ISA Transactions 51 (2012) 111–119

Contents lists available at SciVerse ScienceDirect

ISA Transactions

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

Control chart pattern recognition using K-MICA clustering and neural networks

Ataollah Ebrahimzadeh *, Jalil Addeh, Zahra Rahmani

Faculty of Electrical and Computer Engineering, Babol University of Technology, Babol, Iran

ARTICLE INFO

Article history:
Received 26 May 2011
Received in revised form
18 August 2011
Accepted 23 August 2011
Available online 28 October 2011

Keywords:
Control chart patterns
Clustering
Neural networks
Modified imperialist competitive algorithm
K-means algorithm

ABSTRACT

Automatic recognition of abnormal patterns in control charts has seen increasing demands nowadays in manufacturing processes. This paper presents a novel hybrid intelligent method (HIM) for recognition of the common types of control chart pattern (CCP). The proposed method includes two main modules: a clustering module and a classifier module. In the clustering module, the input data is first clustered by a new technique. This technique is a suitable combination of the modified imperialist competitive algorithm (MICA) and the K-means algorithm. Then the Euclidean distance of each pattern is computed from the determined clusters. The classifier module determines the membership of the patterns using the computed distance. In this module, several neural networks, such as the multilayer perceptron, probabilistic neural networks, and the radial basis function neural networks, are investigated. Using the experimental study, we choose the best classifier in order to recognize the CCPs. Simulation results show that a high recognition accuracy, about 99.65%, is achieved.

1. Introduction

Control chart patterns (CCPs) are important statistical process control tools for determining whether a process is run in its intended mode or in the presence of unnatural patterns. CCPs can exhibit six types of pattern: normal (NR), cyclic (CC), increasing trend (IT), decreasing trend (DT), upward shift (US), and downward shift (DS) [1]. Except for normal patterns, all other patterns indicate that the process being monitored is not functioning correctly and requires adjustment. Fig. 1 shows six pattern types of control chart.

Over the years, numerous supplementary rules known as zone tests or run tests [2] have been proposed to analyze control charts. Interpretation of the process data still remains difficult because it involves pattern recognition tasks. It often relies on the skill and experience of the quality control personnel to identify the existence of an unnatural pattern in the process. An efficient automated control chart pattern (CCP) recognition system can compensate this gap and ensure consistent and unbiased interpretation of CCPs, leading to a smaller number of false alarms and better implementation of control charts. With this aim, several approaches have been proposed for CCP recognition. Some of the researchers have used expert systems [2], the fuzzy-clustering method [3] and decision tree (DT) based classifiers [4]. Other researchers have used artificial neural networks (ANNs) for recognition of CCPs [5–15]. ANNs can be simply classified into two main categories: supervised ANNs and unsupervised ANNs. A literature review shows that the techniques that use supervised neural networks as the classifier have higher performances. The advantage with a neural network is that it does not require the provision of explicit rules or templates. Most existing techniques use unprocessed data as the inputs of the CCP recognition system. The use of unprocessed CCP data has many additional problems, such as the amount of data to be processed being large. On the other hand, the approaches which use features are more flexible to deal with a complex process problem, especially when no prior information is available. If the features represent the characteristic of patterns explicitly, and if their components are reproducible with the process conditions, the classifier recognition accuracy will increase [15]. Further, if the feature is amenable to reasoning, it will help in understanding how a particular decision was made, and this makes the recognition process a transparent process. Features could be obtained in various forms, including principal component analysis shape features [11,13], correlation between the input and various reference vectors [16], and statistical correlation coefficients [17].

This paper presents a novel hybrid intelligent method (HIM) for recognition of the common types of control chart pattern. The proposed method includes two main modules: a clustering module and a classifier module. In the clustering module, the input data is first clustered by a new technique. This technique is a suitable combination of the modified imperialist competitive algorithm (MICA) and the *K*-means algorithm. Then the Euclidean distance of each pattern is computed from the determined clusters. It is used as the characteristic feature. The classifier module determines the membership of the patterns using the computed distance.

The rest of paper is organized as follows. Section 2 explains the general scheme of the proposed method. Sections 3 and 4

* Corresponding author.
E-mail address: ataebrahim@yahoo.com (A. Ebrahimzadeh).

0019-0578/$ – see front matter © 2011 ISA. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.isatra.2011.08.005

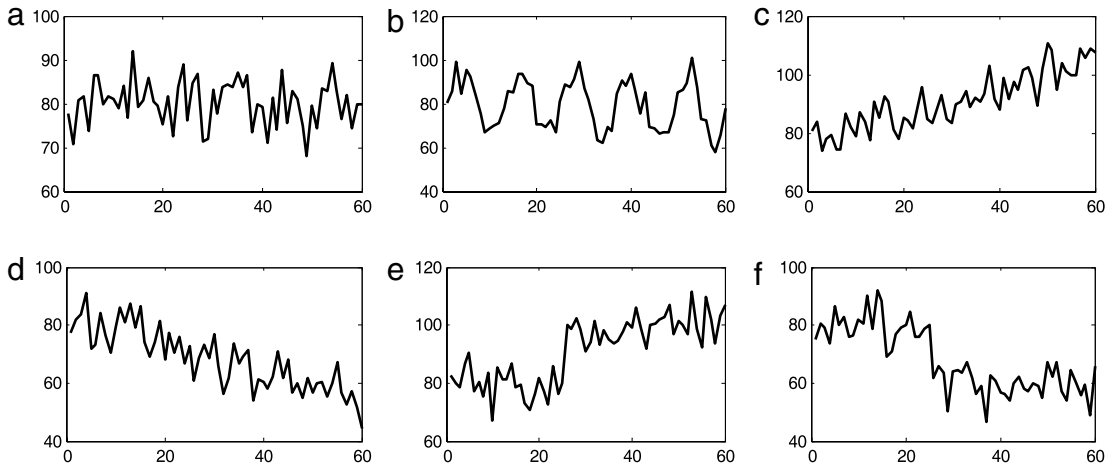

Fig. 1. Six various basic patterns of control charts: (a) normal pattern, (b) cyclic pattern, (c) upward trend, (d) downward trend, (e) upward shift, and (f) downward shift.

Fig. 2. General scheme of the proposed method.

describe the main parts of the proposed method, i.e., the clustering technique and classifier module, respectively. Section 5 shows some simulation results, and Section 6 concludes the paper.

2. General structure of the proposed method

The proposed method includes two main modules: a clustering module and a classifier module. Fig. 2 shows the general scheme of this method. In the clustering module, the input data is first clustered by a new technique. This technique is a suitable combination of the modified imperialist competitive algorithm (MICA) and the *K*-means algorithm. It is named the K-MICA clustering technique. Then the Euclidean distance of each pattern is computed from the determined clusters. It is used as the characteristic feature. It is obvious that the Euclidean distance of each signal from its own cluster center is smaller than the value of Euclidean distance of that signal from other cluster centers. Subsequently, each signal is shown as a 1×6 vector, in which one of the rows has a small value and the remaining rows have a large value. Application of this approach causes the dimension of the classifier to be decreased. Also, we have a more efficient feature that is better than the raw data. The classifier module determines the membership of the patterns using the computed distance. The following sections present the main modules.

3. K-MICA clustering technique

Clustering is an important problem that must often be solved as a part of more complicated tasks in pattern recognition, image analysis, and other fields of science and engineering. Clustering procedures partition a set of objects into clusters such that objects in the same cluster are more similar to each other than objects in different clusters, according to some predefined criteria. In this section, the applied clustering algorithm is described.

3.1. K-means

K-means is one of the simplest unsupervised learning algorithms. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume *K* clusters) fixed a priori. The main idea is to determine the *K* centroids. These centroids should be placed in a cunning way, as different locations cause different results. So, the best choice is to place them as far away from each other as possible. The next step is to take each point referring to a given data set and associate it to the nearest centroid. When no point remains, the first step is completed, and an early grouping is done. At this point, we need to recalculate the *K* new centroids as centers of the clusters resulting from the previous step. After we have these *K* new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop, we may notice that the *K* centroids change their location step by step until no more changes are made. In other words, the centroids do not move any further. Finally, the goal of this algorithm is to minimize an objective function, which in this case is a squared error function [18,19]. The objective function has been calculated as follows:

$$\cos t(X) = \sum_{i=1}^{N} \min\{\|Y_i - X_i\|\}, \quad j = 1, 2, 3, \ldots, k \tag{1}$$

where $\|Y_i - X_i\|$ is a chosen distance measurement between a data input Y_i and the cluster center X_i. *N* and *K* are the number of input data and the number of cluster centers, respectively.

The algorithm is composed of the following steps.

1. Place the *K* points into the space represented by the objects that are clustered. These points represent initial group centroids.
2. Assign each object to the group that has the closest centroid.
3. When all objects have been assigned, recalculate the positions of the *K* centroids.
4. Repeat Steps 2 and 3 until the centroids are immobilized.

Although it can be proved that the procedure will always terminate, the *k*-means algorithm does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centres. The *k*-means algorithm can be run multiple times to reduce this effect.

3.2. The imperialist competitive algorithm (ICA)

The imperialist competitive algorithm (ICA) is a population-based stochastic search algorithm. It was introduced by Atashpaz and Lucas [20,21]. Since then, it has been used to solve some

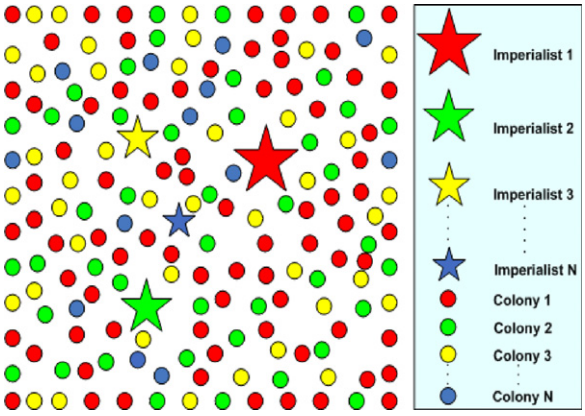

Fig. 3. Generating the initial empire.

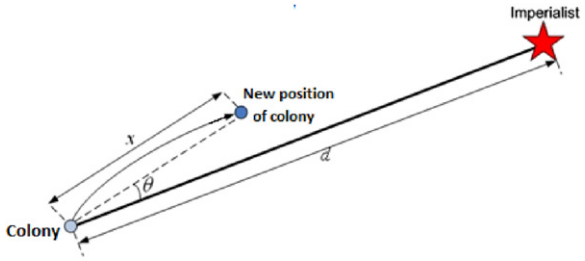

Fig. 4. Moving colonies toward their relevant imperialist.

kinds of optimization problem. The algorithm is inspired by imperialistic competition. It attempts to present the social policy of imperialisms to control more countries and use their sources when colonies are dominated by some rules. If one empire loses its power, the others will compete to take possession of it. In the ICA, this process is simulated by individuals that are known as countries.

This algorithm starts with a randomly initial population and objective function which is computed for them. The most powerful countries are selected as imperialists and the others are colonies of these imperialists. Then a competition between imperialists takes place to get more colonies. The best imperialist has more chance to possess more colonies. Then one imperialist with its colonies makes an empire. Fig. 3 shows the initial populations of each empire [20–22]. If the empire is bigger, its colonies are greater and the weaker ones are less. In this figure, Imperialist 1 is the most powerful and has the greatest number of colonies.

After dividing the colonies between the imperialists, these colonies approach their related imperialist countries. Fig. 4 represents this movement. Based on this concept, each colony moves toward the imperialist by α units and reaches its new position.

$$\alpha \approx U(0, \beta \times S). \tag{2}$$

Here, α is a random variable with uniform (or any proper) distribution; β, a number greater than 1, causes colonies move toward their imperialists from different direction, and S is the distance between the colony and imperialist.

If after this movement one of the colonies possess more power than its relevant imperialist, they will exchange their positions. To begin the competition between empires, the total objective function of each empire should be calculated. It depends on the objective function of both the imperialist and its colonies. Then the competition starts: the weakest empire loses its possessions and powerful empires try to gain them. An empire that has lost all its colonies will collapse. Finally, the most powerful empire will take possession of other empires, and it wins the competition.

To apply the ICA for clustering, the following steps have to be taken [22].

Step 1: The initial population for each empire should be generated randomly.
Step 2: Move the colonies toward their relevant imperialist.
Step 3: Exchange the position of a colony and the imperialist if its cost is lower.
Step 4: Compute the objective function of all empires.
Step 5: Pick the weakest colony and give it to one of the best empires.
Step 6: Eliminate the powerless empires.
Step 7: If there is just one empire, stop; if not, go to 2.

The last imperialist is the solution of the problem.

3.3. Modified ICA (MICA)

In order to improve the convergence velocity and accuracy of the ICA, [23] recommends a modified imperialist competitive algorithm (MICA). Premature convergence may occur under different situations: the population converges to local optima of the objective function or the search algorithm proceeds slowly or does not proceed at all. In [23], a new mutation operator is proposed. Mutation is a powerful strategy which diversifies the ICA population and improves the ICA's performance by preventing premature convergence to local minima. During the assimilation policy, each colony (X) moves toward its relevant imperialist by a unit, where the initial distance between them is S. The new position of each colony would be $X_{\text{move},j}^{t+1}$ (t is the iteration number):

$$\alpha \cong U(0, \beta \times S) \tag{3}$$
$$X_{\text{move},j}^{t+1} = X_j^t + \alpha,$$

where X_j^t and $X_{\text{move},j}^{t+1}$ are the jth colony of each empire. After this movement for each colony X, a mutant colony X_{mut}^{t+1} is generated as follows:

$$X_{\text{mut},j}^{t+1} = X_{m1}^t + \text{rand}(.) \times (X_{m2}^t - X_{m3}^t) \tag{4}$$
$$X_{\text{mut},j} = [X_{\text{mut},1}, X_{\text{mut},2}, \ldots, X_{\text{mut},b}]_{1\times b},\ b = k \times d.$$

Then the selected colony would be

$$X_{\text{new},j}^{t+1} = [X_{\text{new},1}, X_{\text{new},2}, \ldots, X_{\text{new},b}]_{1\times b}$$

$$X_{\text{new},z} = \begin{cases} X_{\text{mut},z} & \text{if rand}(.) < \gamma \\ X_z & \text{otherwise,} \end{cases} \quad z = 1, 2, \ldots, b, \tag{5}$$

where rand(.) is a random number between 0 and 1, and γ is a number less than 1. m_1, m_2, m_3 are three individuals which are selected from initial colonies randomly. In order to cover the entire colonies uniformly, it is better to select them as $m_1 \neq m_2 \neq m_3 \neq j$. k is the number of clusters, and the dimension of each cluster center will be d. To choose the best colony between $X_{\text{move},j}$ and $X_{\text{new},j}$ to replace the jth colony (X_j), an objective function is used:

$$X_j^{t+1} = \begin{cases} X_{\text{move},j}^{t+1} & \text{if cost}(X_{\text{move},j}^{t+1}) \leq \text{cost}(X_{\text{new},j}^{t+1}) \\ X_{\text{new},j}^{t+1} & \text{otherwise.} \end{cases} \tag{6}$$

3.4. Hybrid K-MICA

As mentioned before, K-means is used for its ease and simplicity for applications. However, it has some drawbacks. First, its result may depend on the initial values. Also, it may converge to a local minimum. Recently, numerous ideas have been used to alleviate this drawback by using global optimization algorithms such as the genetic algorithm (GA) [24], hybrid PSO-SA [25], hybrid PSO-ACO-K [26], and HBMO [27]. In this study, we used a method that was given in [23], called the hybrid K-MICA. According to the original ICA, first a primary population is generated and then empires with

```
Begin
Generate an initial population randomly
Calculate the objective function for the initial population
Sort the initial population based on their objective function values
Select the imperialist states
Divide colonies among imperialist
Use K-means algorithm for each empire
do{
Place one colony as an initial K centroids object that are clustered.
do{
Assign each object to the group that has the closest centroid
Recalculate the positions of the K centroids
}
while (the centroids no longer move)
}
while (all colonies selected)
do{
do{
Select the ith empire
do{
Select the jth colony
Move the colony toward its imperialist state
Use mutation to change the direction of colony
Calculate the objective function value for the two new population
Compare both new cost and select the best one
Replace jth colony with new one
}
while (all colonies selected)
Sort all colonies of ith empire based on their cost functions
Check cost of all colonies in each empire
if there is a colony which has a lower cost than its imperialist
exchange the position of the colony and the imperialist
end if
Update the position of the ith empire
}
while (all empires selected)
calculate total cost of empires
find the weakest empire
Give one of its colony to the winner empire
Check the number of colony in each empire
If there is an empire without colony
Remove empire and give its imperialist to the best empire
End if
Search around the global solution by CLS
One country selected randomly is replaced with the best solution among them
}
While (there is more than one empire)
End
```

Fig. 5. Pseudo-code of the K-MICA algorithm.

their possessions appear. Applying K-means to each empire causes us to improve the initial population of colonies. It makes the hybrid algorithm converge more quickly and prevents it from falling into local optima. The outputs of K-means form the initial empires of the modified ICA.

To improve the outcome of the algorithm, it is better that a powerless imperialist is not removed when it loses all possessions. This imperialist is one of the best answers, and it can contribute in imperialistic competition as a weak colony or as one to be given to a powerful empire. The pseudo-code of this algorithm is shown in Fig. 5.

To apply the ICA for clustering, the following steps have to be taken [23].

Step 1: Generate an initial population.

An initial population of input data is generated by chaos initialization as follows:

$$\text{Population} = \begin{bmatrix} X_1 \\ X_2 \\ \dots \\ X_{N_{\text{initial}}} \end{bmatrix}$$

$$\begin{aligned} X_i &= \text{Country}_i = [\text{center}_1, \text{center}_2, \dots, \text{center}_k] \\ & i = 1, 2, \dots, N_{\text{initial}} \end{aligned} \tag{7}$$

$$\text{center}_j = [x_1, x_2, \dots, x_d] \quad j = 1, 2, \dots, k$$

$$H = k \times d$$

$$X_0 = [X_0^1, X_0^2, \dots, X_0^H]$$

$$x_0^j = \text{rand}(.) \times (x_{\max}^j - x_{\min}^j) + x_{\min}^j, \quad j = 1, 2, \dots, H$$

$$X_i = [x_i^1, x_i^2, \dots, x_i^H], \quad i = 1, 2, \dots, N_{\text{initial}}$$

$$x_i^j = 4 \times x_{i-1}^j \times (1 - x_{i-1}^j), \quad j = 1, 2, \dots, H$$

$$x_j^{\min} < x_j < x_j^{\max},$$

where center_j is the jth cluster center for the ith country. X_i is one of the countries. N_{initial} is the population number and d is the dimension of each cluster center. $x_j^{\max}$ and $x_j^{\min}$ (each feature of center) are the maximum and minimum value of each point referring to the jth cluster center, which are in order. k is the number of clusters. H is the number of state variables. X_0 is an initial solution.

Step 2: Calculate the objective function value.

Suppose that we have N sample feature vectors. The objective function is evaluated for each country as follows:

Step 2-1: $i = 1$ and Objec $= 0$.

Step 2-2: select the ith sample.

Step 2-3: calculate the distances between the ith sample and $\text{center}_j (j = 1, 2, \dots, K)$.

Step 2-4: add the value of Objec with the minimum distance calculated in Step 2-3.

$$(\text{Objec} = \text{Objec} + \min(|\text{center}_i - Y_m|, i = 1, 2, \dots, K)).$$

Step 2-5: if all samples have been selected, go to the next step, otherwise $i = i + 1$ and return to step 2-2.

Step 2-6: $\text{cost}(X) = \text{Objec}$.

The objective function is calculated mathematically as below:

$$\text{cost}(X) = \sum_{m=1}^{N} \min(|\text{center}_i - Y_m|, i = 1, 2, \dots, K). \tag{8}$$

Step 3: Sort the initial population based on the objective function values.

The initial population is increased based on the value of the objective function.

Step 4: Select the imperialist states.

Countries that have the minimum objective function are selected as the imperialist states and the remaining ones form the colonies of these imperialists.

Step 5: Divide the colonies among the imperialists.

Based on the power of each imperialist the colonies are divided among them. The power of each imperialist is calculated as follows.

$$C_n = \max\{\text{cost}\} - \text{cost}_n \tag{9}$$

$$P_n = \left| \frac{C_n}{\sum_{i=1}^{N_{\text{imp}}} C_i} \right| \tag{10}$$

$$C_n^{\text{norm}} = \text{round}(P_n(N_{\text{col}})). \tag{11}$$

In the above equations, cost_n is the cost of the nth imperialist and C_n the normalized cost of each one. The normalized power of each imperialist is introduced as P_n; then the initial number of colonies for each empire will be C_n^{norm}, where N_{col} and N_{imp} are the total numbers of colonies and imperialists.

Step 6: Use the K-means algorithm for each empire.

Step 7: Move colonies toward their imperialist states as described in Section 3.

Step 8: Use mutation to change the direction of colonies. This is mentioned in modified ICA.

Step 9: Check the cost of all colonies in each empire.

During the previous steps, the cost of each colony might have changed. Check the cost of all colonies of an empire. If there is one that has a lower cost than its relevant imperialist, exchange their positions.

Step 10: Check the total cost of each empire.

The cost of each empire depends on the power of both the imperialist and its colonies. It is calculated as follows:

$$TC_n = \text{cost}(\text{imperialist}_n) + \xi \,\text{mean}\{\text{cost}(\text{colonies of empire}_n)\}. \quad (12)$$

TC_n is the total cost of the nth empire, and ξ is an attenuation coefficient between 0 and 1 to reduce the effect of the cost of the colonies.

Step 11: Perform an imperialistic competition.

All empires, according their power (total cost), try to get the colonies of the weakest empire.

$$TC_n^{\text{norm}} = \max\{TC_i\} - TC_n \quad (13)$$

$$P_{P_n} = \left| \frac{TC_n^{\text{norm}}}{\sum_{i=1}^{N_{\text{imp}}} TC_i^{\text{norm}}} \right|, \quad (14)$$

where TC_n^{norm} is the normalized total cost of the nth empire and the possession probability of each empire is P_{P_n}.

A roulette wheel can be used for stochastic selection of the winning empire which will dominate the weakest colony of the weakest empire. To perform the roulette wheel algorithm, it is necessary to calculate the cumulative probability as follows:

$$C_{P_n} = \sum_{i=1}^{n} P_{P_n}.$$

According to this equation, the cumulative probability for $n = 1$ is equal to its probability, while for the last n it corresponds to 1.

Then a random number with uniform distribution is generated and compared with all C_{P_n}.

Each sector with higher probability will have more chance to be chosen. Therefore the winner empire will specify.

As mentioned, to use the roulette wheel algorithm, computing the cumulative distribution function is essential. To reduce this time-consuming step an approach has been presented as follows:

$$P = [P_{P_1}, P_{P_2}, \ldots, P_{P_{N_{\text{imp}}}}] \quad (15)$$

$$R = [r_1, r_2, \ldots, r_{N_{\text{imp}}}] \qquad r_1, r_2, \ldots, r_{N_{\text{imp}}} \approx U(0, 1) \quad (16)$$

$$D = P - R = [D_1, D_2, \ldots, D_{N_{\text{imp}}}] = [P_{P_1} - r_1, P_{P_2} - r_2, \ldots, P_{P_{N_{\text{imp}}}} - r_{N_{\text{imp}}}], \quad (17)$$

where P is the vector of possession probability of all empires and R is a vector with uniformly distributed random numbers. The maximum index in D shows the winner empire that gets the colony.

After realizing the winner empire, the weakest colony of the weakest empire will be given to the winner empire. Then we should subtract one of the populations of this weak empire and add one to the winner's population.

Step 12: Remove the weakest empire.

If there is any empire without a colony, eliminate it. Replace one of the weakest colonies of the best empire (low cost) with this imperialist.

Step 13: Apply chaotic local search (CLS) to search around the global solution. The ICA has gained much attention and widespread applications in different fields. However, it often converges to local optima. In order to avoid this shortcoming, a CLS algorithm is used to search around the global solution in this study. CLS is based on the logistic equation as follows:

$$Cx_i = [Cx_i^1, Cx_i^2, \ldots, Cx_i^H]_{1\times H}, \quad i = 0, 1, 2, \ldots, N_{\text{chaos}} \quad (18)$$

$$Cx_{i+1}^i = 4 \times Cx_i^j \times (1 - Cx_i^j), \quad j = 1, 2, \ldots, H$$

$$Cx_0^j = \text{rand}(.)$$

$$Cx_i^j \int [0, 1], \quad Cx_0^j \notin \{0.25, 0.5, 0.75\}.$$

In the CLS, the best solution is considered as an initial solution X_{ds}^0 for the CLS. X_{cls}^0 is scaled into (0, 1) according to the following equation:

$$X_{\text{cls}}^0 = [X_{ds,0}^1, X_{ds,0}^2, \ldots, X_{\text{cls},0}^H]_{1\times H} \quad (19)$$

$$Cx = [Cx_0^1, Cx_0^2, \ldots, Cx_0^H]$$

$$Cx_0^j = \frac{x_{\text{cls},0}^j - x_{\min}^j}{x_{\max}^j - x_{\min}^j}, \quad j = 1, 2, \ldots, H.$$

The chaos population for CLS is generated as follows:

$$X_{\text{cls}}^i = [X_{\text{cls},i}^1, X_{\text{cls},i}^2, \ldots, X_{\text{cls},i}^H]_{1\times H}, \quad i = 1, 2, \ldots, N_{\text{chaos}} \quad (20)$$

$$x_{\text{cls},i}^j = cx_{i-1}^j \times (x_{\max}^j - x_{\min}^j) + x_{\min}^j, \quad j = 1, 2, \ldots, H$$

where cx_i^j indicates the jth chaotic variable, and N_{chaos} is the number of individuals for th CLS. rand(.) is a random number between 0 and 1.

The objective function is evaluated for all individuals of the CLS. One country selected randomly is replaced with the best solution among them.

Step 14: Check the number of empires.

If there is just one empire remaining, stop. Else go to step 7.

4. Classifiers

This section briefly describes the neural network classifiers.

4.1. Multi-layer perceptron (MLP) neural networks

An MLP neural network consists of an input layer (of source nodes), one or more hidden layers (of computation nodes), and an output layer. The recognition basically consists of two phases: training and testing. In the training stage, weights are calculated according to the chosen learning algorithm. The issue of the learning algorithm and its speed is very important for the MLP model. In this study, the following learning algorithms are considered.

4.1.1. Back-propagation with momentum (BP with momentum) algorithm

The BP algorithm makes use of gradient descent with a momentum term to smooth out oscillation [28]. Eq. (21) gives the weight update for BP with momentum:

$$\Delta W_{ij}(t+1) = -\varepsilon \frac{\delta E}{\delta W_{ij}}(t) + \mu \frac{\delta E}{\delta W_{ij}}(t-1), \quad (21)$$

where w_{ij} represents the weight value from neuron j to neuron i, ε is the learning rate parameter, and E represents the error function. It adds an extra momentum parameter, μ, to the weight changes.

4.1.2. Resilient back-propagation (RPROP) algorithm

The RPROP algorithm considers the sign of derivatives as the indication for the direction of the weight update [29]. In doing so, the size of the partial derivative does not influence the weight step. The following equation shows the adaptation of the update values of Δ_{ij} (weight changes) for the RPROP algorithm. For initialization, all are set to small positive values:

$$\Delta_{ij}(t) = \begin{cases} \eta^+ \times \Delta_{ij}(t-1); & \text{if } \frac{\delta E}{\delta W_{ij}}(t-1)\frac{\delta E}{\delta W_{ij}}(t) > 0 \\ \eta^- \times \Delta_{ij}(t-1); & \text{if } \frac{\delta E}{\delta W_{ij}}(t-1)\frac{\delta E}{\delta W_{ij}}(t) < 0 \\ \eta^0 \times \Delta_{ij}(t-1); & \text{otherwise,} \end{cases} \tag{22}$$

where $\eta^0 = 0, 0 < \eta^- < 1 < \eta^+, \eta^{-,0,+}$ are known as the update factors. Whenever the derivative of the corresponding weight changes its sign, this implies that the previous update value is too large and it has skipped a minimum. Therefore, the update value is then reduced (η^-), as shown above. However, if the derivative retains its sign, the update value is increased (η^+). This will help to accelerate convergence in shallow areas. To avoid over-acceleration, in the epoch following the application of (η^+), the new update value is neither increased nor decreased (η^0) from the previous one. Note that the values of Δ_{ij} remain non-negative in every epoch. This update value adaptation process is then followed by the actual weight update process, which is governed by the following equations:

$$\Delta W_{ij}(t) = \begin{cases} -\Delta_{ij}; & \text{if } \frac{\delta E}{\delta W_{ij}}(t) > 0 \\ +\Delta_{ij}; & \text{if } \frac{\delta E}{\delta W_{ij}}(t) < 0 \\ 0; & \text{otherwise.} \end{cases} \tag{23}$$

The values of the training parameters adopted for the algorithms were determined empirically. They were $\eta^- = .05$, $\eta^+ = 1.2$.

4.2. Radial basis function neural networks (RBFNNs)

RBF neural networks with their structural simplicity and training efficiency are a good candidate to perform a nonlinear mapping between the input and the output vector space. The RBFNN is fully connected feed forward structure and it consists of three layers, namely, an input layer, a single layer of nonlinear processing units, and an output layer. The input layer is composed of input nodes that are equal to the dimension of the input vector x. The output of the jth hidden neuron with Gaussian transfer function can be calculated as

$$h_j = \exp(-\|x - c_j\|^2/\sigma^2), \tag{24}$$

where h_j is the output of the jth neuron, $x \in \Re^{n\times 1}$ is an input vector, $c_j \in \Re^{n\times 1}$ is the jth RBF center, σ is the center spread parameter, which controls the width of the RBF, and $\|.\|^2$ represent the Euclidean norm. The output of any neuron at the output layer of the RBF network is calculated as

$$y_i = \sum_{j=1}^{k} w_{ij} h_j, \tag{25}$$

where w_{ij} is the weight connecting hidden neuron j to output neuron i and k is the number of hidden layer neurons.

Table 1
K-MICA parameters for clustering.

Parameter	Value
N_{pop}	50
N_{imp}	12
γ	0.4
ζ	0.1
β	20
Maximum number of iterations	100

4.3. Probabilistic neural networks (PNNs)

A probabilistic neural network (PNN) is a kind of radial basis network suitable for classification problems. PNNs have three layers: input, pattern, and summation. The input layer has as many elements as there are individual parameters needed to describe the samples to be classified [30]. The pattern layer organizes the training set in such a way that an individual processing element represents each input vector. The pattern units in the probabilistic network are used to store pattern examples, taken directly from the training data. The entire set of training data is used, and so the number of units in the first hidden layer is set equal to the number of training cases. The summation layer has as many processing elements as there are classes to be recognized, and simply collects the outputs from all hidden neurons of each respective class. The products of the summation layer are forwarded to the output (one neuron for each data class), where the estimated probability of the new pattern being a member of that data class is computed. The transfer function is a radial basis function for the first layer and is a competitive function for the second layer. Only the first layer has biases. Training of the probabilistic neural network is much easier than with back-propagation. It can be simply finished by setting the weights of the network using the training set. The outputs of summary layer are binary values. If y_i is larger than input of other neurons (which means that this input pattern belongs to class i), y_i is set to 1, otherwise it is set to zero.

5. Simulation results

In this section, the performance of the proposed recognizer is evaluated. For this purpose, we have used practical and real-world data [31]. This dataset contains 600 examples of control charts. For this study, we have used 60% of the data for training the classifier and the rest for testing. The easiest way to assess the performance rate is to choose a test set independent of the training set and validation set to classify its examples, count the examples that have been correctly classified, and divide by the size of the test set. The proportion of test-set examples that are classified correctly to the total samples estimates the performance of the recognizer for each pattern. In order to obtain the recognition accuracy (RA) of system, one needs to compute the average value of the performances of the CCPs.

We apply the K-MICA for finding the optimum cluster centers. The parameters of the clustering algorithm used in this study are shown in Table 1. These values were selected for the best performance after several experiments.

As an example, patterns' Euclidean distance of signals from cluster center 2 (normal pattern) is shown in Fig. 6. The horizontal axis shows the number of signals and the vertical axis shows the Euclidean distance of signals from cluster center 2. As shown in the figure, the normal pattern signals have smaller values due to other signals.

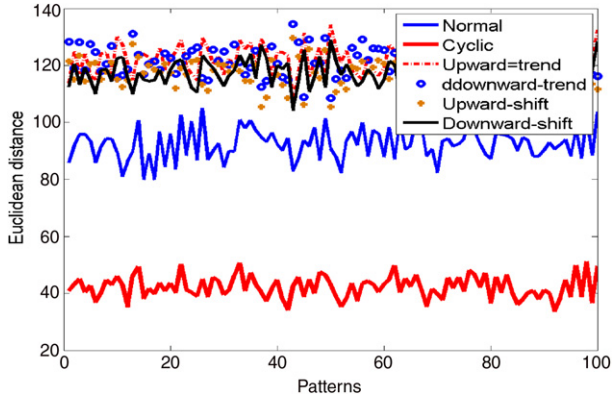

Fig. 6. Euclidean distance of signals from cluster center 2.

Table 2
MLP architecture and training parameters.

Number of layers	2
Number of output neurons	6
Learning algorithm	Resilient back-propagation Back-propagation with momentum
The initial weights and basis	Random
Activation function (RP)	Tangent-sigmoid Linear

Table 3
Performance comparisons of different classifiers with row data.

Classifier	Parameter	RA (%)
MLP (RP)	NNHL = 20	93.81
MLP (with momentum)	NNHL = 10	93.57
RBFNN	Spread = 2	95.33
PNN	Spread = 5	93.75

Table 4
Performance comparisons of different classifiers with Euclidean distance of signals from cluster center.

Classifier	Parameter	RA (%)
MLP (RP)	NNHL = 20	99.65
MLP (with momentum)	NNHL = 30	99.23
RBFNN	Spread = 7	99.53
PNN	Spread = 3	96.42

5.1. Performance comparison of different neural network and proposed features

The training parameters and the configuration of the MLP used in this study are shown in Table 2. The MLP classifiers were tested with various neurons for a single hidden layer, and the best networks were selected. For the PNN and RBFNN, a Gaussian activation function and a single hidden layer with 360 neurons were considered. These values were selected for the best performance after several experiments.

Tables 3 and 4 show the recognition accuracy (RA) of different systems. In these tables, NNHL means the number neurons in the hidden layers. The obtained results are the average of 10 independent runs. As depicted in Table 3, using various neural networks and unprocessed data (raw data), the highest accuracy is 95.33%, which is achieved by the RBF neural network. From Table 4, it can be found that, by using the proposed features as the input of the classifier, the accuracy of the system is increased to 99.65%. This accuracy value is achieved by the MLP neural network with the RP learning algorithm.

A comparison between Tables 3 and 4 shows the efficiency of the proposed feature. As can be recognized from these tables, the accuracy of MLP (RP), MLP (momentum), the PNN, and the RBF neural network, are increased to 99.65%, 99.23%, 99.53%, 96.42%, respectively, which shows the significant role of the feature in classifier performance.

In order to indicate the details of the recognition for each pattern, the confusion matrix of the recognizer without optimization is shown in Table 5. As we know, the values in the diagonal of the confusion matrix show the correct performance of the recognizer for each pattern. In other words, these values show how many of considered patterns are recognized correctly by the system. The other values show the mistakes of the system. For example, look at the third row of this matrix. The value of 97.90% shows the percentage of correct recognition of upward trend patterns and the value of 2.10% shows that this type of pattern is wrongly recognized as upward shift patterns. In order to obtain the recognition accuracy (RA) of the system, it is needed to compute the average value that will be appeared in the diagonal of the confusion matrix.

Table 6
Comparison of the effect of the clustering algorithm.

Classifier	Clustering algorithm	RA (%)
MLP-RP	GA	98.6
MLP-RP	FCM	97.2
MLP-RP	KMC	97.3
MLP-BP with momentum	GA	99.04
MLP-BP with momentum	FCM	97.18
MLP-BP with momentum	KMC	97.74
RBF	GA	98.73
RBF	FCM	97.85
RBF	KMC	97.44
PNN	GA	96.7
PNN	FCM	93.9
PNN	KMC	92.4

Table 7
Recognition accuracy of the recognizer for different values of parameters.

Case	N_{pop}	N_{imp}	β	ξ	γ	Classifier	RA (%)
1	50	12	25	0.5	0.6	MLP(RP)	99.54
2	50	10	25	0.1	0.4	MLP(RP)	99.55
3	50	8	25	0.05	0.2	MLP(RP)	99.60
4	**50**	**12**	**20**	**0.1**	**0.4**	**MLP(RP)**	**99.65**
5	50	10	20	0.5	0.8	MLP(RP)	99.51
6	50	8	20	0.05	0.6	MLP(RP)	99.61
7	50	10	15	0.5	0.8	MLP(RP)	99.56
8	50	8	15	0.05	0.4	MLP(RP)	99.63
9	50	10	10	0.1	0.8	MLP(RP)	99.60
10	50	8	10	0.05	0.6	MLP(RP)	99.62

Table 5
Confusion matrix for best result.

	Normal	Cyclic	Upward trend	Downward trend	Upward shift	Downward shift
Normal	100%	0	0	0	0	0
Cyclic	0	100%	0	0	0	0
Upward trend	0	0	97.9%	0	2.1%	0
Downward trend	0	0	0	100%	0	0
Upward shift	0	0	0	0	100%	0
Downward shift	0	0	0	0	0	100%

Table 8
A summary of different classification algorithms together with their reported results used measures of the accuracy.

Ref. no	Number of CCP types	Input	Classifier	Total recognition accuracy (%)
[5]	6	Unprocessed data	MLP(RSFM)	97.46
[6]	6	Unprocessed data	MLP	94.30
[7]	6	Unprocessed data	PNN	95.58
[8]	6	Unprocessed data	MLP	93.73
[9]	6	Unprocessed data	LVQ	97.7
[35]	4	Unprocessed data	MLP(SPA)	96.38
This work	6	Euclidean distance	MLP	99.65

5.2. Performance comparison of different clustering algorithms

The performance of the classifiers has been compared with different clustering algorithms. For this purpose, fuzzy C-mean clustering [32,33], *K*-mean clustering [18,19], and genetic algorithm clustering [34] are considered. Table 6 shows the RA of different systems. From Tables 4 and 6, it can be seen that the K-MICA clustering based system achieves higher recognition accuracy, 99.65%.

5.3. Effects of the K-MICA parameters on the performance of the method

In this subsection, we report the sensitivity of the recognition system with respect to N_{imp}, N_{pop}, β_ξ, and γ, which control the behavior, and thus the goodness of the K-MICA search process. The results obtained for 10 sets of parameters are shown in Table 7. To investigate the influence of the parameters in each case, the neural network is tested 10 times independently with various numbers of neurons in the hidden layer. The average of the best obtained value is given in the table. It illustrates that this hybrid system has a little dependency on variation of the parameters.

5.4. Comparison of the proposed method (HIM) with other methods in the literature

Several researchers in the past have addressed arrhythmia detection and the classification problem using the CCP signals directly or by analyzing the pattern rate variability signal. Direct comparison with other works is difficult for control chart pattern recognition problems. This is mainly because of the fact that there is no single unified data set available. A different setup of patterns (for example, the number of training and testing samples and the number of patterns) will lead to different performance. Besides, there are many different kinds of benchmarking system used for system quality. This causes difficulties for direct numerical comparison. Table 8 compares the different methods in case of: the recognition accuracy, the used classifier and the used inputs.

As for neural network-based CCP recognizers, Le et al. [5] introduced a new ANN model, and their numerical simulations showed that this model has a recognition accuracy of about 97.46% for recognition of six types of CCP. Pham and Oztemel [6] reported a generalization rate of 94.30%. Cheng and Ma [7] have gained recognition accuracy (RA) of about 95.58%. However, the performance for lower patterns is reported to be less than 90%. The proposed method in [8] reached a RA of about 93.73% of classification accuracy for recognition of six types of CCP. In [9], the authors used an LVQ neural network and achieved an RA of about 97.70%. In [35], Guh and Tannock proposed a sequential pattern analysis method and reported a classification rate of about 96.38%. Compared to these papers, an effective system is proposed in the current work which provides a better accuracy over a wider range of different types of CCP (six different classes).

6. Conclusion

Accurate recognition of control chart patterns (CCPs) is very important for producing high-quality products. This study investigated the design of an accurate system for automatic recognition of CCPs. Here the usage of the Euclidean distance of the patterns from the cluster centers is proposed as the efficient features. For extraction of these features we have used a combination of the *K*-means algorithm and modified imperialist competitive algorithm (MICA). This algorithm is named as K-MICA. The extracted features are applied to the different types of the neural networks. Simulations result show that the MLP neural network with RP learning algorithm has the highest rate of the recognition accuracy (RA). This RA is about 99.65%.

References

[1] Montgomery DC. Introduction to statistical quality control. 5th ed. Hoboken (NJ, USA): John Wiley; 2005.
[2] Swift JA, Mize JH. Out-of-control pattern recognition and analysis for quality control charts using LISP-based systems. Computers & Industrial Engineering 1995;28:81–91.
[3] Wang CH, Kuo W. Identification of control chart patterns using wavelet filtering and robust fuzzy clustering. Journal of Intelligent Manufacturing 2007;18:343–50.
[4] Wang CH, Guo RS, Chiang MH, Wong JY. Decision tree based control chart pattern recognition. International Journal of Production Research 2008; 124–34.
[5] Le Q, Goal X, Teng L, Zhu M. A new ANN model and its application in pattern recognition of control charts. In: Proc. IEEE. WCICA. 2008. p. 1807–11.
[6] Pham DT, Oztemel E. Control chart pattern recognition using neural networks. Journal of Systems Engineering 1992;2:256–62.
[7] Cheng Z, Ma Y. A research about pattern recognition of control chart using probability neural network. In: Proc. ISECS. 2008. p. 140–5.
[8] Sagiroujlu S, Besdoc E, Erler M. Contro chart pattern recognition using artificial neural networks. Turkish Journal of Electrical Engineering 2000;8:137–47.
[9] Pham DT, Oztemel E. Control chart pattern recognition using linear vector quantization networks. International Journal of Production Research 1994; 256–62.
[10] Guh RS. Robustness of the neural network based control chart pattern recognition system to non-normality. International Journal of Quality and Reliability Management 2002;19:97–112.
[11] Gauri SK, Chakraborty S. A study on the various features for effective control chart pattern recognition. International Journal of Advanced Manufacturing Technology 2007;34:385–98.
[12] Gou Y, Dooley KJ. Identification of change structure in statistical process control. International Journal of Production Research 1992;30:1655–69.
[13] Wani MA, Rashid S. Parallel algorithm for control chart pattern recognition. In: Proc. IEEE the fourth international conference on machine learning and applications. ICMLA'05. 2005.
[14] Guh RS, Shiue YR. On-line identification of control chart patterns using self-organized approaches. International Journal of Production Research 2005;43: 1225–54.
[15] Pacella M, Semeraro Q, Anglani A. Adaptive resonance theory-based neural algorithms for manufacturing process quality control. International Journal of Production Research 2004;43:4581–607.
[16] Al-Ghanim AM, Ludeman LC. Automated unnatural pattern recognition on control charts using correlation analysis techniques. Computers & Industrial Engineering 1997;32:679–90.
[17] Yang JH, Yang MS. A control chart pattern recognition scheme using a statistical correlation coefficient method. Computers & Industrial Engineering 2005;48:205–21.
[18] Morales AK, Erazo FR. A search space reduction methodology for data mining in large databases. Engineering Applications of Artificial Intelligence 2009;22: 57–65.
[19] Sakai T, Imiya A. Unsupervised cluster discovery using statistics in scale space. Engineering Applications of Artificial Intelligence 2009;22:92–100.

[20] Atashpaz-Gargari E, Lucas C. Designing an optimal PID controller using colonial competitive algorithm. In: Proceedings of the first Iranian joint congress on fuzzy and intelligent systems. 2007.
[21] Atashpaz-Gargari E, Lucas C. Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: Proceedings of the IEEE congress on evolutionary computation. 2007. p. 4661–7.
[22] Rajabioun R, Hashemzadeh F, Atashpaz-Gargari E. Colonial competitive algorithm a novel approach for PID controller design in MIMO distillation column process. International Journal of Intelligent Computing and Cybernetics 2008; 3:337–55.
[23] Niknam T, Taherian Fard E, Pourjafarian N, Rousta A. An efficient hybrid algorithm based on modified imperialist competitive algorithm and *K*-means for data clustering. Engineering Applications of Artificial Intelligence 2011;24: 306–17.
[24] Krishna K, Murty M. Genetic *k*-means algorithm. IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics) 1999;29:433–9.
[25] Niknam T, Amiri B, Olamaie J, Arefi A. An efficient hybrid evolutionary optimization algorithm based on PSO and SA for clustering. Journal of Zhejiang University Science 2009;10:512–9.
[26] Firouzi B, Sadeghi MS, Niknam T. A new hybrid algorithm based on PSO, SA, and *K*-means for cluster analysis. International Journal of Innovative Computing, Information and Control 2010;6:1–10.
[27] Fathian M, Amiri B. A honey-bee mating approach on clustering. The International Journal of Advanced Manufacturing Technology 2007;38(7–8): 809–21.
[28] Haykin S. Neural networks: a comprehensive foundation. New York: MacMillan; 1999.
[29] Riedmiller M, Braun H. A direct adaptive method for faster back-propagation learning: the RPROP algorithm. In: Proc. ICNN. 1993. p. 586–91.
[30] Specht DF. Probabilistic neural networks. Neural Networks 1990;3:109–18.
[31] http://archive.ics.uci.edu/ml/databases/syntheticcontrol/syntheticcontrol.data.html.
[32] Dunn JC. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. Journal of Cybernetics 1973;3:32–57.
[33] Bezdek JC. Pattern recognition with fuzzy objective function algorithms. New York: Plenum Press; 1981.
[34] Murthy CA, Chowdhury N. In search of optimal clusters using genetic algorithms. Pattern Recognition Letters 1996;17:825–32.
[35] Guh RS, Tannock JDT. A neural network approach to characterize pattern parameters in process control charts. Journal of Intelligent Manufacturing 1999;10:449–62.