, f can be a probabilistic function, in which case y_j will be a binary quantity, for example, +1 or -1. The net input to such a stochastic neuron — that is, the sum of weighted input signals x_i — will then give the probability of y_j being +1 or -1.

How the interneuron connections are arranged and the nature of the connections determine the structure of a network. How the strengths of the connections are adjusted or trained to achieve a desired overall behaviour of the network is governed by its learning algorithm. Neural networks can be classified according to their structures and learning algorithms.

In terms of their structures, neural networks can be divided into two types: feedforward networks and recurrent networks. *Feedforward networks* can perform a static mapping between an input space and an output space: the output at a given instant is a function only of the input at that instant. The most popular feedforward neural network is the multi-layer perceptron (MLP): all signals flow in a single direction from the input to the output of the network. Figure 1.5 shows an MLP with three layers: an input layer, an output layer, and an intermediate or hidden layer. Neurons in the input layer only act as buffers for distributing the input signals x_i to neurons in the hidden layer. Each neuron j in the hidden layer operates according to the model of Figure 1.4. That is, its output y_j is given by

$$y_j = f(\sum w_{ji}x_i) \quad \text{Equation (1.12)}$$

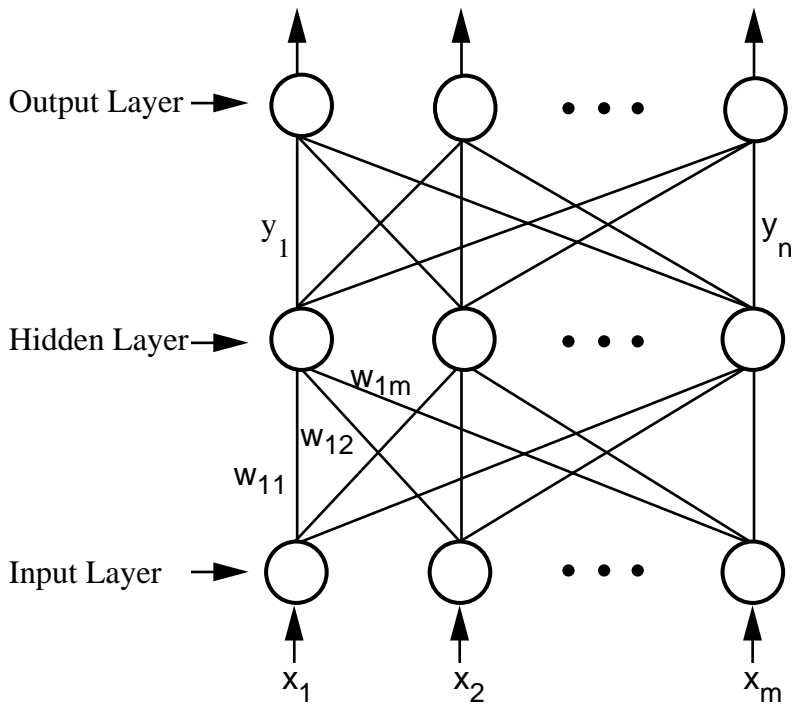


FIGURE 1.5 A multi-layer perceptron.

The outputs of neurons in the output layer are computed similarly.

Other feedforward networks [Pham and Liu, 1999] include the learning vector quantisation (LVQ) network, the cerebellar model articulation control (CMAC) network, and the group-method of data handling (GMDH) network.

Recurrent networks are networks where the outputs of some neurons are fed back to the same neurons or to neurons in layers before them. Thus signals can flow in both forward and backward directions. Recurrent networks are said to have a dynamic memory: the output of such networks at a given instant reflects the current input as well as previous inputs and outputs. Examples of recurrent networks [Pham and Liu, 1999] include the Hopfield network, the Elman network and the Jordan network. Figure 1.6 shows a well-known, simple recurrent neural network, the Grossberg and Carpenter ART-1 network. The network has two layers, an input layer and an output layer. The two layers are fully interconnected, the connections are in both the forward (or bottom-up) direction and the feedback (or top-down) direction. The vector W_i of weights of the bottom-up connections to an output neuron i forms an exemplar of the class it represents. All the W_i vectors constitute the long-term memory of the network. They are employed to select the winning neuron, the latter again being the neuron whose W_i vector is most similar to the current input pattern. The vector V_i of the weights of the top-down connections from an output neuron i is used for *vigilance* testing, that is, determining whether an input pattern is sufficiently close to a stored exemplar. The vigilance vectors V_i form the short-term memory of the network. V_i and W_i are related in that W_i is a normalised copy of V_i , viz.

$$W_i = \frac{V_i}{\varepsilon + \sum V_{ji}} \quad \text{Equation (1.13)}$$

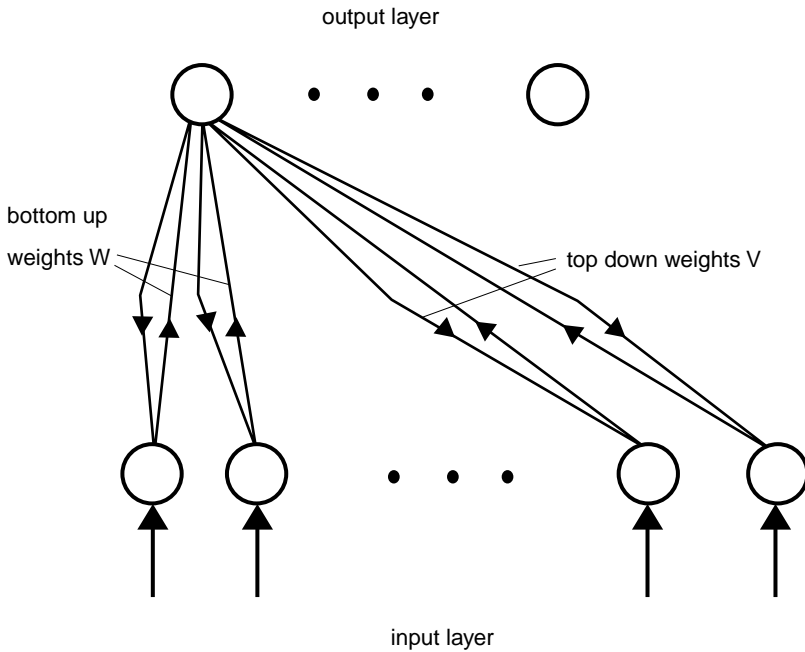


FIGURE 1.6 An ART-1 network.

where ε is a small constant and V_{ji} , the j th component of V_i (i.e., the weight of the connection from output neuron i to input neuron j).

Implicit “knowledge” is built into a neural network by training it. Neural networks are trained and categorised according to two main types of learning algorithms: supervised and unsupervised. In addition, there is a third type, reinforcement learning, which is a special case of supervised learning. In supervised training, the neural network can be trained by being presented with typical input patterns and the corresponding expected output patterns. The error between the actual and expected outputs is used to modify the strengths, or weights, of the connections between the neurons. The backpropagation (BP) algorithm, a gradient descent algorithm, is the most commonly adopted MLP training algorithm. It gives the change Δw_{ji} in the weight of a connection between neurons i and j as follows:

$$\Delta w_{ji} = \eta \delta_j x_i \quad \text{Equation (1.14)}$$

where η is a parameter called the learning rate and δ_j is a factor depending on whether neuron j is an output neuron or a hidden neuron. For output neurons,

$$\delta_j = \left(\frac{\partial f}{\partial \text{net}_j} \right) (y_j^{(t)} - y_j) \quad \text{Equation (1.15)}$$

and for hidden neurons,

$$\delta_j = \left(\frac{\partial f}{\partial \text{net}_j} \right) \sum_q w_{qj} \delta_q \quad \text{Equation (1.16)}$$

In Equation 1.15, net_j is the total weighted sum of input signals to neuron j and $y_j^{(t)}$ is the target output for neuron j .

As there are no target outputs for hidden neurons, in Equation 1.16, the difference between the target and actual output of a hidden neuron j is replaced by the weighted sum of the δ_q terms already obtained for neurons q connected to the output of j . Thus, iteratively, beginning with the output layer, the δ term is computed for neurons in all layers and weight updates determined for all connections. The weight updating process can take place after the presentation of each training pattern (pattern-based training) or after the presentation of the whole set of training patterns (batch training). In either case, a training epoch is said to have been completed when all training patterns have been presented once to the MLP.

For all but the most trivial problems, several epochs are required for the MLP to be properly trained. A commonly adopted method to speed up the training is to add a “momentum” term to Equation 1.14 which effectively lets the previous weight change influence the new weight change, viz.

$$\Delta w_{ji}(k+1) = \eta \delta_j x_i + \mu \Delta w_{ji}(k) \quad \text{Equation (1.17)}$$

where $\Delta w_{ji}(k+1)$ and $\Delta w_{ji}(k)$ are weight changes in epochs $(k+1)$ and (k) , respectively, and μ is the “momentum” coefficient.

Some neural networks are trained in an unsupervised mode where only the input patterns are provided during training and the networks learn automatically to cluster them in groups with similar features. For example, training an ART-1 network involves the following steps:

1. Initialising the exemplar and vigilance vectors W_i and V_i for all output neurons by setting all the components of each V_i to 1 and computing W_i according to Equation 1.13. An output neuron with all its vigilance weights set to 1 is known as an *uncommitted* neuron in the sense that it is not assigned to represent any pattern classes.
2. Presenting a new input pattern x .
3. Enabling all output neurons so that they can participate in the competition for activation.
4. Finding the winning output neuron among the competing neurons, i.e., the neuron for which $x \cdot W_i$ is largest; a winning neuron can be an uncommitted neuron as is the case at the beginning of training or if there are no better output neurons.
5. Testing whether the input pattern x is sufficiently similar to the vigilance vector V_i of the winning neuron. Similarity is measured by the fraction r of bits in x that are also in V_i , viz.

$$r = \frac{x \cdot V_i}{\sum x_i} \quad \text{Equation (1.18)}$$

x is deemed to be sufficiently similar to V_i if r is at least equal to vigilance threshold ρ ($0 < \rho \leq 1$).

6. Going to step 7 if $r \geq \rho$ (i.e., there is *resonance*); else disabling the winning neuron temporarily from further competition and going to step 4, repeating this procedure until there are no further enabled neurons.
7. Adjusting the vigilance vector V_i of the most recent winning neuron by logically ANDing it with x , thus deleting bits in V_i that are not also in x ; computing the bottom-up exemplar vector W_i using the new V_i according to Equation 1.13; activating the winning output neuron.
8. Going to step 2.

The above training procedure ensures that if the same sequence of training patterns is repeatedly presented to the network, its long-term and short-term memories are unchanged (i.e., the network is *stable*). Also, provided there are sufficient output neurons to represent all the different classes, new patterns can always be learned, as a new pattern can be assigned to an uncommitted output neuron if it does not match previously stored exemplars well (i.e., the network is *plastic*).

In reinforcement learning, instead of requiring a teacher to give target outputs and using the differences between the target and actual outputs directly to modify the weights of a neural network, the learning algorithm employs a critic only to evaluate the appropriateness of the neural network output corresponding to a given input. According to the performance of the network on a given input vector, the critic will issue a positive or negative reinforcement signal. If the network has produced an appropriate output, the reinforcement signal will be positive (a reward). Otherwise, it will be negative (a penalty). The intention of this is to strengthen the tendency to produce appropriate outputs and to weaken the propensity for generating inappropriate outputs. Reinforcement learning is a trial-and-error operation designed to maximise the average value of the reinforcement signal for a set of training input vectors. An example of a simple reinforcement learning algorithm is a variation of the associative reward–penalty algorithm [Hassoun, 1995]. Consider a single stochastic neuron j with inputs $(x_1, x_2, x_3, \dots, x_n)$. The reinforcement rule may be written as [Hassoun, 1995]

$$w_{ji}(k + 1) = w_{ji}(k) + l r(k) [y_j(k) - E(y_j(k))] x_i(k) \quad \text{Equation (1.19)}$$

w_{ji} is the weight of the connection between input i and neuron j , l is the learning coefficient, r (which is +1 or -1) is the reinforcement signal, y_j is the output of neuron j , $E(y_j)$ is the expected value of the output, and $x_i(k)$ is the i th component of the k th input vector in the training set. When learning converges, $w_{ji}(k + 1) = w_{ji}(k)$ and so $E(y_j(k)) = y_j(k) = +1$ or -1 . Thus, the neuron effectively becomes deterministic. Reinforcement learning is typically slower than supervised learning. It is more applicable to small neural networks used as controllers where it is difficult to determine the target network output.

For more information on neural networks, see [Hassoun, 1995; Pham and Liu, 1999].

Neural networks can be employed as mapping devices, pattern classifiers, or pattern completers (auto-associative content addressable memories and pattern associators). Like expert systems, they have found a wide spectrum of applications in almost all areas of engineering, addressing problems ranging from modelling, prediction, control, classification, and pattern recognition, to data association, clustering, signal processing, and optimisation. Some recent examples of such applications are:

- Modelling and controlling dynamic systems including robot arms [Pham and Liu, 1999]
- Predicting the tensile strength of composite laminates [Teti and Caprino, 1994]
- Controlling a flexible assembly operation [Majors and Richards, 1995]
- Recognising control chart patterns [Pham and Oztemel, 1996]
- Analysing vibration spectra [Smith et al., 1996]
- Deducing velocity vectors in uniform and rotating flows by tracking the movement of groups of particles [Jambunathan et al., 1997]

1.6 Genetic Algorithms

Conventional search techniques, such as hill-climbing, are often incapable of optimising nonlinear or multimodal functions. In such cases, a random search method is generally required. However, undirected search techniques are extremely inefficient for large domains. A genetic algorithm (GA) is a directed random search technique, invented by Holland [1975], which can find the global optimal solution in complex multidimensional search spaces. A GA is modelled on natural evolution in that the operators it employs are inspired by the natural evolution process. These operators, known as genetic operators, manipulate individuals in a population over several generations to improve their fitness gradually. Individuals in a population are likened to chromosomes and usually represented as strings of binary numbers.

The evolution of a population is described by the “schema theorem” [Holland, 1975; Goldberg, 1989]. A schema represents a set of individuals, i.e., a subset of the population, in terms of the similarity of bits at certain positions of those individuals. For example, the schema 1^*0^* describes the set of individuals

whose first and third bits are 1 and 0, respectively. Here, the symbol “*” means any value would be acceptable. In other words, the values of bits at positions marked “*” could be either 0 or 1. A schema is characterised by two parameters: defining length and order. The defining length is the length between the first and last bits with fixed values. The order of a schema is the number of bits with specified values. According to the schema theorem, the distribution of a schema through the population from one generation to the next depends on its order, defining length and fitness.

GAs do not use much knowledge about the optimisation problem under study and do not deal directly with the parameters of the problem. They work with codes that represent the parameters. Thus, the first issue in a GA application is how to code the problem, i.e., how to represent its parameters. As already mentioned, GAs operate with a population of possible solutions. The second issue is the creation of a set of possible solutions at the start of the optimisation process as the initial population. The third issue in a GA application is how to select or devise a suitable set of genetic operators. Finally, as with other search algorithms, GAs have to know the quality of the solutions already found to improve them further. An interface between the problem environment and the GA is needed to provide this information. The design of this interface is the fourth issue.

1.6.1 Representation

The parameters to be optimised are usually represented in a string form since this type of representation is suitable for genetic operators. The method of representation has a major impact on the performance of the GA. Different representation schemes might cause different performances in terms of accuracy and computation time.

There are two common representation methods for numerical optimisation problems [Michalewicz, 1992]. The preferred method is the binary string representation method. This method is popular because the binary alphabet offers the maximum number of schemata per bit compared to other coding techniques. Various binary coding schemes can be found in the literature, for example, uniform coding, gray scale coding, etc. The second representation method uses a vector of integers or real numbers with each integer or real number representing a single parameter.

When a binary representation scheme is employed, an important step is to decide the number of bits to encode the parameters to be optimised. Each parameter should be encoded with the optimal number of bits covering all possible solutions in the solution space. When too few or too many bits are used, the performance can be adversely affected.

1.6.2 Creation of Initial Population

At the start of optimisation, a GA requires a group of initial solutions. There are two ways of forming this initial population. The first consists of using randomly produced solutions created by a random number generator, for example. This method is preferred for problems about which no *a priori* knowledge exists or for assessing the performance of an algorithm.

The second method employs *a priori* knowledge about the given optimisation problem. Using this knowledge, a set of requirements is obtained and solutions that satisfy those requirements are collected to form an initial population. In this case, the GA starts the optimisation with a set of approximately known solutions, and therefore convergence to an optimal solution can take less time than with the previous method.

1.6.3 Genetic Operators

The flowchart of a simple GA is given in [Figure 1.7](#). There are basically four genetic operators: selection, crossover, mutation, and inversion. Some of these operators were inspired by nature. In the literature, many versions of these operators can be found. It is not necessary to employ all of these operators in a GA because each operates independently of the others. The choice or design of operators depends on

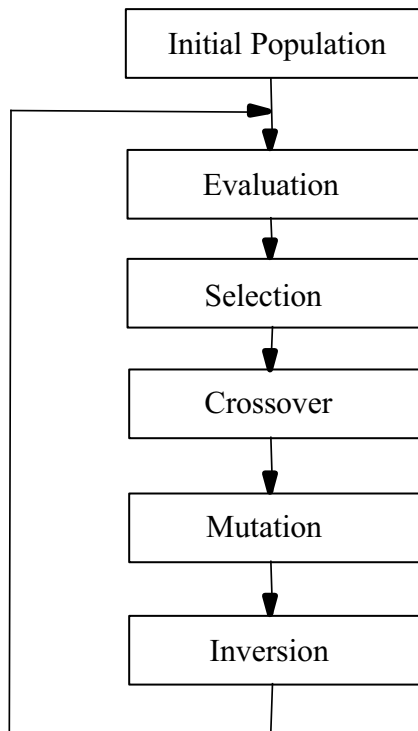


FIGURE 1.7 Flowchart of a basic genetic algorithm.

the problem and the representation scheme employed. For instance, operators designed for binary strings cannot be directly used on strings coded with integers or real numbers.

1.6.3.1 Selection

The aim of the selection procedure is to reproduce more of individuals whose fitness values are higher than those whose fitness values are low. The selection procedure has a significant influence on driving the search toward a promising area and finding good solutions in a short time. However, the diversity of the population must be maintained to avoid premature convergence and to reach the global optimal solution. In GAs there are mainly two selection procedures: proportional selection, also called stochastic selection, and ranking-based selection [Whitley, 1989].

Proportional selection is usually called “roulette wheel” selection, since its mechanism is reminiscent of the operation of a roulette wheel. Fitness values of individuals represent the widths of slots on the wheel. After a random spinning of the wheel to select an individual for the next generation, slots with large widths representing individuals with high fitness values will have a higher chance to be selected.

One way to prevent premature convergence is to control the range of trials allocated to any single individual, so that no individual produces too many offspring. The ranking system is one such alternative selection algorithm. In this algorithm, each individual generates an expected number of offspring based on the rank of its performance and not on the magnitude [Baker, 1985].

1.6.3.2 Crossover

This operation is considered the one that makes the GA different from other algorithms, such as dynamic programming. It is used to create two new individuals (children) from two existing individuals (parents) picked from the current population by the selection operation. There are several ways of doing this. Some common crossover operations are one-point crossover, two-point crossover, cycle crossover, and uniform crossover.

Parent 1	1 0 0 0 1 0 0 1 1 1 1 0
Parent 2	0 0 1 0 1 1 0 0 0 1 1 0
New string 1	1 0 0 0 1 1 0 0 0 1 1 0
New string 2	0 0 1 0 1 0 0 1 1 1 1 0

FIGURE 1.8 Crossover.

Old string	1 1 0 0 0 1 0 1 1 1 0 1
New string	1 1 0 0 1 1 0 1 1 1 0 1

FIGURE 1.9 Mutation.

Old string	1 0 1 1 0 0 1 1 1 0 1
New string	1 0 0 0 1 1 1 1 1 0 1

FIGURE 1.10 Inversion of a binary string segment.

One-point crossover is the simplest crossover operation. Two individuals are randomly selected as parents from the pool of individuals formed by the selection procedure and cut at a randomly selected point. The tails, which are the parts after the cutting point, are swapped and two new individuals (children) are produced. Note that this operation does not change the values of bits. An example of one-point crossover is shown in [Figure 1.8](#).

1.6.3.3 Mutation

In this procedure, all individuals in the population are checked bit by bit and the bit values are randomly reversed according to a specified rate. Unlike crossover, this is a monadic operation. That is, a child string is produced from a single parent string. The mutation operator forces the algorithm to search new areas. Eventually, it helps the GA to avoid premature convergence and find the global optimal solution. An example is given in [Figure 1.9](#).

1.6.3.4 Inversion

This operator is employed for a group of problems, such as the cell placement problem, layout problem, and travelling salesman problem. It also operates on one individual at a time. Two points are randomly selected from an individual, and the part of the string between those two points is reversed (see [Figure 1.10](#)).

1.6.3.5 Control Parameters

Important control parameters of a simple GA include the population size (number of individuals in the population), crossover rate, mutation rate, and inversion rate. Several researchers have studied the effect of these parameters on the performance a GA [Schaffer et al., 1989; Grefenstette, 1986; Fogarty, 1989]. The main conclusions are as follows. A large population size means the simultaneous handling of many solutions and increases the computation time per iteration; however, since many samples from the search space are used, the probability of convergence to a global optimal solution is higher than with a small population size.

The crossover rate determines the frequency of the crossover operation. It is useful at the start of optimisation to discover promising regions in the search space. A low crossover frequency decreases the speed of convergence to such areas. If the frequency is too high, it can lead to saturation around one solution. The mutation operation is controlled by the mutation rate. A high mutation rate introduces high diversity in the population and might cause instability. On the other hand, it is usually very difficult for a GA to find a global optimal solution with too low a mutation rate.

1.6.3.6 Fitness Evaluation Function

The fitness evaluation unit in a GA acts as an interface between the GA and the optimisation problem. The GA assesses solutions for their quality according to the information produced by this unit and not by directly using information about their structure. In engineering design problems, functional requirements are specified to the designer who has to produce a structure that performs the desired functions within predetermined constraints. The quality of a proposed solution is usually calculated depending on how well the solution performs the desired functions and satisfies the given constraints. In the case of a GA, this calculation must be automatic, and the problem is how to devise a procedure that computes the quality of solutions.

Fitness evaluation functions might be complex or simple depending on the optimisation problem at hand. Where a mathematical equation cannot be formulated for this task, a rule-based procedure can be constructed for use as a fitness function or in some cases both can be combined. Where some constraints are very important and cannot be violated, the structures or solutions that do so can be eliminated in advance by appropriately designing the representation scheme. Alternatively, they can be given low probabilities by using special penalty functions.

For further information on genetic algorithms, see [Holland, 1975; Goldberg, 1989; Davis, 1991; Pham and Karaboga, 2000].

Genetic algorithms have found applications in engineering problems involving complex combinatorial or multiparameter optimisation. Some recent examples of those applications follow:

- Configuring transmission systems [Pham and Yang, 1993]
- Generating hardware description language programs for high-level specification of the function of programmable logic devices [Seals and Whapshott, 1994]
- Designing the knowledge base of fuzzy logic controllers [Pham and Karaboga, 1994]
- Planning collision-free paths for mobile and redundant robots [Ashiru et al., 1995; Wilde and Shellwat, 1997; Nearchou and Aspragathos, 1997]
- Scheduling the operations of a job shop [Cho et al., 1996; Drake and Choudhry, 1997]

1.7 Some Applications in Engineering and Manufacture

This section briefly reviews five engineering applications of the aforementioned computational intelligence tools.

1.7.1 Expert Statistical Process Control

Statistical process control (SPC) is a technique for improving the quality of processes and products through closely monitoring data collected from those processes and products and using statistically based tools such as control charts.

XPC is an expert system for facilitating and enhancing the implementation of statistical process control [Pham and Oztemel, 1996]. A commercially available shell was employed to build XPC. The shell allows a hybrid rule-based and pseudo object-oriented method of representing the standard SPC knowledge and process-specific diagnostic knowledge embedded in XPC. The amount of knowledge involved is extensive, which justifies the adoption of a knowledge-based systems approach.

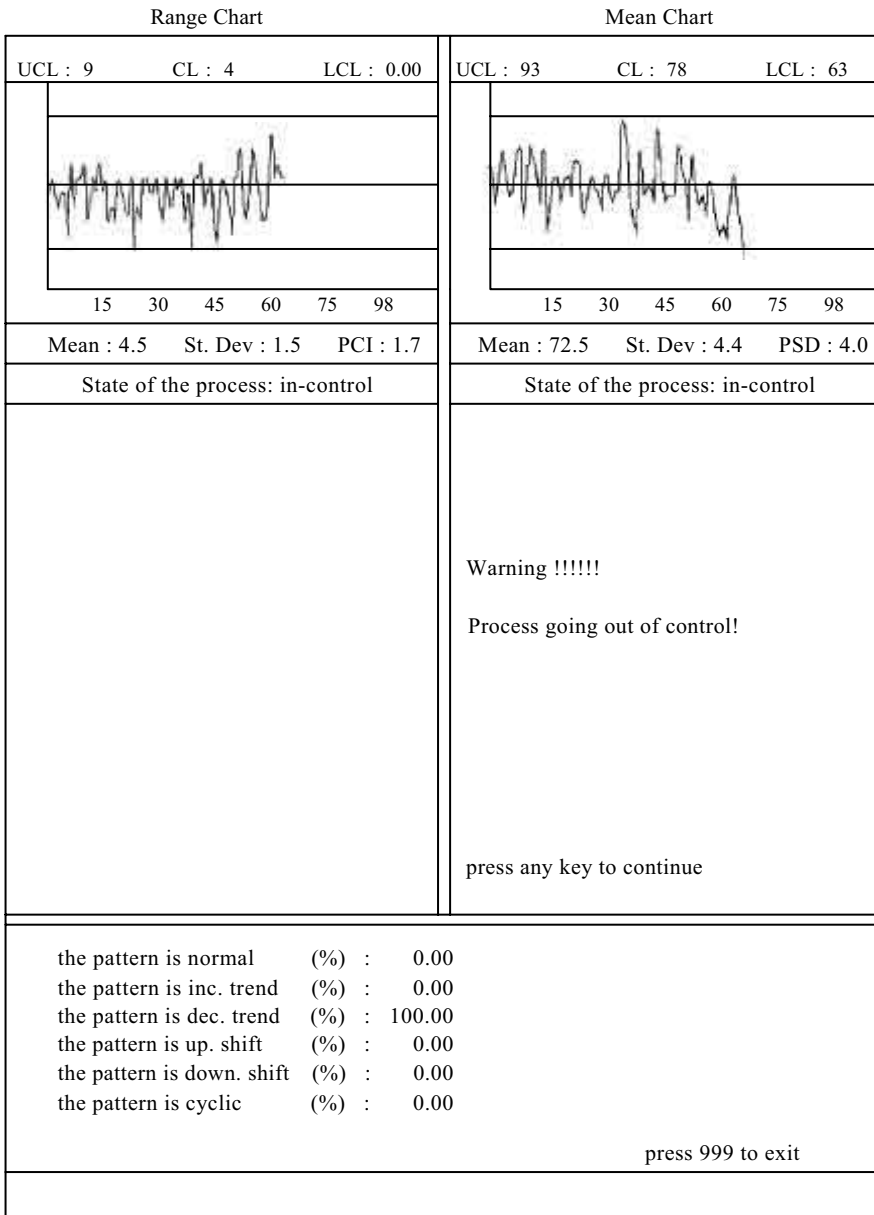


FIGURE 1.11 XPC output screen.

XPC comprises four main modules. The *construction module* is used to set up a control chart. The *capability analysis module* is for calculating process capability indices. The *on-line interpretation and diagnosis module* assesses whether the process is in control and determines the causes for possible out-of-control situations. It also provides advice on how to remedy such situations. The *modification module* updates the parameters of a control chart to maintain true control over a time-varying process. XPC has been applied to the control of temperature in an injection molding machine producing rubber seals. It has recently been enhanced by integrating a neural network module with the expert system modules to detect abnormal patterns in the control chart (see [Figure 1.11](#)).

1.7.2 Fuzzy Modelling of a Vibratory Sensor for Part Location

Figure 1.12 shows a six-degree-of-freedom vibratory sensor for determining the coordinates of the centre of mass (x_G, y_G) and orientation γ of bulky rigid parts. The sensor is designed to enable a robot to pick up parts accurately for machine feeding or assembly tasks. The sensor consists of a rigid platform (P) mounted on a flexible column (C). The platform supports one object (O) to be located at a time. O is held firmly with respect to P. The static deflections of C under the weight of O and the natural frequencies of vibration of the dynamic system comprising O, P, and C are measured and processed using a mathematical model of the system to determine x_G, y_G , and γ for O. In practice, the frequency measurements have low repeatability, which leads to inconsistent location information. The problem worsens when γ is in the region 80° to 90° relative to a reference axis of the sensor, because the mathematical model becomes ill-conditioned. In this “ill-conditioning” region, an alternative to using a mathematical model to compute γ is to adopt an experimentally derived fuzzy model. Such a fuzzy model has to be obtained for each specific object through calibration.

A possible calibration procedure involves placing the object at different positions (x_G, y_G) and orientations γ and recording the periods of vibration T of the sensor. Following calibration, fuzzy rules relating x_G, y_G and T to γ could be constructed to form a fuzzy model of the behaviour of the sensor for the given object. A simpler fuzzy model is achieved by observing that x_G and y_G only affect the reference level of T and, if x_G and y_G are employed to define that level, the trend in the relationship between T and γ is the same regardless of the position of the object. Thus, a simplified fuzzy model of the sensor consists of rules such as “If $(T - T_{ref})$ is small Then $(\gamma - \gamma_{ref})$ is small” where T_{ref} is the value of T when the object is at position (x_G, y_G) and orientation γ_{ref} . γ_{ref} could be chosen as 80° , the point at which the fuzzy model is to replace the mathematical model. T_{ref} could be either measured experimentally or computed from the mathematical model. To counteract the effects of the poor repeatability of period measurements, which are particularly noticeable in the “ill-conditioning” region, the fuzzy rules are modified so that they take into account the variance in T . An example of a modified fuzzy rule is “If $(T - T_{ref})$ is small and σ_T is small, Then $(\gamma - \gamma_{ref})$ is small.” In this rule, σ_T denotes the standard deviation in the measurement of T .

Fuzzy modelling of the vibratory sensor is detailed in Pham and Hafeez (1992). Using a fuzzy model, the orientation γ can be determined to $\pm 2^\circ$ accuracy in the region 80° to 90° . The adoption of fuzzy logic in this application has produced a compact and transparent model from a large amount of noisy experimental data.

1.7.3 Induction of Feature Recognition Rules in a Geometric Reasoning System for Analysing 3D Assembly Models

Pham et al. (1999) have described a concurrent engineering approach involving generating assembly strategies for a product directly from its 3D CAD model. A feature-based CAD system is used to create assembly models of products. A geometric reasoning module extracts assembly oriented data for a product from the CAD system after creating a virtual assembly tree that identifies the components and subassemblies making up the given product (Figure 1.13(a)). The assembly information extracted by the module includes placement constraints and dimensions used to specify the relevant position of a given component or subassembly; geometric entities (edges, surfaces, etc.) used to constrain the component or subassembly; and the parents and children of each entity employed as a placement constraint. An example of the information extracted is shown in Figure 1.13(b).

Feature recognition is applied to the extracted information to identify each feature used to constrain a component or subassembly. The rule-based feature recognition process has three possible outcomes

1. The feature is recognised as belonging to a unique class.
2. The feature shares attributes with more than one class (see Figure 1.13(c)).
3. The feature does not belong to any known class.

Cases 2 and 3 require the user to decide the correct class of the feature and the rule base to be updated. The updating is implemented via a rule induction program. The program employs RULES-3 Plus, which

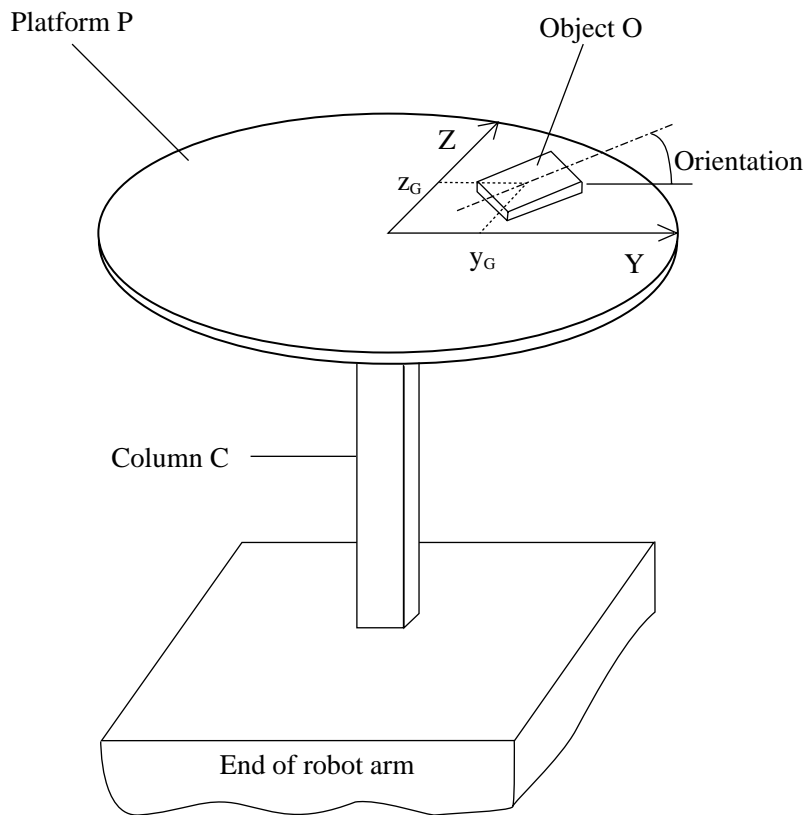


FIGURE 1.12 Schematic diagram of a vibratory sensor mounted on a robot wrist.

automatically extracts new feature recognition rules from examples provided to it in the form of characteristic vectors representing different features and their respective class labels.

Rule induction is very suitable for this application because of the complexity of the characteristic vectors and the difficulty of defining feature classes manually.

1.7.4 Neural-Network-Based Automotive Product Inspection

Figure 1.14 depicts an intelligent inspection system for engine valve stem seals [Pham and Oztemel, 1996]. The system comprises four CCD cameras connected to a computer that implements neural-network-based algorithms for detecting and classifying defects in the seal lips. Faults on the lip aperture are classified by a multilayer perceptron. The inputs to the network are a 20-component vector, where the value of each component is the number of times a particular geometric feature is found on the aperture being inspected. The outputs of the network indicate the type of defect on the seal lip aperture. A similar neural network is used to classify defects on the seal lip surface. The accuracy of defect classification in both perimeter and surface inspection is in excess of 80%. Note that this figure is not the same as that for the accuracy in detecting defective seals, that is, differentiating between good and defective seals. The latter task is also implemented using a neural network which achieves an accuracy of almost 100%. Neural networks are necessary for this application because of the difficulty of describing precisely the various types of defects and the differences between good and defective seals. The neural networks are able to learn the classification task automatically from examples.

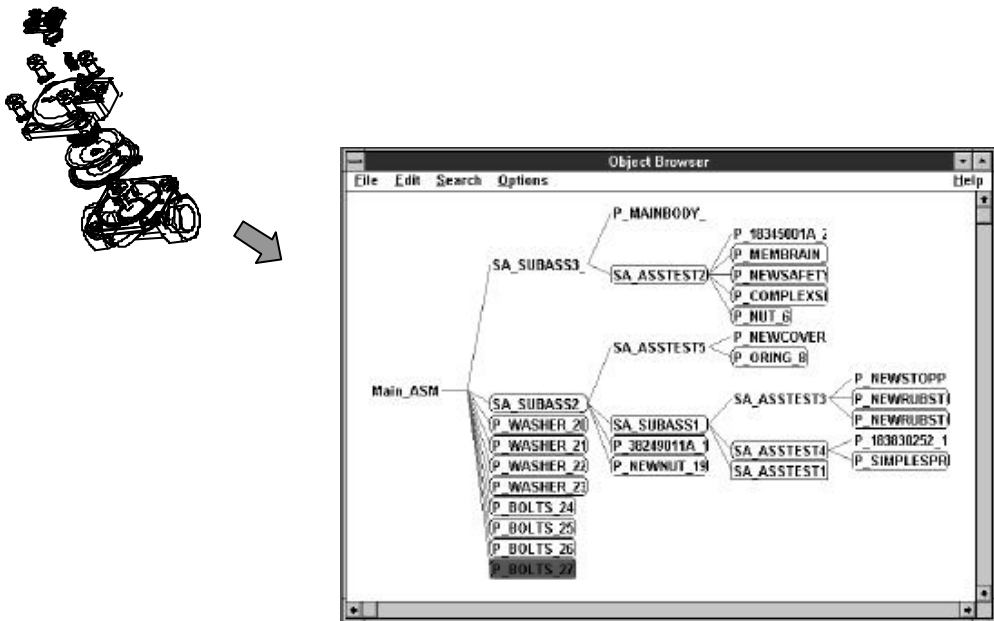


FIGURE 1.13(a) An assembly model.

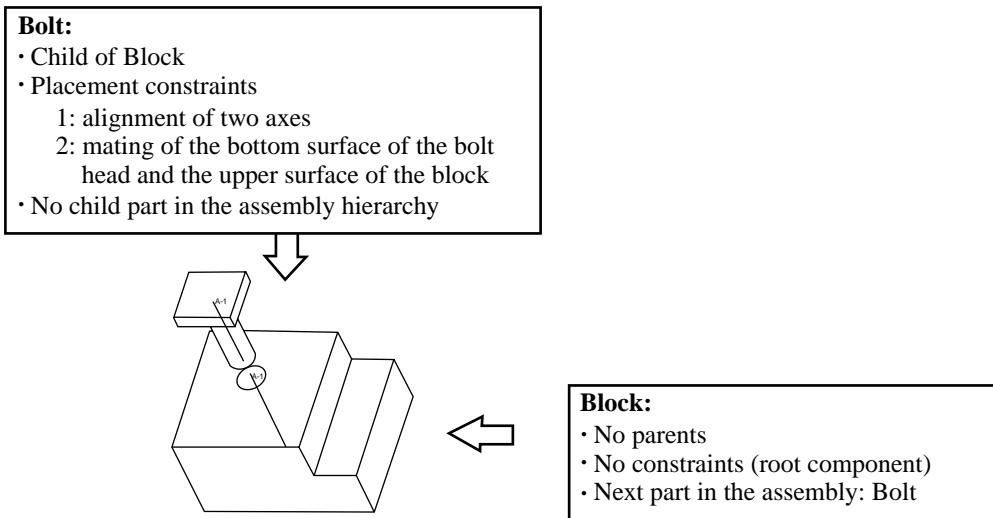


FIGURE 1.13(b) An example of assembly information.

1.7.5 GA-Based Conceptual Design

TRADES is a system using GA techniques to produce conceptual designs of transmission units [Pham and Yang, 1992]. The system has a set of basic building blocks, such as gear pairs, belt drives, and mechanical linkages, and generates conceptual designs to satisfy given specifications by assembling the building blocks into different configurations. The crossover, mutation, and inversion operators of the

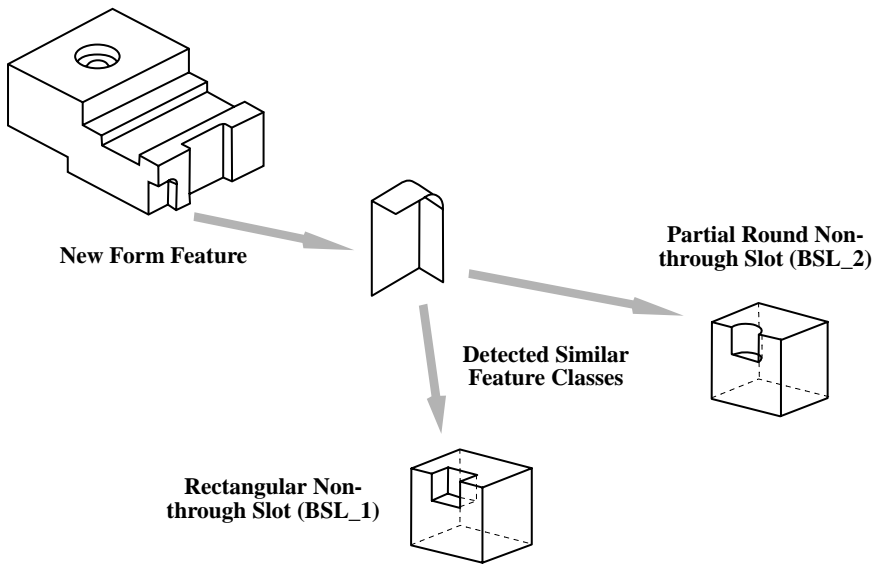


FIGURE 1.13(c) An example of feature recognition.

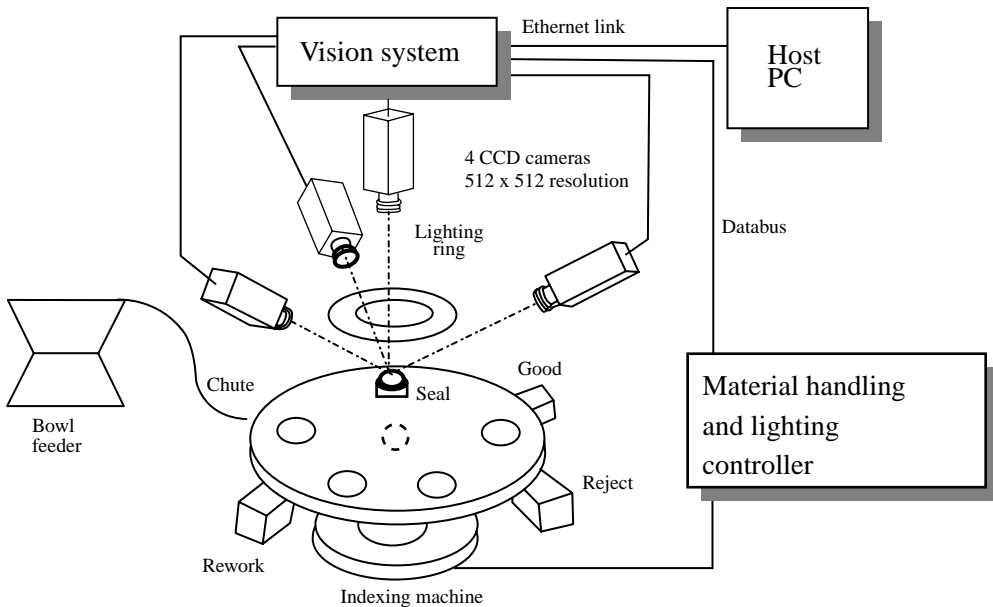


FIGURE 1.14 Valve stem seal inspection system.

GA are employed to create new configurations from an existing population of configurations. Configurations are evaluated for their compliance with the design specifications. Potential solutions should provide the required speed reduction ratio and motion transformation while not containing incompatible building blocks or exceeding specified limits on the number of building blocks to be adopted. A fitness function codifies the degree of compliance of each configuration. The maximum fitness value is assigned to configurations that satisfy all functional requirements without violating any constraints. As in a standard GA, information concerning the fitness of solutions is employed to select solutions for

reproduction, thus guiding the process toward increasingly fitter designs as the population evolves. In addition to the usual GA operators, TRADES incorporates new operators to avert premature convergence to nonoptimal solutions and facilitate the generation of a variety of design concepts. Essentially, these operators reduce the chances of any one configuration or family of configurations dominating the solution population by avoiding crowding around very fit configurations and preventing multiple copies of a configuration, particularly after it has been identified as a potential solution.

TRADES is able to produce design concepts from building blocks without requiring much additional *a priori* knowledge. The manipulation of the building blocks to generate new concepts is carried out by the GA in a stochastic but guided manner. This enables good conceptual designs to be found without the need to search the design space exhaustively.

Due to the very large size of the design space and the quasi random operation of the GA, novel solutions not immediately evident to a human designer are sometimes generated by TRADES. On the other hand, impractical configurations could also arise. TRADES incorporates a number of heuristics to filter out such design proposals.

1.8 Conclusion

Over the past 40 years, computational intelligence has produced a number of powerful tools. This chapter has reviewed five of those tools, namely, knowledge-based systems, fuzzy logic, inductive learning, neural networks, and genetic algorithms. Applications of the tools in engineering and manufacture have become more widespread due to the power and affordability of present-day computers. It is anticipated that many new applications will emerge and that, for demanding tasks, greater use will be made of hybrid tools combining the strengths of two or more of the tools reviewed here [Medsker, 1995]. Other technological developments in computational intelligence that will have an impact in engineering include data mining, or the extraction of information and knowledge from large databases [Limb and Meggs, 1994], and multi-agent systems, or distributed self-organising systems employing entities that function autonomously in an unpredictable environment concurrently with other entities and processes [Wooldridge and Jennings, 1994; Rzevski, 1995; Márkus et al., 1996; Tharumarajah et al., 1996; Bento and Feijó, 1997]. The appropriate deployment of these new computational intelligence tools and of the tools presented in this chapter will contribute to the creation of more competitive engineering systems.

Acknowledgments

This review was carried out as part of the *Innovation in Manufacturing* and *Innovative Technologies for Effective Enterprises* projects supported by the European Regional Development Fund. The Fund is administered by the Welsh Assembly, the Welsh Office, and the Welsh European Programme Executive for the European Commission. Information for the review was also obtained from work on the *INFOMAN* project and the *CONFLOW* project funded by the European Commission under the ESPRIT and INCO-COPERNICUS Programmes.

References

- Ashiru I., Czanecki C. and Routen T., (1995), Intelligent operators and optimal genetic-based path planning for mobile robots, *Proc. Int. Conf. Recent Advances in Mechatronics*, Istanbul, Turkey, August, 1018-1023.
- Baker J. E., (1985), Adaptive selection methods for genetic algorithms, *Proc. First Int. Conf. Genetic Algorithms and Their Applications*, Pittsburgh, PA, 101-111.
- Bas K. and Erkmen A. M., (1995), Fuzzy preshape and reshape control of Anthrobot-III 5-fingered robot hand, *Proc. Int. Conf. Recent Advances in Mechatronics*, Istanbul, Turkey, August, 673-677.
- Bento J. and Feijó B., (1997), An agent-based paradigm for building intelligent CAD systems, *Artificial Intelligence in Engineering*, 11(3), 231-244.

- Bigand A., Goureau P. and Kalemkarian J., (1994), Fuzzy control of a welding process, *Proc. IMACS Int. Symp. Signal Processing, Robotics and Neural Networks (SPRANN 94)*, Villeneuve d'Ascq, France, April, 379-342.
- Bose A., Gini M. and Riley D., (1997), A case-based approach to planar linkage design, *Artificial Intelligence in Engineering*, 11(2), 107-119.
- Cho B. J., Hong S. C. and Okoma S., (1996), Job shop scheduling using genetic algorithm, *Proc. 3rd World Congress Expert Systems*, Seoul, Korea, February, 351-358.
- Da Rocha Fernandes A. M. and Cid Bastos R., (1996), Fuzzy expert systems for qualitative analysis of minerals, *Proc. 3rd World Congress Expert Systems*, Seoul, Korea, February, 673-680.
- Davis L., (1991), *Handbook of Genetic Algorithms*, Van Nostrand, New York.
- Drake P. R. and Choudhry I. A., (1997), From apes to schedules, *Manufacturing Engineer*, 76 (1), 43-45.
- Durkin J., (1994), *Expert Systems Design and Development*, Macmillan, New York.
- Ferreiro Garcia R., (1994), FAM rule as basis for poles shifting applied to the roll control of an aircraft, *Proc. IMACS Int. Symp. Signal Processing, Robotics and Neural Networks (SPRANN 94)*, 375-378.
- Fogarty T. C., (1989), Varying the probability of mutation in the genetic algorithm, *Proc. Third Int. Conf. on Genetic Algorithms and Their Applications*, George Mason University, 104-109.
- Goldberg D. E., (1989), *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison-Wesley, Reading, MA.
- Grefenstette J. J., (1986), Optimization of control parameters for genetic algorithms, *IEEE Trans. on Systems, Man and Cybernetics*, 16(1), 122-128.
- Hassoun M. H., (1995), *Fundamentals of Artificial Neural Networks*, MIT Press, Cambridge, MA.
- Holland J. H., (1975), *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI.
- Jackson P., (1999), *Introduction to Expert Systems*, 3rd ed., Addison-Wesley, Harlow, Essex, U.K.
- Jambunathan K., Fontama V. N., Hartle S. L. and Ashforth-Frost S., (1997), Using ART 2 networks to deduce flow velocities, *Artificial Intelligence in Engineering*, 11(2), 135-141.
- Kaufmann A., (1975), *Introduction to the Theory of Fuzzy Subsets*, vol. 1, Academic Press, New York.
- Koo D. Y. and Han S. H., (1996), Application of the configuration design methods to a design expert system for paper feeding mechanism, *Proc. 3rd World Congress on Expert Systems*, Seoul, Korea, February, 49-56.
- Kostov A., Andrews B., Stein R. B., Popovic D. and Armstrong W. W., (1995), Machine learning in control of functional electrical stimulation systems for locomotion, *IEEE Trans. Biomedical Engineering*, 44(6), 541-551.
- Lara Rosano F., Kemper Valverde N., De La Paz Alva C. and Alcántara Zavala J., (1996), Tutorial expert system for the design of energy cogeneration plants, *Proc. 3rd World Congress on Expert Systems*, Seoul, Korea, February, 300-305.
- Limb P. R. and Meggs G. J., (1994), Data mining tools and techniques, *British Telecom Technology Journal*, 12(4), 32-41.
- Luzeaux D., (1994), Process control and machine learning: rule-based incremental control, *IEEE Trans. Automatic Control*, 39(6), 1166-1171.
- Majors M. D. and Richards R. J., (1995), A topologically-evolving neural network for robotic flexible assembly control, *Proc. Int. Conf. Recent Advances in Mechatronics*, Istanbul, Turkey, August, 894-899.
- Márkus A., Kis T., Váncza J. and Monostori L., (1996), A market approach to holonic manufacturing, *CIRP Annals*, 45(1), 433-436.
- Medsker L. R., (1995), *Hybrid Intelligent Systems*, Kluwer, Boston.
- Michalewicz Z., (1992), *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin.
- Michalski R. S., (1990), A theory and methodology of inductive learning, in *Readings in Machine Learning*, Eds. Shavlik, J. W. and Dietterich T. G., Kaufmann, San Mateo, CA, 70-95.
- Muggleton S. (Ed.), (1992), *Inductive Logic Programming*, Academic Press, London.

- Nearchou A. C. and Aspragathos N. A., (1997), A genetic path planning algorithm for redundant articulated robots, *Robotica*, 15(2), 213-224.
- Peers S. M. C., Tang M. X. and Dharmavasan S., (1994), A knowledge-based scheduling system for offshore structure inspection, *Artificial Intelligence in Engineering IX (AIEng 9)*, Eds. Rzevski G., Adey R. A. and Russell D. W., Computational Mechanics, Southampton, 181-188.
- Pham D. T. and Aksoy M. S., (1994), An algorithm for automatic rule induction, *Artificial Intelligence in Engineering*, 8, 277-282.
- Pham D. T. and Aksoy M. S., (1995a), RULES : A rule extraction system, *Expert Systems Applications*, 8, 59-65.
- Pham D. T. and Aksoy M. S., (1995b), A new algorithm for inductive learning, *Journal of Systems Engineering*, 5, 115-122.
- Pham D. T. and Dimov S. S. (1997), An efficient algorithm for automatic knowledge acquisition, *Pattern Recognition*, 30 (7), 1137-1143.
- Pham D. T., Dimov S. S. and Setchi R. M. (1999), Concurrent engineering: a tool for collaborative working, *Human Systems Management*, 18, 213-224.
- Pham D. T. and Hafeez K., (1992), Fuzzy qualitative model of a robot sensor for locating three-dimensional objects, *Robotica*, 10, 555-562.
- Pham D. T. and Karaboga D., (1994), Some variable mutation rate strategies for genetic algorithms, *Proc. IMACS Int. Symp. Signal Processing, Robotics and Neural Networks (SPRANN 94)*, 73-96.
- Pham D. T. and Karaboga D., (2000), *Intelligent Optimisation Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*, Springer-Verlag, London.
- Pham D. T. and Liu X., (1999), *Neural Networks for Identification, Prediction and Control*, Springer-Verlag, London.
- Pham D. T., Onder H. H. and Channon P. H., (1996), Ergonomic workplace layout using genetic algorithms, *J. Systems Engineering*, 6(1), 119-125.
- Pham D. T. and Oztemel E., (1996), *Intelligent Quality Systems*, Springer-Verlag, London.
- Pham D. T. and Pham P. T. N., (1988), Expert systems in mechanical and manufacturing engineering, *Int. J. Adv. Manufacturing Technol.*, special issue on knowledge based systems, 3(3), 3-21.
- Pham D. T. and Yang Y., (1993), A genetic algorithm based preliminary design system, *Proc. IMechE, Part D: J Automobile Engineering*, 207, 127-133.
- Price C. J., (1990), *Knowledge Engineering Toolkits*, Ellis Horwood, Chichester, U.K.
- Quinlan J. R., (1983), Learning efficient classification procedures and their applications to chess end games, in *Machine Learning, An Artificial Intelligence Approach*, Eds. Michalski R. S., Carbonell J. G. and Mitchell T. M., Morgan Kaufmann, Palo Alto, CA, 463-482.
- Rzevski G., (1995), Artificial intelligence in engineering: past, present and future, *Artificial Intelligence in Engineering X*, Eds. Rzevski G., Adey R. A. and Tasso C., Computational Mechanics, Southampton, U.K., 3-16.
- Schaffer J. D., Caruana R. A., Eshelman L. J. and Das R., (1989), A study of control parameters affecting on-line performance of genetic algorithms for function optimisation, *Proc. Third Int. Conf. on Genetic Algorithms and Their Applications*, George Mason University, 51-61.
- Seals R. C. and Whapshott G. F., (1994), Design of HDL programmes for digital systems using genetic algorithms, *AI Eng 9 (ibid.)*, 331-338.
- Shi Z. Z., Zhou H. and Wang J., (1997), Applying case-based reasoning to engine oil design, *Artificial Intelligence in Engineering*, 11(2), 167-172.
- Skibniewski M., Arciszewski T. and Lueprasert K., (1997), Constructability analysis: machine learning approach, *ASCE J Computing Civil Eng.*, 12(1), 8-16.
- Smith P., MacIntyre J. and Husein S., (1996), The application of neural networks in the power industry, *Proc. 3rd World Congress on Expert Systems*, Seoul, Korea, February, 321-326.
- Szwarc D., Rajamani D. and Bector C. R., (1997), Cell formation considering fuzzy demand and machine capacity, *Int. J. Advanced Manufacturing Technology*, 13(2), 134-147.

- Tarng Y. S., Tseng C. M. and Chung L. K., (1997), A fuzzy pulse discriminating systems for electrical discharge machining, *Int. J. Machine Tools Manufacture*, 37(4), 511-522.
- Teti R. and Caprino G., (1994), Prediction of composite laminate residual strength based on a neural network approach, *AI Eng 9* (ibid), 81-88.
- Tharumarajah A., Wells A. J. and Nemes L., (1996), Comparison of the bionic, fractal and holonic manufacturing system concepts, *Int. J. Computer Integrated Manufacturing*, 9(3), 217-226.
- Wang L. X. and Mendel M., (1992), Generating fuzzy rules by learning from examples, *IEEE Trans. on Systems, Man and Cybernetics*, 22(6), 1414-1427.
- Whitely D., (1989), The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best, *Proc. Third Int. Conf. on Genetic Algorithms and Their Applications*, George Mason University, 116-123.
- Wilde P. and Shellwat H., (1997), Implementation of a genetic algorithm for routing an autonomous robot, *Robotica*, 15(2), 207-211.
- Wooldridge M. J. and Jennings N. R., (1994), Agent theories, architectures and languages: a survey, *Proc. ECAI 94 Workshop on Agent Theories, Architectures and Languages*, Amsterdam, 1-32.
- Yano H., Akashi T., Matsuoka N., Nakanishi K., Takata O. and Horinouchi N., (1997), An expert system to assist automatic remeshing in rigid plastic analysis, *Toyota Technical Review*, 46(2), 87-92.
- Zadeh L. A., (1965), Fuzzy sets, *Information Control*, 8, 338-353.
- Zimmermann H. J., (1991), *Fuzzy Set Theory and Its Applications*, 2nd ed., Kluwer, Boston.