Expert Systems with Applications 36 (2009) 8000–8012

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Induction machine fault detection using clone selection programming

Zhaohui Gan [a], Ming-Bo Zhao [b], Tommy W.S. Chow [a,*]

[a] Department of Electric Engineering, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong
[b] School of Physics and Electronics Engineering, Shanxi University, China

ARTICLE INFO

Keywords:
Clone selection principle
Programming
Immune system
Classification
Induction machine fault detection

ABSTRACT

A clonal selection programming (CSP)-based fault detection system is developed for performing induction machine fault detection and analysis. Four feature vectors are extracted from power spectra of machine vibration signals. The extracted features are inputs of an CSP-based classifier for fault identification and classification. In this paper, the proposed CSP-based machine fault diagnostic system has been intensively tested with unbalanced electrical faults and mechanical faults operating at different rotating speeds. The proposed system is not only able to detect electrical and mechanical faults correctly, but the rules generated is also very simple and compact and is easy for people to understand, This will be proved to be extremely useful for practical industrial applications.

1. Introduction

In general, induction machines are important and expensive devices in certain industries, such as manufacture, transportation. They play the essential role for industrial success so that the maintenance of them is essential and profitable to most electrical industrial processes. If the lifetime of induction machines was extended, and efficiency of manufacturing lines was improved, it would lead to smaller production expenses and lower prices for the end user. In order to keep machines in good condition, some techniques i.e., fault monitoring, fault detection, and fault diagnosis have become increasingly essential (Isermann, 1997; Leohardt & Ayoubi, 1997; Patton, Frank, & Clark, 1989). Although there are different types of methods used for detecting machine faults (Ye & Wu, 2000), the use of vibration signals for fault detection and diagnostic analysis has widely recognized as a reliable approach.

Most signal processing techniques applied to machine fault diagnostic analysis are in either the time domain or the frequency domain. Although the information about the rotating machine conditions can be obtained through the vibration signal transmitted through the machine casing, it is still a difficult task to determine the machine conditions from the measured vibration signals. Although there are a number of different approaches reported for performing machine fault diagnostic analysis, such as time-domain analysis (Lipovszky et al., 1990; Ragulskis & Yurkauskas, 1989), probabilistic analysis (Lipovszky et al., 1990; Ragulskis & Yurkauskas, 1989), and finite-element analysis (Donley, Stokes, Jeong, Suh, & Jung, 1996), frequency-domain analysis appears to be the most popular and computational non demanding approach for providing the required information, which is totally attributed to the more salient characteristic features in the frequency domain.

Machine fault detection is classified invasive and noninvasive methods (Chow & Methedologies, 1997; Chow, Sharpe, & Hung, 1993). The noninvasive methods are more preferable than the invasive methods because they are based on easily accessible and cheap measurements to diagnose the machine conditions without disintegrating the machine structure. The common types of machine fault diagnostic in the frequency domain include bearing defect frequency analysis (Collacott, 1979; Dimarogonas, 1996), high-frequency shock pulse and friction forces methods (Collacott, 1979; Dimarogonas, 1996), enveloped spectrum methods (Collacott, 1979; Dimarogonas, 1996), high-order-spectrum (HOS) methods (Chow, 2000; Chow & Fei, 1995), and wavelet methods (Tse, Peng, & Yam, 2001; Wang, 2001) etc. Recently, artificial intelligence (AI) techniques have been proposed for the noninvasive machine fault detection (Chow & Methedologies, 1997; Filippetti, Franceschini, Tassoni, & Vas, 2000). They have several advantages over the traditional model-based techniques (Chow & Methedologies, 1997; Filippetti et al., 2000). They require no detailed analysis of the different kinds of faults or modeling of the system. These AI-based techniques include expert systems, neural network, and fuzzy logic etc. An expert system is able to manage knowledge-based production rules that model the physical system (Filippetti et al. 2000). Neural network approaches can be considered as "black-box" methods as they do not provide heuristic reasoning about the fault detection process (Chow & Methedologies, 1997; Chow et al., 1993). Fuzzy logic systems can heuristically implement fault detection principles and heuristically interpret and analyze their results (Chow & Methedologies, 1997). There are a few papers reporting unsupervised neural network as diagnostic preprocessor for classification purpose (Penman & Yin, 1994; Schlang, Habetler,

* Corresponding author.
E-mail address: eetchow@cityu.edu.uk (T.W.S. Chow).

0957-4174/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2008.10.058

& Lin, 1997), while supervised neural network approaches (Alguindigue, Buczak, & Uhrig, 1993; Chow & Mangum, 1991; Filippetti, Franceschini, & Tassoni, 1995) can be used for fault and fault severity classification. The supervised neural networks in Alguindigue et al. (1993), Chow and Mangum (1991), Filippetti et al. (1995) are multilayered networks (MLNs). Coupled with the backpropagation (BP) algorithm, the MLNs fall short for their slow convergence rate and local minimum problem (Er, Wu, Lu, & Toh, 2002). On the other hand, RBF neural networks have drawn extensive interest because of its well-known classification performance compared with MLNs (Er et al., 2002). But it requires the many trial tests to determine the appropriate network architecture, which is not user friendly for general industrial applications. Sitao Wu et al. presented a self-organizing-map (SOM)-based radial-basis-function (RBF) network (Wu & Chow, 2004) for fault diagnostic classification. It can automatically select the centers and the number of hidden neurons of the RBF networks to determine the appropriate network architecture so that machine fault detection accuracy is higher. A great deal of references has proven the success of these neural networks for machine fault detection. But, the generations of these classifiers are difficult for layman to understand and implement. This prevents classifiers from being widely used in most machine fault detection applications.

Genetic Programming (GP), first introduced by Koza (1992), is derived from GA. In the short period of its development, GP has been successfully applied in many fields, including condition monitoring (Kojima, Kubota, & Hashimoto 2001; Helmer et al., 2002; Cheng & Chiu, 2005). In fact, the GP based classifier is a kind of the rule-based classifiers, which can build a set of IF-THEN rules according to a training dataset S whose instance has some attributes and a unique target class label so that the target accuracy for new unseen data can be predicted. So when it is applied into fault detection, it can generate some rules to discriminate whether machine exists fault or not, which will make people analyse and understand the reason of the fault. Liang Zhang et al. applied GP into roller bearing fault detection successfully (Zhang & Lindsay, 2005; Zhang & Nandi, 2006), the experimental results show that the classification rules derived are easily understood and capable of independent verification on other data. However, it is not reported that GP is applied into induction machine fault detection.

Due to the tree type structure, GP can express complex relationship between the observed data using any combinations of logical, mathematical functions of input attributes. Thus, GP can provide a better solution structure adaption to the data compared with GA. Therefore, GP can find more accurate and complex classification rules than GA. But compared with GA, GP has much larger searching space, so it consume more time than GA. In additional, GP cannot keep syntax closure property during evolution. Moreover, GP's tree-based individuals typically result in bloating (Zhang & Muhlenbein, 1995).

The classifier based on GP for two class problems has been successfully elaborated and applied into classification of medical data (Dounias et al., 2002; Falco et al., 2002; Stanhope & Daida, 1998). Although many researchers use genetic algorithm to design classifiers for multi class problems, only a few GP methods have been made to solve multi class problem (Bojarczuk, Lopes, & Freitas, 2000; Chien, Lin, & Hong, 2002; Dounias et al., 2002; Kishore, 2000; Lim, Loh, & Shih, 2000; Mendes, Voznika, Freitas, & Nievola, 2001). Usually, people consider a c class problem as a set of c two-class problems (Kishore, 2000), which is called "binary decomposition" (Brodley & Danyluk, 2001) or "one-against-all learning" (Loveard & Ciesielski, 1983). For a c class problem, the system is run c times, so, the efficiency of the system is not high. For a c class pattern classification problem, another method for designing classifiers is just using a single run of GP. A multi tree classifier having c trees is constructed, where each tree represents a classifier for a particular class. The performance of a multi tree classifier depends on the performance of its corresponding trees. Compared with "one-against-all learning" method, it is direct and demonstrates more high efficient. Muni et al. (2004) have designed a multi tree classifier using GP solve this question successfully.

Artificial immune system is a kind of methodology inspired by the human immune system. Research on artificial immune system (de Castro & Timmis, 2002; Wierzchon, 2002) has become increasingly popular in the area of evolutionary computing. New models of artificial immune system are proposed, and more applied research have been explored, such as computer security, data mining, clustering, pattern recognition and function optimization etc (Cutello, Nicosia, Pavone, & Timmis, 2007; de Castro & Von Zuben, 2000; Watkins, Timmis, & Boggess, 2004). Despite the flourishing of artificial immune system in some areas, there is only a handful of papers (Sahan, Kodaz, Güneş, & Polat, 2004; Watkins et al., 2004) focused on the design of classifier using artificial immune system. The research on immune programming is even fewer, and until now the rule-based classifier using artificial immune system has not been reported in literature.

We have proposed a new programming method, called clone selection programming (CSP) (Gan et al., 2008) based on the theories of the immune system to enhance the effectiveness of programs encoding and search engine. The newly proposed CSP can become a powerful tool applied widely into artificial intelligence and machine learning etc. field. Compared with other approaches based on artificial immune system, clone selection programming (Gan et al., 2008) based method can significantly improve the program performance. In this paper, we extend clone selection programming to the design of classifier for machine fault detection. A kind of master/slave-style parallel computing multi thread model was proposed to improve the performance of evolutionary search of CSP. After the vibration signals are transformed into the frequency domain, four characteristic features vectors are extracted from the power spectra in the frequency domain as inputs of the proposed clone selection programming based classifier. The advantages of the proposed CSP-based classifier are twofold. First, the classifying rules established by using our proposed method are comprehensible and capable of independent verification on other data. In addition, these fault detection rules are more compact than ones established by GP classifier. Second, CSP-based classifier is very easy to implement and does not require the many trial tests to determine the appropriate parameters. This feature is user friendly for general industrial applications.

The experimental results demonstrate that our CSP-based approach is promising for detecting machine faults via monitoring the vibration signals.

The presentation of this paper is organized as follows. In Section 2, we briefly describe how the four characteristic features are extracted from the power spectra after the signals are transformed into the frequency domain. In Section 3, clone selection programming and our proposed classification strategy are detailed. In Section 4, two types of machine faults including "electrical faults," and "mechanical faults" were used for system validation. Finally, conclusions are drawn in Section 5.

2. Preprocessing of vibration signals

Vibration is the best indicator of overall mechanical condition and the earliest indicator of defects developing. Vibration analysis is based on the principle that faults can be detected in characteristic frequencies associated with particular type of faults in the frequency domain. The basic fault detection system in the frequency domain is briefly shown in Fig. 1. First vibration signals are collected from transducer based data acquisition systems. Then the signals are transformed by fast Fourier transform (FFT) into signals

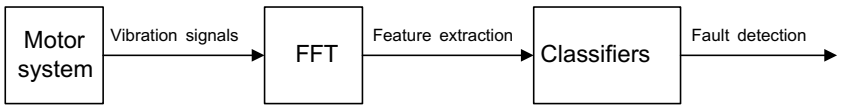


Fig. 1. Flowchart of fault detection system in frequency domain.

in the frequency domain, which can be analyzed and processed easier than those in the time domain. Then some features are extracted from the frequency-domain signals as inputs of classifiers. Through supervised training with inputs and outputs, the learned classifiers can detect different types of faults. As the spectra include many frequency components, usually we need to reduce frequency components into fewer useful features. One method is to extract feature components at characteristic frequencies associated with certain types of faults (Li, Chow, & Tipsuwan, 2000). Another type is to compress the spectrum while nearly maintaining the shape of the spectrum (Alguindigue et al., 1993). The first method requires knowledge of the characteristic frequencies in the type of machine faults involved. But in practical situations, many unexpected conditions can degrade the expected results and make theoretic analysis difficult and inconsistent. The second method attempts to use the shape of the spectra of the vibration signals. But the compression of the spectrum in Alguindigue et al. (1993) is too complicated and time consuming. In our proposed CSP-based classifier approach, we use four features extracted from the power spectra of the vibration signals. When the machines are in faults, the shapes of the frequency distribution of energy drift from the normal condition. The spread (dispersion) of distribution about central value (central moment) gives information about shape or dispersion of the power spectra about its average frequency. The dispersion indices used in this study include normalized second and third-order central moments (Thomas, 1971). By extracting the total power, average frequency, and dispersion indices of the power spectra of vibration signals, we can detect different types of faults. The fundamental definitions of the four features are given as follows:

(1) Total power

$$TP = \int_0^\infty P(f)df. \tag{1}$$

(2) Average frequency

$$\bar{f} = \frac{\int_0^\infty fp(f)df}{TP}. \tag{2}$$

(3) Normalized second-order central moment

$$NU2 = \frac{\int_0^\infty (f-\bar{f})^2 p(f)df}{TP}. \tag{3}$$

(4) Normalized third-order central moment

$$NU3 = \frac{\int_0^\infty (f-\bar{f})^3 p(f)df}{TP}. \tag{4}$$

Therefore, the complete input vector of the CSP classifier is

$$X = \left[TP, \bar{f}, NU2, NU3\right]. \tag{5}$$

3. Clone selection programming and its application in classification

The immune system of human body, which consists of an innate and an adaptive immune system, is a very complex, rapid, and effective defense mechanism against disease. The innate and adaptive immune systems both depend on the activity of a great variety of molecules, cells, and organs spread throughout the body. Various distributed elements incorporate immune functions that do not need central control. The cells of the innate immune system are in-born and exist throughout our body. Once a wide variety of bacteria occur in the body, these cells are immediately available to fight against them. The response produced by an antibody combating a determined infectious agent is called adaptive immune response. The presence of antibodies reflects the kind of infection that our body has already been exposed. Cells of the adaptive system can extinguish the same antigenic stimulus when these antigens attack the body again. The capability of adaptive immune system is called immune memory which enables diseases within human organism be rapidly destroyed. The adaptive immune system is mainly made up of lymphocytes which are responsible for the recognition and elimination of the pathogenic agents. The main function of the immune system is to protect human organism against pathogens and to eliminate malfunctioning self cells. This self-defense function is largely relied on the ability of recognition. The immune system not only recognizes pathogens and malfunctioning cells, it is also able to recognize the organism's own properly functioning cells and tissues in order to prevent them from inadvertent destruction. All elements, pathogens, malfunctioning cells, and healthy cells etc recognized by the immune system are called antigens. The cells which belong to the organism and are harmless to its functioning are termed self, while the harm-causing elements are termed non-self. The immune system has the ability to distinguish which element is self or non-self. In immunology, this process is named self/non-self discrimination. When the immune system finds an antigen and recognizes it as non-self, it generates a response to eliminate the pathogen. But the process of antigen recognition and elimination is not enough on their own to deactivate various pathogens. In order to be better to recognize new pathogens and to improve response to pathogens already encountered, the immune system is provided with memory and an ability to learn from the processes of pattern recognition, clone selection, negative selection, and affinity maturation.

3.1. Antibody encoding

3.1.1. Biological structure of antibody

An antibody, or immunoglobulin (Ig) has four polypeptide chains: two identical light (L) and two identical heavy (H) chain (Tonegawa, 1983) composed of an amino terminal region that is highly variable (variable region) and a carboxi terminal region that can be assumed as one of a few types (constant region). The variable region, or V-region, is responsible for the antigenic recognition. They contain some special variable sub-regions whose composition is a consequence of the contact with an antigen. Moreover, a number of effector functions, such as complement fixation and ligation to other cell receptors of the immune system, are developed by the constant region, or C-region.

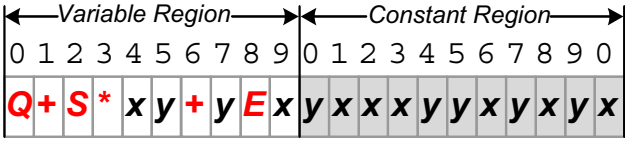
Multiple gene segments scattered along a chromosome of a genome consists of a polypeptide chain of an antibody molecule which means that genes located in several different gene libraries are concatenated to form the heavy and light chains of the antibody molecules. For example, the V-region of heavy and light chains of the antibody molecules is coded by two separated gene

segments named V (for variable) and J (for junction). Beyond the V and J segments, the area between the segments V and J of the heavy chain is the third segment named D which is for diversity. Therefore, antibody population diversity, which is due to the random recombination of gene fragments, is contained in several libraries. The process of somatic hypermutation is used to increase the Ag–Ab (antigen–antibody) affinity so that the immune diversity and capacity of response are improved. The affinity can be understood as the strength of binding between two binding sites, such as a cell receptor and an epitope. Thus, the above two mechanisms of generation and diversification of antibodies makes the immune system capable of synthesizing an almost infinite number of cell receptors from a finite genome.

3.1.2. Expression tree and antibody encoding

In contrast to its analogous antibody structure expression, antibody in CSP consists of a linear, symbolic string of fixed length that composes of one or more genes. Expression of antibody is rather simple. It has two main components: the antibody encoding and the expression trees. The latter is an expression of the immune information encoded in the former. The process of information decoding (from the antibody to the expression tree) is called translation. The immune code is very simple: a one to one relationship between the symbols of the antibody and program or problem they represent. The spatial organization of the function and terminals in the expression trees is determined by the corresponding rules. Therefore, there exist two languages in CSP: the language of the antibodies and the language of the expression trees. It is noted that the two languages are sufficient to infer exactly the other. For example, the algebraic expression $\sqrt{\sin(xy)+y+e^x}$ can also be used for representing an expression tree (ET), where Q represents the square-root function, E represents the exponential function, and S represents the sinusoidal function. The gene can be represented by the expression tree shown in Fig. 2a.

The expression tree as shown in Fig. 2a is, in fact, the phenotype of an antibody gene. An expression string of the above algebraic expression can be translated from the expression tree as follows:



For each sub-tree in the ET, the root must be read out firstly, and then the left child node, the right child node is read out lastly. To complete the translation of the whole expression string to an expression tree, the following rules are used.

(i) The start of expression string corresponds to the root node of the ET as in Fig. 2a.

(ii) If the root node is a terminal, the mapping stops. Or its child branches are processed from left to right. All of the children branches are read from the antibody encoding one by one. If a node is a function having only one argument, only one symbol in the encoding is placed as its child node.If the function has more arguments, add a symbol from the left of the start node. Some symbols in encoding are placed as its child nodes hierarchically until the end is leaf node. The number of child branches is determined by the number of arguments of their parent node. After the left of node has been constructed, the right part of node will be constructed by the same method.

(iii) From left to right, and from top to bottom, new nodes are filled consecutively with the elements of the expression string. This process continues in a way of layer-by-layer until all leaf nodes in the ET are composed of elements from the terminal set.

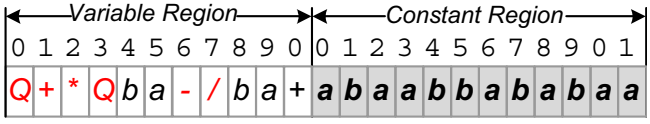
After mapping an antibody encoding into an ET, the affinity of the antibody encoding can be calculated by decoding the ET through traversal. Through various operations, there is no invalid expression or computer program. This encoding scheme makes all programs evolved by CSP syntactically correct. Thus, in CSP, syntactic closure is the intrinsic nature making evolution more efficient. Indeed, this is the paramount difference between other immune programming and other types of GP implementations, which either limit themselves to inefficient genetic operators or checking all the newly created programs exhaustively for syntactic errors.

3.1.3. Structure and functional organization of antibody encoding

The encoding scheme is similar to gene expression encoding in GEP (Ferreira, 2006). The structure of antibody genes composes of two different domains which exhibit different properties and functions. They are a variable region and a constant region. The variable region is used mainly to express the functions chosen for the specific problem, whereas the constant region works as a buffer or reservoir of terminals in order to guarantee the formation of only valid structures. Thus, the variable region contains symbols representing both functions and terminals, whereas the constant region composes of only terminals. For a given problem, the length of the variable region V is chosen, whereas the length of the constant region C is a function of V and the number of arguments of the function with more arguments $n_{\max}$ (also called maximum arity). It is evaluated by:

$$C = V * (n-1) + 1. \tag{6}$$

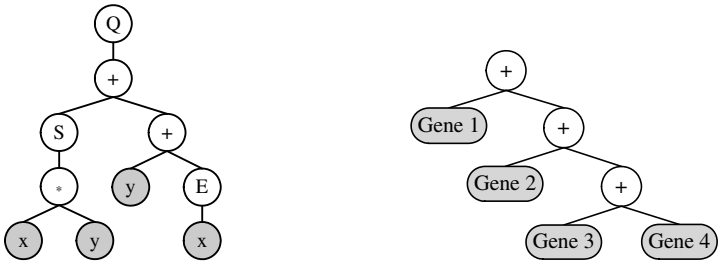
Consider a gene, a set of functions $F = \{Q, *, /, -, +\}$, and a set of terminals $T = \{a, b\}$ are given. Thus, it gives $n_{\max} = 2$, and if we chose an $V = 11$, then $C = 11 \cdot (2-1) + 1 = 12$, the length of the gene is $11 + 12 = 23$. A typical gene is shown below, in which the constant region is shown in bold:



The antibodies usually consist of several genes of equal length. The interaction between the genes is specified by a linking function. An example of a four-gene antibody linked by addition is shown in Fig. 2b.

3.2. Clone selection algorithm

Based on the concept of the clone selection in immune system (Nossal, 1993), the algorithm of CSP is developed. At the initialization stage, antibodies, which are the candidate solutions to a given problem, are randomly generated as an initial population. The cloning and hypermutation operation of the antibody make the population evolve so that the diversity of the population is maintained and the search space for solutions is expanded. Evaluation of the quality of each antibody is based on the affinity value. The higher-affinity antibodies are selected for cloning or hypermutation, while the antibodies with lower affinity are replaced. This selection process finishes through one of the operations of replacement, cloning or mutation to form a new population. The whole process repeats until the stopping criteria or maximal generation number is reached. The



(a) The expression tree of a gene (b) A four-gene antibody linked by addition

Fig. 2. The expression tree of gene and antibody.

final result of search process is then decoded to the program space becoming an implementation of the solution to the given problem. The procedures of CSP can be briefly described as follows:

3.2.1. Initialization

The initial population of antibodies is randomly generated. The encoding structure is introduced in the above section. These N individuals compose of an initial population (P) of candidate solutions which include a subset of memory cell (M);

3.2.2. Evaluation

An antigen (Ag) representing the problem to be solved appears at the initial stage of programming. Different expression forms of antigen depend on the specific problem. Here, we assume that the antigen is taken the form of an arithmetic expression, say, $Ag = x^2 + y^2 + x + y$. To evaluate the affinity of the antibodies, particular values of variable(s) have to be placed on the expression and the programs have to be executed. Since the problem is described in a symbolic form, no numerical argument values are explicitly prescribed and they must be generated. For the antigen used as an example, this corresponds to generation of x and y values. Here, we randomly generate x and y in the range of [0, 255]. Five sets of values x and y are generated to execute each antibody and to compare the execution results to the antigens behavior. The affinity of ith antibody with antigen $f_i = \frac{1}{\frac{1}{n}\sum_{i=1}^{n}(Ab_i - Ag_i)^2 + 1}$, where $Ag_i = x_i^2 + y_i^2 + x_i + y_i$. $Ab_i = f(x_i, y_i)$. $n = 5$. Thus, the affinity of antibody with antigen is in the range of [0, 1]. The whole antibodies are sorted in descending order according to the affinity of all antibodies in the population P with antigen.

3.2.3. Cloning

Before a new antibody has been generated, an antibody Ab_i from the current population is considered for cloning. The antibodies are selected for cloning according to the affinities with antigen. The antibodies in population are sorted in descending order. The n $(n < N)$ highest affinity antibodies will be cloned. The number of clones (given by Eq. (7)) reproduced for each individual is proportional to its affinity.

$$N_c = round\left(\frac{\beta * N}{i}\right), \tag{7}$$

where for each individual, β is a multiplying factor, N_c is the number of clones generated, N is the total number of individuals, i is the index of current individual in the population, and round $(\cdot)$ is the function that rounds its variable toward the closest integer. After these n best individuals are cloned, a temporary clone population (P_C) is generated.

3.2.4. Hypermutation

The individuals in the population P_C of the previous step are submitted to a hypermutation procedure. Suppose an antibody has J genes, while a gene has K bit symbol string, $Ab_i = \langle S_1, S_2, \ldots S_j \rangle$, the mutation process is implemented by replacing some bit symbols of each gene with some new randomly-generated symbols belonging to the defined function set or terminal set. The symbols in variable region may be replaced as function or terminal symbol, whereas the symbols in constant region are only mutated as terminal symbol. The probability of mutation P_m determines this process. To illustrate the process of hypermutation, let us assume that there is an antibody containing three genes, the size of variable region of gene is five, the argument number is two, and the size of constant region of gene is six. This antibody is shown as follow in which the constant regions are shown in bold:

```
0123456789001234567890012345678 90
*x+xyyxxxyx+**xyxyxyxx+y*+xxxyxyy
```

The corresponding arithmetic expression is $f(x,y) = 3x^2 + x^2y + xy + 2y$, and its corresponding expression tree is shown in Fig. 3a. From the first symbol of antibody, a random positive floating number (less than one) is generated. If the number is less than the probability of mutation, P_m, the symbol will mutate. The new symbol that replaces the old one is randomly selected from the function or terminal set. This process repeats until the last symbol of the last gene. The Pseudocode of mutation is shown in Fig. 4. In our example, mutation points of each gene of antibody are highlighted in underline. The mutated antibody is shown as follows:

```
0123456789001234567890012345678 90
*x+x+yyxxxyx+*xxyxyxyxx+y*++xyyxxy
```

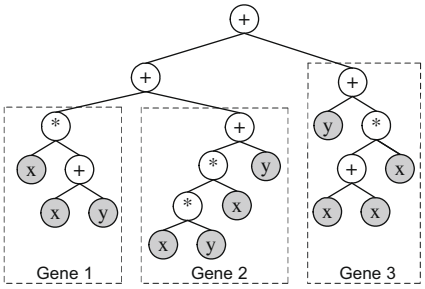
The corresponding arithmetic expression is $f(x,y) = 3x^2 + 4xy + 2y$, and the corresponding expression tree is shown in Fig. 3b. Owing to the hypermutation operator, the algorithm is able to provide new gene material into the antibody population. This can enhance the diversity of antibody population and expand the search space for finding the solution. After hypermutation, an antibody population (P_M) based on clone population (P_C) is generated.

3.2.5. Re-selection

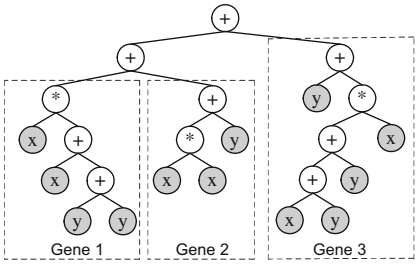
After hypermutation is finished, the individuals in the population P_M are evaluated again so that the individuals with higher affinity are chosen to form the memory cell set M.

3.2.6. Replacement

After the above phases are complete, the algorithm proceeds with generation of new individuals. The new randomly generated



(a) The expression tree of antibody



(b) The expression tree of antibody mutated

Fig. 3. The expression tree of antibody and antibody mutated.

individuals will be put into population directly so that the lower affinity individuals will be replaced with higher probabilities.

Steps (ii)–(vi) iteratively proceed until the stopping criterion is reached. The criterion used in this study takes two forms: maximal number of generations and affinity threshold. At last, the final attribute string is presented and translated into solution of specified problem. The Pseudocode of the whole algorithm is depicted in Fig. 5.

3.3. Classification through clone selection programming

Based on its encoding and powerful global searching ability, this paper shows how clone selection programming CSP [49] is used for induction machine fault detection. In fact, this is a classifier problem, which indicates the potential of CSP in solving complex problems.

In mathematical sense, we define classifier C is a mapping, C: R^n ——L, where R^n is the n-dimensional real space and L is the class label vector. For a c class classification problem, given a set of training pattern vector $X = \{x_1, x_2, \ldots, x_n\} \ni R^n$ and its corresponding class label vector $Y = \{y_1, y_2, \ldots, y_c\} \ni$ L, the classification task is to discover some rules make function $C(x)$ became a c-dimension vector whose component has only one 1 and all others as 0.

In this paper, our objective is to design a classifier using clone selection programming (CSP). Not like "one-against-all learning" method, the CSP-based classifier just needs only one run to get multi class results. This method improves greatly classification efficiency, which totally attributes to encoding scheme in the CSP-based Classifier.

The most current rule-based evolutionary classifiers use tree type structure to represent a classification rule. For a two class problem, only a single tree (T) is enough to classify a pattern thoroughly. For a pattern x,

$$\text{if } T(x) \geqslant 0, x \in \text{class } 1,$$
$$\text{else, } x \in \text{class } 2.$$

This scheme can be extended to a multi category classification problem. In our classifier design, every antibody or individual of CSP consists of c genes which correspond to c class. It means that a gene tree takes charge of classifying a class problem. So, for a c class classification problem, the ith antibody will have c gene trees, and these will be denoted by $GT_k^i, k = 1, 2, \ldots, c$. The ith individual of CSP, namely, a possible solution of classification rules is represented by c gene trees $(GT_1^i, GT_2^i, \ldots, GT_c^i)$.

For a pattern x,

$$\text{if } GT_i(x) \geqslant 0 \text{ and } GT_j(x) < 0 \text{ for all } j \neq i,\ i, j \in \{1, 2, \ldots, c\},$$
$$\text{then } x \in \text{ class } i$$
$$\text{else, } x \notin \text{ any one class}$$

Suppose more than one gene tree of an antibody have positive responses to the training pattern x or all gene trees have negative responses to the training pattern x, then it shows that the rule can not find which class the pattern x belong to, so the rule is not the best one, it needs further training. When we use the rule discovered after training to predict the accuracy of unseen data, if more than one gene tree of the classification rule show positive responses for a pattern, it is necessary to use post-processing methodologies assign a class to the pattern. If no one gene tree of the classification rule has positive response for any pattern, then we can say we do not know the pattern belongs to which class. The following subsections are the steps achieving our goal.

3.3.1. Initialization

Each of the c gene trees of each individual of CSP classifier is initialized randomly using the function set and the terminal set. The function set may consist of many kinds of arithmetic functions, and the terminal set contains all attribute variables of data and constants. The function set F and terminal set T used here are as follows: $F = \{+, -, \times, \div, \sin, \cos\}$ and T = {attribute variables, constant}, where constants are randomly generated in the range of [0, 1.0]. We have initialized gene trees using the method which combines the randomly selected function and terminal together.

3.3.2. Affinity measure and training

We use a set of training samples to train CSP-based classifier. While training, the response of a gene tree GT_i for a pattern x is expected to be as follows:

```
Mutation
    for each gene g(g=1,2,3,…,N) in the antibody
      for each symbol i(i=1,2,3,…,L) in the gene
        if rand(1) < Pm
          if i < head_length
            replace symbol i by a randomly-generated function or
            terminal
          else
            replace symbol i by a randomly-generated terminal
    end
```

Fig. 4. Pseudocode of mutation operation.

```
Input: Ag,MaxGen,N,n, β d,r
Output: the best antibody in memory cell set M
Initialize the population P (N antibodies Ab_N)
for t = 1 to MaxGen
  f = Evaluate(Ab_N,Ag);// evaluate all antibodies in
population P
  Ab_n = Select(f);//select n(n<N) antibodies with higher
affinity
  Ab_C = Clone(Ab_n, β );     //generate clone population PC
  Ab_M =  Mutation(Ab_C);  //generate mutated clone population PM
                           based on clone population PC
  f* = Evaluate(Ab_M,Ag);//re-evaluate antibodies in PM
  Ab_N =  Reselect(Ab_M,f*, Ab_N);//re-select r best antibodies
                           becoming  memory cell set M and
                           replace r worst antibodies in
                           population P with M
  Ab_d =  Generate(d);     //generate d new random antibodies
  Ab_N =  Replace(Ab_d,f, Ab_N)   //replace d low-affinity
                             antibodies in P with Ab_d
end
```

Fig. 5. Pseudocode of clonal selection programming.

if $GT_i(x) \geqslant 0 \quad x \in$ class i
if $GT_i(x) < 0 \quad x \notin$ class i

In other words, a classifier with c genes is said to correctly classify a sample x, if and only if all of its gene trees correctly classify that sample. We emphasize that if a training sample is from class i, then we say that gene tree GT_i correctly classifies x, if $GT_i(x) \geqslant 0$. On the other hand, the gene trees $GT_{j,j\neq i}$ is said to correctly classify X , if $GT_j(x) < 0$.

During the training of gene tree GT_i, the class label of each sample are all known. Suppose in gene tree GT_i, if a pattern x belongs to class i, then the pattern x is defined as positive sample, else it is defined as negative sample. For any gene tree GT_i, we always expect it classify a sample as follows:

$GT_i(x) \geqslant 0$, if x is a positive sample
$GT_i(x) < 0$, if x is a negative sample.

Suppose p and n are the number of positive and negative examples classified by an antibody (rule), respectively, N is the total number of samples in the training set. Then the classification accuracy of a rule to samples in training dataset is defined as follows:

$$Accuracy(GT_i) = \frac{p+n}{N}.$$

The higher training classification accuracy is, the better the rule is. Thus, it can be regarded as a measure of whether the classification rule is good or bad. Here, we define the affinity function of an antibody as

$$Affinity(GT_i) = \frac{p+n}{N}. \tag{8}$$

Since most of the processing time is used on evolutionary search for classification rules of each gene tree, which are independent of one another (within a generation), it is quite obvious that we can focus on searching each gene tree in each parallel program. Thus, better solution can be determined if we search the classification rules of each gene tree in parallel. We use the master/slave-style parallel model implement parallel CSP on a single computer. The whole system is described in Fig. 6. Suppose the dataset has c classes, then all antibody individuals consist of c gene trees, namely, Antibody $= \{GT_1, GT_2, \ldots, GT_c\}$. The whole system will have a master thread and c slave threads. A controlling host computer program, called master thread, performs whole search of classification rules which are encoded by antibody including c gene trees. Moreover, c slave threads respectively run CSP algorithm to find c best classification gene trees, where the affinity function in slave thread is defined by Eq. (8). The ith slave thread does its best discover the best classification gene tree which can classify correctly the ith class pattern and non ith class pattern. In each slave thread, after running a generation, if the best classification gene tree in current generation is better than that of last generation, the best classification gene tree is chosen and sent to master thread replace the corresponding gene tree in antibody, else the gene trees in antibody will keep unchangeable. After master thread gets some new genes, these genes will construct a new antibody, the affinity of the new antibody will be calculated, then compared with that of old one, if the new antibody has higher affinity, it will kept down and gradually become the best classification rule, else it will be put away. The rule of calculating the affinity of antibody in master thread is as follows:

For a pattern x whose class label is i,

if $GT_i(x) \geqslant 0$ and $GT_j(x) < 0$ for all $j \neq i,\ i,j \in \{1,2,\ldots,c\}$,
then the antibody correctly classify the pattern, the score will increase by 1,
else the score will not be increased.

So, suppose the number of correctly classified pattern is n, the affinity value of an antibody in master thread is calculated by

$$Affinity = \frac{n}{N} \tag{9}$$

where N is the total number of training pattern.

The multi thread of the whole system runs in parallel making each slave thread concentrate on finding the best classification gene tree of each class. Compared with single thread method, it can enhance the efficiency of searching and improve the classification accuracy.

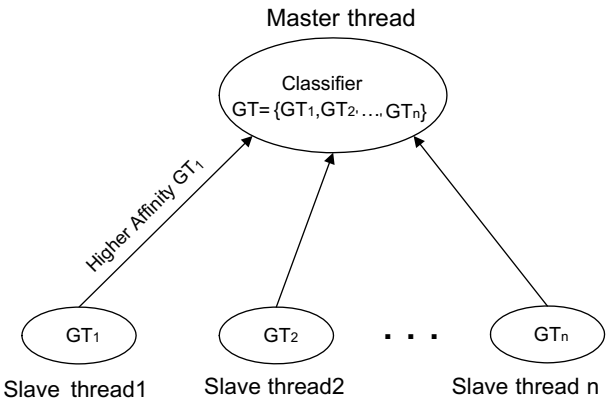


Fig. 6. Multi thread model for CSP-based classifier.

3.3.3. Termination of training of CSP classifier

The training procedures of CSP-based classifier is terminated if all training samples are classified correctly by a classifier or a predefined number of generations are completed. If all of the training samples are correctly classified by the CSP-based classifier, the best individual of the master thread is the required optimal classification rule. Otherwise, if CSP algorithm is terminated after completion of predefined generations, the best J individual of the population in each slave thread is passed through post-processing operation to get the optimal classification rule. The best J individual(s) in each slave thread are selected as follows:

Based on the accuracy of correctly classified training samples, we sort all individuals in population from high to low, the J highest accuracy of classifiers are chosen to pass through post-processing operation for further selecting the optimal classifier.

3.3.4. Post-processing operation to CSP-based classifier

After completion of training of CSP-based classifier, c best gene trees, say, $\{GT_1, GT_2, \ldots, GT_c\}$ are generated respectively from c slave threads. The gene tree GT_1 performs its best in distinguishing the first class data from other class data, GT_2 does its best in distinguishing the second class data from other class data, and so on. GT_c can distinguish the cth class data from other class data. Although these c best gene trees can classify c class data correctly, their combination, antibody which represents a classifier, is possible not obtaining the highest classification accuracy. Suppose a slave thread has J better gene trees, $GT_i^1, GT_i^2, \ldots, GT_i^j, GT_i^1$ is good for a particular segment of the feature space, while GT_i^j models well another segment of the feature space. The overall performance, in terms of misclassification, of GT_i^1 and GT_i^j could be comparable or different. In this case, combining GT_i^1 and GT_i^j into an antibody may result in a much better classifier. This is the principle behind this post-processing operation. If the best individual of the CSP-based classifier run is unable to classify correctly all training samples, then J best gene trees from each thread will be kept down and combined, selecting the best performing (in terms of number of misclassifications) from these combination so that the classification accuracy is further strengthened by this operation. Next, we explain how the combination is obtained.

Suppose having c class data in dataset, there are c slave threads in our system. In the ith slave thread, firstly we randomly choose J individuals as the best J individuals in evolution, namely, J gene trees $GT_i^1, GT_i^2, \ldots, GT_i^j$. After completion of each generation, if the classification accuracy of the best gene tree GT_i in current population is better than the one of the last population, then compare the classification accuracy of gene tree GT_i with the maximal value of J gene trees $GT_i^1, GT_i^2, \ldots, GT_i^j$. If GT_i is greater than the maximal value, then GT_i will replace the minimal value of J gene trees $GT_i^1, GT_i^2, \ldots, GT_i^j$, else value of J gene trees $GT_i^1, GT_i^2, \ldots, GT_i^j$ will keep unchangeable. The procedure continues until the completion of training, the best J individuals, $GT_{ibest}^1, GT_{ibest}^2, \ldots, GT_{ibest}^j$ in training procedure of ith thread will be chosen. Using this method, we can obtain the best J *gene trees* $GT_i^1, GT_i^2, \ldots, GT_i^j$ *of C threads. Here,* $i = 1, 2, \ldots, c$, we select a gene tree randomly from J best individuals of each thread to construct an antibody. Thus, we can obtain total J^c combinations. Using the N training samples, we can calculate the affinity of J^c antibodies. The antibody having the highest affinity is chosen as final trained classifier.

3.3.5. Conflict resolution with weighting scheme

After we have obtained the final trained classifier, Antibody $= \{GT_1, GT_2, \ldots, GT_c\}$, we now focus on the test phase. For a test pattern x, whose class label is i, if $GT_i(x) \geqslant 0$ and $GT_j(x) < 0$ for all $j \neq i,\ i,j \in \{1, 2, \ldots, c\}$, then we are easy to judge if x belong to class i or not. When more than one gene trees show positive responses for the pattern x, we must use additional method to estimate the class label of pattern x. Here, we use weighting scheme to solve the problem. For the ith gene tree GT_i of the classifier, we define its classification weight

$$w_i = \frac{p+n}{N}, \tag{10}$$

where N is the total number of training samples, p is the total number of cases such that the instances is from class i, and the ith gene tree of classifier shows a positive response, n is the total number of cases such that the instances is not from class i, and the ith gene tree of classifier shows a negative response. In fact, the weight of ith gene tree is its classification accuracy to training patterns. This provides the relative importance of the gene tree in making a correct prediction. After completion of training, we can calculate weights of c gene trees. For a test pattern x, if a classifier finds a conflict between classes i and j, which means that GT_i and GT_j all give positive responses, then x is assigned to class i, if $w_i > w_j$; otherwise, x is assigned to class j.

4. Fault detection and results

The above CSP-based classifier is used in this study to verify its effectiveness. Two types of faults are considered. The first are electrical faults, and second are mechanical faults. In the following obtained results, they show that both types of faults can be detected and distinguished. Also, it is worth noting that the classifying rules for machine fault can be understood easier.

We often select different function sets and terminal sets for CSP to solve different classification problems. But general speaking, the chosen function set is the mathematical function, terminal set is the attributes of patterns and some constants. In all the following experiments, we used the function set $F = \{+, -, \times, \div, \sin, \cos\}$. All input attributes of each problem and constants randomly generated in the range of [0, 1.0] were included in the terminal set. According to different datasets, we chose different the CSP parameters, e.g., antibody size, population size, probability of mutation. The listed parameters in Tables 1 and 2 are also the best setup for establishing the best classification rules. In our simulations, we used 100 for population size and 0.2 for mutation rate on all the databases. Other parameters, e.g., the antibody size, the maximum number of iterations to run, or the length of the variable region V depend on the characteristics of classification problems such as feature distribution. For each class learning, the CSP algorithm stops when the number of generation reaches or the affinity measure reaches 1.0, which means it correctly classifies all positive and negative examples. The number of chosen individuals for post-processing in each slave thread for electrical and mechanical fault

were set to 5, 10, respectively. The common parameters used for all dataset in CSP are listed in Table 3. In order to avoid bias, the presented results are the statistics of 10 different trials.

4.1. Electrical faults

Unbalance fault of a three-phase machine system is used in this section to demonstrate the effectiveness of the proposed CSP-based classifier. In order to simulate the electrical unbalance faults in a three-phase induction motor, a noninvasive test rig as shown in Fig. 7 is used in this study. As faults associated with the unbalanced phase alter the electromagnetic flux, the electromagnetic force acting upon the stator core changes. Therefore, the detection of machine electrical unbalance faults in this paper is based on the stator core vibration signals. The data acquisition system is shown in Fig. 8. The rated power and speed of the induction machine are 1100 W and 1440 r/min. The stator winding are connected in Y. The cases of running at different rotating speeds are studied. This is obtained by feeding the machine with different supply frequency ranged from 35 to 50 Hz. In order to simulate unbalanced electrical faults, the "C" phase is serially connected through a variable resistor ranging from 0 to 40 Ω to introduce unbalanced electrical faults. The machine is driven under different asymmetrical faults by different values of the resistor from 10 to 40 Ω under different supply frequencies from 35 to 50 Hz. The vibration signal is measured by an accelerometer mounted on the induction machine as shown in Figs. 8 and 9. The sensitivity and upper frequency limit of the accelerometer is 1 pC/ms^2 and 12 kHz, respectively. A charge amplifier (Bruel & Kjar, type 2635) with the gain of 316 mV/unit amplifies the signal picked up by the accelerometer. The acquired data are fed into a low-pass filter with bandwidth of 3 kHz. The amplified and filtered signal is sampled and inputted into a Pentium PC. The sampling rate is 10 kHz, and 1024 samples for each data frame, which are transformed into frequency-domain signals and used for feature vectors as the inputs of the CSP-based classifier.

4.2. Mechanical faults

Mechanical faults of machine rotors are the most common in industry since rotors are essential components of rotating machine systems. For simplicity, in this study only rotor fault situation was used to represent a general type of mechanical fault. The experimental test rig shown in Fig. 10a shows two identical 10-hp three-phase induction machines operating at 380 V/50 Hz which were used. To obtain different rotating speeds, a three phase converter was used to vary the supply frequency from 35 to 50 Hz. One normal machine state was used as a reference, while another state with a synthetic rotor fault was used for diagnostic analysis. The synthetic rotor fault was obtained by drilling a small hole of 3-mm diameter and 4-mm depth on the rotor bar, as shown in Fig. 10b. In order to distinguish the different conditions of the mechanical unbalanced faults, two operating states were used in this investigation. The vibration signals were measured from an accelerometer mounted on the cage of the monitored machine. The sensitivity and upper frequency limit of the accelerometer are 1 pC/ms^2 and 12 kHz, respectively. The vibration data was amplified by a charge amplifier (Bruel & Kjar, type 2635) with the gain of 316 mV/unit. The amplified signals were sampled and interfaced to a 586 PC by an AT-MIO-16 D data A/D card and DAQ-Ware 4.5 (National Instruments Corporation) software.

4.3. Results using the CSP-based classifier

First, the power supply frequency was varied in the range from 35 to 50 Hz. There are four frequencies investigated in this paper: 50, 45, 40, and 35 Hz. The number of training and testing data at different conditions and frequencies are listed in Table 4. In electrical fault test, for training the CSP-based classifier, 500×1024 time-series data were collected from the accelerometer: 100×1024 data for normal condition, 400×1024 data for different extents of electrical faults. After computation by 1024-point FFT and feature extraction, we obtained 100 feature vectors for normal condition, 400 for electrical condition. The testing data under certain supply frequency include 100 feature vectors for normal condition, 400 for four different extents of electrical fault. In mechanical fault test, using the same preprocessing method, we obtained 100 feature vectors of normal condition and 100 of mechanical fault condition for training CSP-based classifier. The testing data under certain supply frequency include 100 feature vectors for normal condition, 100 for mechanical fault. As the four features are in different scales, the four extracted features are normalized by zero-mean normalization: subtracting the mean and dividing by the standard deviation of each feature in turn.

We define the accuracy as the number of correctly diagnostic data over that of total data. The results of the machine operating at different frequencies and conditions are listed in Tables 5 and 6, respectively. From Table 5, it is shown that the proposed method is able to identify the types of faults investigated in this study. The maximal and minimal accuracy of detecting electrical faults is 100% and 85.64% for the training data, respectively, and 99.98% and 84.36% for the testing data. The maximal and minimal accu-

Table 1
Different parameters used for dataset composed of normal and electrical fault at all frequency in CSP.

Frequency (Hz)	No. of population	Mutation rate	Length of the variable region V	No. of generation	No. of selected individual J
35	100	0.2	10	500	5
40	100	0.2	10	300	5
45	100	0.2	10	200	5
50	100	0.2	10	500	5

Table 2
Different parameters used for dataset composed of normal and mechanical fault at all frequency in CSP.

Frequency (Hz)	No. of population	Mutation rate	Length of the variable region V	No. of generation	No of selected individual J
35	100	0.2	8	100	10
40	100	0.2	8	300	10
45	100	0.2	8	300	10
50	100	0.2	8	300	10

Table 3
Common parameters for all frequency in CSP.

Parameter	Value
Population size N	100
Number of antibodies selected n	20
Clone factor β	2
Number of antibodies replaced r	80
Number of new antibodies d	40

racy of detecting the normal condition is 100% and 98.14% for the training data, and 99.98% and 85.1%for the testing data. From Table 6, we can find that the maximal and minimal accuracy of detecting mechanical faults is 98.8% and 89.3% for the training data, respectively, and 96.9% and 87.05% for the testing data. The maximal and minimal accuracy of detecting the normal condition is 98.55% and 88.25%, respectively for the training data, and 96.85% and 86.2% for the testing data. The detail performance of our proposed method is listed in Tables 5 and 6. From these results, we also can find that our proposed method only needs a few training data can explore better fault detection rulers.

After Normal and Electrical fault signals are processed, these Datasets are composed of four features which are average frequency, normalized second-order central moment, normalized third-order central moment and total power respectively. Here, we use $x1$, $x2$, $x3$, $x4$ represent them, respectively. The expressions of a classifier (five gene trees) for electrical fault detection are shown as follows:

- Gene tree 1: (/ cos − $x2$ $x2$ − / $x1$ 4.788971 ∗ $x2$ $x1$)
- Gene tree 2: (sin + 3.927002 − − $x3$ $x2$ sin $x3$)
- Gene tree 3: (/ sin $x1$ − / $x1$ $x2$ ∗ 1.483307 $x2$)
- Gene tree 4: (− / $x3$ − $x2$ $x1$ + / 0.440369 $x1$ 2.237396)
- Gene tree 5: (− sin + $x1$ $x3$ + cos $x1$ $x1$)

$x1$, $x2$, $x3$, and $x4$ represent four features of vibration signal in frequency domain. After transforming the above expressions into if-then rule, we can get the following classification rules:

For a pattern $x = (x1, x2, x3, x4)$,

- Rule 1: IF $\left(\frac{\cos(x2-x2)}{\frac{x1}{4.788971}-x1*x2}\right) > 0$, THEN Class = Normal;
- Rule 2: IF $\sin(3.927002 + x3 - x2 - \sin(x3)) > 0$, THEN Class = 10 Ω Electrical Fault;
- Rule 3: IF $\left(\frac{\sin(x1)}{\frac{x1}{x2}-1.483307*x2}\right) > 0$, THEN Class = 20 ΩElectrical Fault;
- Rule 4: IF $\left(\frac{x3}{x2-x1} - \left(\frac{0.440369}{x1} + 2.237396\right)\right) > 0$, THEN Class = 30 ΩElectrical Fault;
- Rule 5: IF $(\sin(x1 + x3) - (\cos(x1) + x1)) > 0$, THEN Class = 40 ΩElectrical Fault

In additional, normal and mechanical fault signals are also processed by the same method, these Datasets are also composed of four features which are average frequency, normalized second-order central moment, normalized third-order central moment and total power respectively. Here, we still use $x1$, $x2$, $x3$, $x4$ represent them, respectively. These signals are classified normal and

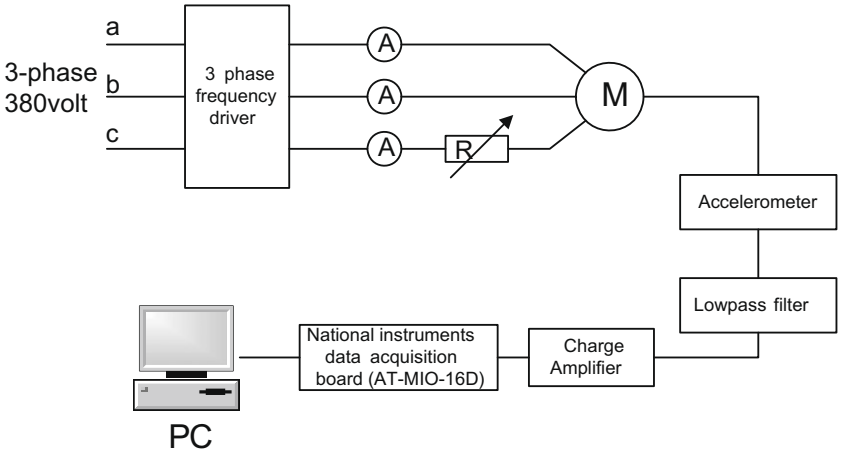


Fig. 7. Testing rig of three induction motor.

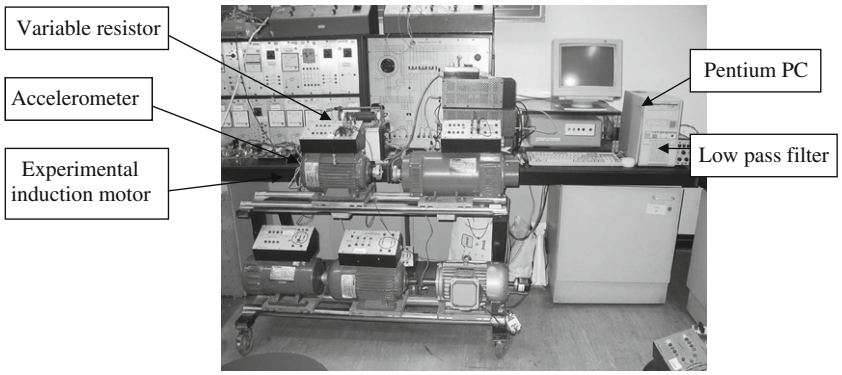


Fig. 8. Real-time vibration signal acquisition system.

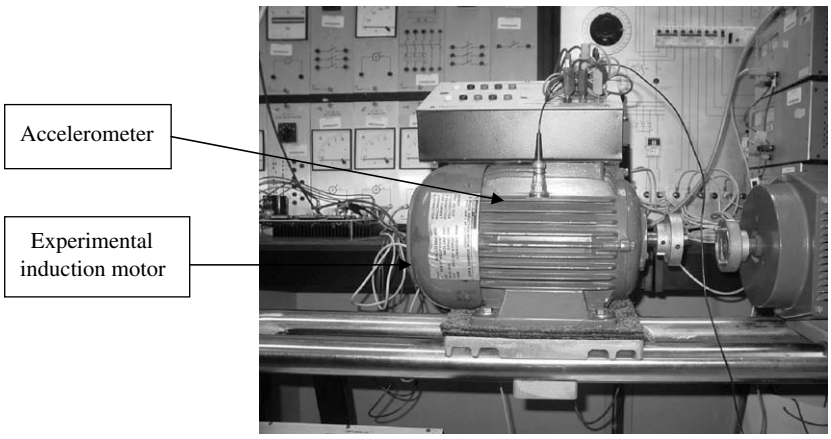


Fig. 9. Accelerometer mounted on the induction motor.

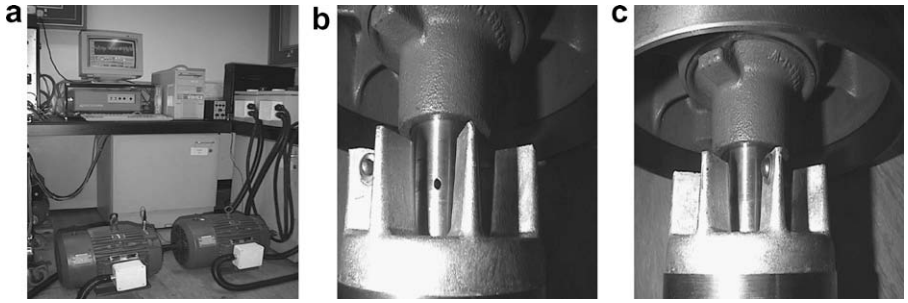


Fig. 10. Experimental rig for detecting *mechanical faults* (a) by vibration analysis of induction motors under different speeds; (b) a rotor with a 3 mm hole and (c) a normal rotor.

Table 4
The number of training and testing data at different conditions and frequencies.

			50 Hz/45 Hz/40 Hz/35 Hz
Number of training data	Normal		100
	Mechanical fault		100
	Electrical fault	10 Ω	100
		20 Ω	100
		30 Ω	100
		40 Ω	100
Number of testing data	Normal		100
	Mechanical fault		100
	Electrical fault	10 Ω	100
		20 Ω	100
		30 Ω	100
		40 Ω	100

Table 5
Accuracy of training and testing data for electrical fault at different frequencies.

		35 Hz	40 Hz	45 Hz	50 Hz
Accuracy of training data	Normal	100% ± 0	100% ± 0	100% ± 0	98.14% ± 0.1
	10 Ω	99.66% ± 0.35	99.36% ± 1.28	100% ± 0	85.64% ± 3.96
	20 Ω	99.98% ± 0.06	99.92% ± 0.17	100% ± 0	93.5% ± 1.46
	30 Ω	89.66% ± 2.21	100% ± 0	100% ± 0	98.4% ± 0.49
	40 Ω	88.42% ± 1.71	88.56% ± 5.79	100% ± 0	99.92% ± 0.1
Accuracy of testing data	Normal	94.04% ± 6.27	99.98% ± 0.06	85.1% ± 4.18	92.46% ± 4.41
	10 Ω	85.22% ± 5.74	99.3% ± 0.84	96.9% ± 4.58	84.36% ± 3.64
	20 Ω	93.96% ± 4.58	99.22% ± 1.48	86.32% ± 1.73	92.18% ± 1.66
	30 Ω	85.56% ± 8.42	99.88% ± 0.38	97.24% ± 5.85	95.46% ± 0.97
	40 Ω	84.54% ± 3.5	84.62% ± 5.51	99.9% ± 0.25	99.3% ± 0.34

Table 6
Accuracy of training and testing data for mechanical fault at different frequencies.

		35 Hz	40 Hz	45 Hz	50 Hz
Accuracy of raining data	Normal	96.15% ± 2.8	88.25% ± 1.29	98.55% ± 3.91	98% ± 1.68
	Mechanical fault	97.6% ± 1.15	89.3% ± 2.45	98.8% ± 3.62	95.6% ± 0.94
Accuracy of testing data	Normal	90.3% ± 8.67	86.2% ± 2.8	96.85% ± 3.95	94.1% ± 4.29
	Mechanical fault	95.3% ± 4.48	87.05% ± 1.8	96.9% ± 3.94	92% ± 2.69

mechanical fault. The expressions of a classifier (two gene trees) for mechanical fault detection are shown as follows:

- Gene tree 1: (+ / cos $x1$ $x3$ + * $x1$ 4.659882 $x3$)
- Gene tree 2: (+ / sin $x2$ * $x2$ $x1$ - $x2$ $x1$)

After transforming the above expressions into if-then rule, we obtain the following classification rules:

For a pattern $x = (x1, x2, x3, x4)$,

- Rule 1: IF $\left(\frac{\cos(x1)}{x3} + 4.659882 * x1 + x3\right) > 0$, THEN Class = Normal;
- Rule 2: IF $\left(\frac{\sin(x2)}{x1 * x2} + x2 - x1\right) > 0$, THEN Class = Mechanical Fault;

The above Rules show one typical example set of identified CSP rules for electrical and mechanical fault detection of induction motors. It is worth noting that the CSP generated rules are quite accurate, compact, and easier to understand.

5. Conclusion

A newly developed CSP-based classifier is employed for detecting induction machine faults operating at different rotating speeds. Intensive tests were performed at different operating conditions. This paper shows that when appropriate feature vectors are extracted as the inputs of the proposed CSP-based classifier, the proposed method is able to deliver very high detection accuracy. In the proposed approach, not only different types of machine fault can be detected, but also the obtained rules are easily understood. This is very useful for us to implement fault detection system and to analyze the reasons machine faults generated. In additional, our proposed method is very simple to implement, and it need not too much trials to determine the best parameters for algorithm, this will be proved to be extremely useful for practical industrial applications.

References

Alguindigue, I. E., Buczak, A. L., & Uhrig, R. E. (1993). Monitoring and diagnosis of rolling element bearings using artificial neural networks. *IEEE Transactions in Industrial Electronics, 40*, 209–217.

Bojarczuk, C. C., Lopes, H. S., & Freitas, A. A. (2000). Genetic Programming for knowledge discovery in chest pain diagnosis. *IEEE Engineering in Medicine and Biology Magazine, 19*(4), 38–44.

Brodley, C. E., & Danyluk, A. P. (2001). Round robin rule learning. In *Proceedings of 18th International Conference on Machine Learning (ICML-01)* (pp. 146–153).

Cheng, C., & Chiu, M. S. (2005). Nonlinear process monitoring using jitl-pca. *Chemometrics and Intelligent Laboratory Systems, 76*, 1–13.

Chien, B.-C., Lin, J. Y., & Hong, T.-P. (2002). Learning discriminant functions with fuzzy attributes for classification using genetic programming. *Expert Systems with Applications, 23*, 31–37.

Chow, M.-Y. (1997). *Methodologies of using neural network and fuzzy logic for motor incipient fault detection*. Singapore: World Scientific.

Chow, T. W. S. (2000). HOS-based nonparametric and parametric methodologies for machine fault detection. *IEEE Transactions on Industrial Electronics, 47*, 1051–1059.

Chow, T. W. S., & Fei, G. (1995). Three phase induction machines asymmetrical faults identification using bispetrum. *IEEE Transactions on Energy Conversion, 10*, 688–693.

Chow, M.-Y., & Mangum, C. (1991). A neural network approach to real time condition monitoring of induction machines. *IEEE Transactions on Industrial Electronics, 38*, 449–454.

Chow, M.-Y., Sharpe, R. N., & Hung, J. C. (1993). On the application and design consideration of artificial neural network fault detectors. *IEEE Transactions in Industrial Electronics, 40*, 181–198.

Collacott, R. A. (1979). *Vibration monitoring and diagnosis*. New York: Halsted.

Cutello, V., Nicosia, G., Pavone, M., & Timmis, J. (2007). An immune algorithm for protein structure prediction on lattice models. *IEEE Transactions on Evolutionary Computation, 11*(1), 101–117.

De Castro, L. N., & Von Zuben, F. J. (2000). The Clonal Selection Algorithm with engineering applications. In *Proceedings of GECCO'00, workshop on artificial immune systems and their applications* (pp. 36–37).

De Castro, L. N., & Timmis, J. (2002). *Artificial immune systems: A new computational intelligence approach*. Springer.

De Falco, I., Della Cioppa, A., & Tarantino, E. (2002). Discovering interesting classification rules with genetic programming. *Applied Soft Computing, 23*, 1–13.

Dimarogonas, A. (1996). *Vibration for engineers*. Englewood Cliffs, NJ: Prentice-Hall.

Donley, M., Stokes, W., Jeong, G. S., Suh, K. K., & Jung, S. G. (1996). Validation of finite element models for noise/vibration/harshness simulation. *Sound and Vibration, 30*(8), 18–23.

Dounias, G., Tsakonas, A., Jantzen, J., Axer, H., Bjerregaard, B., & Keyserlingk, D. (2002). Genetic programming for the generation of crisp and fuzzy rule bases in classification and diagnosis of medical data. In *Proceedings of 1st International NAISO Congress Neuro Fuzzy Technologies*. Canada: Academic Press, [CD-ROM].

Er, M. J., Wu, S., Lu, J., & Toh, H. L. (2002). Face recognition with radial basis function (RBF) neural networks. *IEEE Transactions on Neural Networks, 13*, 697–710.

Ferreira, C. (2006). *Gene expression programming: Mathematical modeling by an artificial intelligence*. Springer.

Filippetti, F., Franceschini, G., & Tassoni, C. (1995). Neural network aided on-line diagnostics of induction motor rotor faults. *IEEE Transactions on Industry Applications, 31*, 892–899.

Filippetti, F., Franceschini, G., Tassoni, C., & Vas, P. (2000). Recent developments of induction motor drives fault diagnosis using AI techniques. *IEEE Transactions on Industrial Electronics, 47*, 994–1004.

Gan, Z. et al. (2008). Clone selection programming and its application to symbolic regression. *Expert System and Application, 36*(2P2), 3996–4005.

Helmer, G., Wong, J. S. K., Honavar, V., & Miller, L. (2002). Automated discovery of concise predictive rules for intrusion detection. *Journal of Systems and Software, 60*, 165–175.

Isermann, R. (1997). Supervision, fault-detection and fault-diagnosis methods – An introduction. *Control Engineering Practice, 5*(5), 639–652.

Kishore, J. K., Patnaik, L. M., Mani, V., & Agrawal, V. K. (2000). Application of genetic programming for multicategory pattern classification. *IEEE Transactions on Evolutionary Computation, 4*, 242–258.

Kojima, F., Kubota, N., & Hashimoto, S. (2001). Identification of crack profiles using genetic programming and fuzzy inference. *Journal of Materials Processing Technology, 108*, 263–267.

Koza, J. R. (1992). *Genetic Programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Leohardt, S., & Ayoubi, M. (1997). Methods of fault diagnosis. *Control Engineering Practice, 5*(5), 683–692.

Li, B., Chow, M. Y., & Tipsuwan, T. (2000). Neural-network-based motor rolling bearing fault diagnosis. *IEEE Transactions on Industrial Electronics, 47*, 1060–1069.

Lim, T.-S., Loh, W.-Y., & Shih, Y.-S. (2000). A comparison of prediction accuracy, complexity and training time of thirty-three old and new classification algorithms. *Machine Learning Journal, 40*, 203–228.

Lipovszky, G., Solyomvari, K., & Varga, G. (1990). *Vibration testing of machines and their maintenance*. Amsterdam, The Netherlands: Elsevier.

Loveard, T., & Ciesielski, V. (1983). Representing classification problems in genetic programming. In *Proceedings of congress evolutionary computation, May 27–30, 2001* (pp. 1070–1077).

Mendes, R. R. F., Voznika, F. B., Freitas, A. A., & Nievola, J. C. (2001). Discovering fuzzy classification rules with genetic programming and co-evolution. In *Proceedings of 5th European conference PKDD, lecture notes in artificial intelligence* (Vol. 2168, pp. 314–325).

Muni, D. P. et al. (2004). A novel approach to design classifiers using genetic programming. *Evolutionary Computation, 8*(2), 183–196.

Nossal, G. J. V. (1993). The molecular and cellular basis of affinity maturation in the antibody response. *Cell, 68*, 1–2.

Patton, R., Frank, P., & Clark, R. (1989). *Fault diagnosis in dynamic systems, theory and application*. Englewood Cliffs, NJ: Prentice-Hall.

Penman, J., & Yin, C. M. (1994). Feasibility of using unsupervised learning NN for the condition monitoring of electrical machines. *Proceedings of IEE Electric Power Applications, 141*(6), 317–322.

Ragulskis, K., & Yurkauskas, A. (1989). *Vibration of bearings*. Bristol, PA: Hemisphere.

Sahan, S., Kodaz, H., Güneş, S., & Polat, K. (2004). A new classifier based on attribute weighted artificial immune system. *Lecture Notes in Computer Science (LNSC 3280)*, 11–20.

Schlang, J. H., Habetler, T. G., & Lin, B. K. (1997). An unsupervised neural network fault discriminating system implementation for on line condition monitoring of induction machine using stator current. In *Proceedings of SDEMPED'97, Carry le Rouet, France* (pp. 285–289).

Stanhope, S. A., & Daida, J. M. (1998). Genetic programming for automatic target classification and recognition in synthetic aperture radar imagery. In *Proceedings of 7th annual conference on evolutionary programming, evolutionary programming VII* (pp. 735–744).

Thomas, D. W. (1971). *Vehicle sound recognition: A pilot study*. Ph.D. dissertation. Dept. Electron., Univ. Southampton, Southampton, UK.

Tonegawa, S. (1983). Somatic generation of antibody diversity. *Nature, 302*, 575–581.

Tse, P. W., Peng, Y. H., & Yam, R. (2001). Wavelet analysis and envelope detection for rolling element bearing fault diagnosis-their effectiveness and flexibilities. *Journal of Vibration and Acoustics, 123*, 303–310.

Wang, W. J. (2001). Wavelets for detecting mechanical faults with high sensitivity. *Mechanical Systems and Signal Processing, 15*(4), 685–696.

Watkins, A., Timmis, J., & Boggess, L. (2004). Artificial immune recognition system (AIRS): An immune-inspired supervised learning algorithm. *Genetic Programming and Evolvable Machines, 5*(3), 291–317.

Wierzchon, S. T. (2002). Function optimization by the immune metaphor. *Task Quarterly, 6*(3), 493–508.

Wu, Sitao., & Chow, T. W. S. (2004). Induction machine fault detection using SOM-based RBF neural networks. *IEEE Transactions on Industrial Electronics, 51*, 183–194.

Ye, Z., & Wu, B. (2000). A review on induction motor online fault diagnosis. In *Proceedings of third International power electronics and motion control conference* (pp. 1353–1358).

Zhang, L., Jack, L. B., & Nandi, A. K. (2005). Fault detection using genetic programming. *Mechanical Systems and Signal Processing, 19*, 271–289.

Zhang, B.-T., & Muhlenbein, H. (1995). Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation, 3*(1), 17–38.

Zhang, Liang, & Nandi, A. K. (2006). Fault classification using genetic programming. *Mechanical Systems and Signal Processing, 21*, 1273–1284.