Self-adaptive Parameter and Strategy based Particle Swarm Optimization for Large-scale Feature Selection Problems with Multiple Classifiers

Yu Xue[a,c,*], Tao Tang[a], Wei Pang[b], Alex X. Liu[c]

[a] *School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044, China*
[b] *School of Natural and Computing Sciences, University of Aberdeen, Aberdeen AB24 3UE, UK*
[c] *Department of Computer Science and Engineering, Michigan State University, East Lansing 48824, USA*

Abstract

Feature selection has been widely used in classification for improving classification accuracy and reducing computational complexity. Recently, evolutionary computation (EC) has become an important approach for solving feature selection problems. However, firstly, as the datasets processed by classifiers become increasingly large and complex, more and more irrelevant and redundant features may exist and there may be more local optima in the large-scale feature space. Therefore, traditional EC algorithms which have only one candidate solution generation strategy (CSGS) with fixed parameter values may not perform well in searching for the optimal feature subsets for large-scale feature selection problems. Secondly, many existing studies usually use only one classifier to evaluate feature subsets. To show the effectiveness of evolutionary algorithms for feature selection problems, more classifiers should be tested. Thus, in order to efficiently solve large-scale feature selection problems and to show whether the EC-based feature selection method is efficient for more classifiers, a self-adaptive parameter and strategy based particle swarm optimization (SPS-PSO) algorithm is proposed in this paper using multiple classifiers. In SPS-PSO, a representation scheme of solutions and five CSGSs have been used. To automatically adjust the CSGSs and their parameter values during the evolutionary process, a strategy self-adaptive mechanism and a parameter self-adaptive mechanism are employed in the framework of particle swarm optimization (PSO). By using the self-adaptive mechanisms, the SPS-PSO can adjust both CSGSs and their parameter values when solving different large-scale feature selection problems. Therefore, SPS-PSO has good global and local search ability when dealing with these large-scale problems. Moreover, four classifiers, i.e., k-nearest neighbor

*Corresponding author
Email addresses: xueyu@nuist.edu.cn (Yu Xue), taotang1993@163.com (Tao Tang), pang.wei@abdn.ac.uk (Wei Pang), alexliu@cse.msu.edu (Alex X. Liu)

(KNN), linear discriminant analysis (LDA), extreme learning machine (ELM), and support vector machine (SVM), are individually used as the evaluation functions for testing the effectiveness of feature subsets generated by SPS-PSO. Nine datasets from the UCI Machine Learning Repository and Causality Workbench are used in the experiments. All the nine datasets have more than 600 dimensions, and two of them have more than 5,000 dimensions. The experimental results show that the strategy and parameter self-adaptive mechanisms can improve the performance of the evolutionary algorithms, and that SPS-PSO can achieve higher classification accuracy and obtain more concise solutions than those of the other algorithms on the large-scale feature problems selected in this research. In addition, feature selection can improve the classification accuracy and reduce computational time for various classifiers. Furthermore, KNN is a better surrogate model compared with the other classifiers used in these experiments.

Keywords: Particle Swarm Optimization, Feature Selection, Large-scale Problems, Self-adaptive, Classification.
2010 MSC: 00-01, 99-00

1. Introduction

Feature selection, as an important technique in machine learning and data mining, has been widely studied over the last two decades [1]. The fields of application of feature selection include credit rating [2], cancer diagnosis [3], image classification [4], intrusion detection [5], medical technology [6], and so on.

Usually, many irrelevant and redundant features exist in the datasets which the classifiers deal with. In order to reduce the computational cost and increase classification accuracy, feature selection is commonly used. Feature selection can improve the classification accuracy and simplify the classification tasks. However, with the advent of the Big Data Era, the processing of the large-scale (high-dimensional) datasets has become unavoidable, and the computational time increases exponentially with the increase of the number of features. Given a dataset with n features, if n is small, the total number of feature subsets is also small. Thus, the optimal feature subset can even be obtained via an exhaustive search. However, as n becomes larger, it would reach a point where it would no longer be possible to enumerate all feature subsets. Therefore, it has become necessary to design efficient algorithms for finding small, high-quality feature subsets for large-scale problems [7].

Over the past few decades, many heuristic feature selection methods have been proposed. Among them, sequential forward selection (SFS) [7] and sequential backward selection (SBS) [7] are two typical approaches. These two methods first start from either an empty set or a complete set then continually add or remove features to optimize the feature subset. The disadvantage of SFS and SBS is that the features that have already been added or removed can no

longer be removed or added again. Actually, some features are not completely independent, i.e., they are interrelated with each other. Later on, to overcome this disadvantage, a plus-l minus-r selection method (LRS) [8] was proposed. This approach can mitigate the aforementioned issue with SFS and SBS, but it still cannot overcome the issue of the algorithms easily falling into local optima. Sequential floating forward selection (SFFS) [9] and sequential floating backward selection (SFBS) [10] were then designed based on LRS. Although these heuristic methods can greatly reduce the time complexity compared to that of exhaustive search, they tend to be easily trapped into local optima because of their fixed searching strategies.

Many kinds of evolutionary computation (EC) techniques have been developed in the past several decades and are widely used in real-world applications because they can find the best possible solution(s) within acceptable periods [1]. Some of these methods have been employed for solving the feature selection problems [1, 12, 13]. For example, the following algorithms have been used for feature selection problems: binary dragonfly algorithm [14], grasshopper optimization approach with evolutionary population dynamics [15], whale optimization approach [16], binary slap swarm algorithm [17], binary slap swarm algorithm with asynchronous updating rules and a new leadership structure [18], whale optimization algorithm with simulated annealing [19], genetic algorithm (GA) [20, 21, 2], artificial bee colony [22, 23], differential evolution (DE) [24], ant colony optimization [25], water drop algorithm [26], lion's algorithm [27], grey wolf optimizer [28], shuffled frog leaping algorithm [29], brain storm optimization [30], particle swarm optimization (PSO) [31], and so on.

Because of the stochastic nature of EC techniques, they can more easily escape from local optima compared with heuristic methods. Therefore, they can more easily obtain the global optimal solutions. However, the solution space of a feature selection problem increases dramatically with the increase of the number of features; meanwhile, the number of irrelevant and redundant features may also grow rapidly, and they may generate many local optima in this huge solution space. These local optima bring big challenges to the EC techniques in finding the global optimum. Furthermore, most EC techniques can achieve good performance only on small-scale feature selection problems. For large-scale feature selection problems, the issue of falling into local optima becomes more prominent. In addition, different datasets may have different characteristics. These characteristics require the EC techniques to have the ability to adaptively adjust their candidate solution generation strategies (CSGSs) and the parameters associated with these CSGSs.

The self-adaptive mechanism of EC has attracted much attention to researchers over the last decade, and many EC methods with self-adaptive mechanisms have been proposed. In general, the EC algorithms with self-adaptive mechanisms can adjust their CSGS(s) or parameter(s) automatically during the evolution process. There are several self-adaptive evolutionary algorithms presented in the literature [32, 33, 34, 35, 36, 37, 38, 39]. For example, Wang *et al.* proposed a self-adaptive learning based particle swarm optimization (SLPSO) [33]. In SLPSO, a probability model is designed to simultaneously adopted

four PSO based search strategies. Besides, Li *et al.* proposed another self-learning particle swarm optimizer to deal with different complex problems, and they have proved the superior performance of the self-adaptive evolutionary algorithm by assessing their algorithm on 45 test functions and two real-world problems [34]. Similarly, Xue *et al.* [39] proposed a self-adaptive artificial bee colony with symmetry initialization (SABC-SI). In SABC-SI, a novel population initialization method based on half space and symmetry is designed; such a method can increase the diversity of initial solutions. Furthermore, several different self-adaptive PSO algorithms have also been proposed and they have been employed to solve different real-world problems, such as the economic dispatch problem [35], multidimensional knapsack problem [36], shop scheduling problem [37], etc. For all of the algorithms previously mentioned, the self-adaptive mechanisms have been effectively applied into the EC methods, and the superiority of the EC algorithms with self-adaptive mechanisms has been proven in the aforementioned studies. To the best of our knowledge, although many self-adaptive EC algorithms have been proposed, these algorithms are rarely used to solve feature selection problems. Recently, a self-adaptive PSO using the strategy self-adaptive mechanism has been developed for solving feature selection problems [40]. However, this existing work lacked the use of the parameter self-adaptive mechanism. Therefore, in this research, the parameter self-adaptive mechanism is also introduced into EC algorithms for solving the feature selection problems; t is combined with the strategy self-adaptive method to further enhance the performance of EC in different evolution stages. A strategy and parameter self-adaptive based EC algorithm is expected to be better able to avoid falling-into-local-optima when solving feature selection problems, especially for large-scale problems. PSO, which was proposed by Eberhart and Kennedy in 1995 [41], is a simple and very powerful optimizer [31]. Recently, PSO and its variations are being widely employed for solving the feature selection problems, and many PSO-based approaches have shown promising results [42, 43, 44, 3, 45]. Thus, within the framework of PSO, a new self-adaptive parameter and strategy based particle swarm optimization (SPS-PSO) algorithm is proposed in this research. The algorithm has the parameter and strategy self-adaptive mechanisms which try to make the algorithm more adaptable to different feature selection problems. In addition, to investigate whether the feature selection approach works well for different classifiers, four classifiers, including the k-nearest neighbor (KNN) [46], latent dirichlet allocation (LDA) [47], support vector machine (SVM) [48, 49], and extreme learning machine (ELM) [50], are employed as evaluation functions, respectively. To assess the performance of the proposed algorithm on large-scale feature selection problems, nine datasets from the UCI Machine Learning Repository [51] and Causality Workbench [52] are used, and all these datasets have more than 600 dimensions, with two of them having more than 5,000 dimensions. The main contributions of this research are listed as follows:

(1) A self-adaptive parameter and strategy based particle swarm optimization (SPS-PSO) algorithm is proposed for feature selection problems. This algorithm

employs the strategy self-adaptive mechanism and the parameter self-adaptive mechanism at the same time to improve the ability of the algorithm to search for optimal solutions and to adapt to different feature selection problems.

(2) This study involves feature selection problems with large-scale dimensions, in contrast with most previous studies, which are performed using small-scale problems. This research deals with the nine datasets with more than 600 dimensions, including two datasets with more than 5,000 dimensions.

(3) Unlike most of the existing approaches which usually use only one classifier to evaluate feature subsets, in this research, four different classifiers, i.e., KNN, LDA, SVM, and ELM, are tested as evaluation functions, respectively.

The rest of the paper is organized as follows: Section 2 introduces the proposed SPS-PSO algorithm, including five CSGSs and the detailed steps of the strategy self-adaptive method and parameter self-adaptive method. Section 3 presents the information of the datasets and the parameter values used for the algorithms, and it also describes the purpose of the two stages of the experiments. Section 4 reports the results and analyses them from multiple perspectives. Finally, Section 5 concludes the paper and proposes future research directions.

2. Self-adaptive Parameter and Strategy based Particle Swarm Optimization (SPS-SPO)

In SPS-PSO, multiple CSGSs are self-adaptively used. Similarly, the parameters of CSGSs are flexible and can be adjusted automatically according to different problems during the evolution process. This section focuses on describing the proposed algorithm and the necessary techniques for solving feature selection problems.

2.1. Representation of Solutions

The goal of feature selection is to select a suitable feature subset from available features. For each feature, there are two statuses: selected or unselected. To conveniently represent feature subsets, we transform the feature selection problem into a combinatorial optimization problem. A D-dimensional vector $\mathbf{B}$ is used to represent a solution, where each value of a dimension in $\mathbf{B}$ belongs to $\{0, 1\}$. If the value of a dimension in $\mathbf{B}$ is 1, it indicates that the corresponding feature is selected, whereas 0 indicates that the corresponding feature is unselected.

In fact, a D-dimensional vector $\mathbf{BI}$ consisting of continuous values is firstly generated to represent each particle in the initial population of PSO. A threshold θ $(0 < \theta < 1)$ is then used to map $\mathbf{BI}$ to $\mathbf{B}$ as in [53]. If the value of a dimension in $\mathbf{BI}$ is greater than θ, the corresponding dimension in $\mathbf{B}$ will be set to 1, and to 0 otherwise.

2.2. Initialization and Update Mechanisms

In [53], Xue *et al.* proposed three new initialization methods for PSO to solve the feature selection problems, and they have found the most promising initialization method. In this research, the same initialization method as in [53] is employed. Furthermore, in [53], four new mechanisms are designed for updating *pbest* (personal best) and *gbest* (global best) [41], where *pbest* represents the best solution of a particle, and *gbest* represents the global best solution so far within the population. Because the second method has been proven to be more efficient than the others in [53], the second update strategy is used in this research.

2.3. Candidate Solution Generation Strategies(CSGSs)

The proposed SPS-PSO algorithm uses the following five CSGSs. These CSGSs have been proven to be effective in [53, 54, 33, 55].

1) The CSGS in [53] is used, and it is described as follows:

$$v_{i,d}^{t+1} = w * v_{i,d}^{t} + c_1 * r_1 * \left(p_{i,d} - x_{i,d}^{t}\right) + c_2 * r_2 * \left(p_{g,d} - x_{i,d}^{t}\right), \tag{1}$$

$$x_{i,d}^{t+1} = x_{i,d}^{t} + v_{i,d}^{t+1} \tag{2}$$

where t represents the t^{th} iteration in the evolutionary process. D is the dimensionality of the search space and $d \in D$ represents the d^{th} dimension. w is an inertia weight. $x_{i,d}^{t}$ represents the d^{th} dimension of the current particle's position. $v_{i,d}^{t} \in [-v_{\max}, v_{\max}]$ represents the velocity of the i^{th} particle in the current iteration t. c_1 and c_2 are acceleration constants. r_1 and r_2 are random values uniformly distributed in the range of [0, 1]. $p_{i,d}$ and $p_{g,d}$ represent the d^{th} elements of personal best solution and global best solution, respectively.

2) The position update strategy proposed in [54] is used in SPS-PSO as one CSGS. It is described as follows:

$$x_{i,d}^{t+1} = r_1 * x_{i,d}^{t} + r_2 * p_{g,d} + r_3 * \left(x_{a,d}^{t} - x_{b,d}^{t}\right), \tag{3}$$

where $x_{a,d}^{t}$ and $x_{b,d}^{t}$ are position vectors of two random particles. The velocity update method and other variables have the same meaning as described previously.

3) Estimation-based velocity update strategy from [56] is used in SPS-PSO as one CSGS. It is described as follows:

$$c = \frac{(D-1)\,N(0,1)}{D} + \frac{C(0,1)}{D} \tag{4}$$

$$v_{i,d}^{t+1} = \left(mean_{i,d}^{t} - x_{id}^{t}\right) + \frac{c}{\sqrt{3}}\sqrt{\left(p_{i,d} - mean_{i,d}^{t}\right)^2 + \left(x_{id}^{t} - mean_{i,d}^{t}\right)^2 + \left(x_{a,d}^{t} - mean_{i,d}^{t}\right)^2}, \tag{5}$$

where $N(0,1)$ and $C(0,1)$ represent two numbers randomly generated by the Gaussian distribution and Cauchy distribution, respectively. $mean_{i,d}^{t}$ is set to be the same as in Ref. [57]. The position update method and other variables have the same meaning as described previously.

4) The velocity update strategy from [55] is used in SPS-PSO as one CSGS. It is described as follows:

$$v_{i,d}^{t+1} = w * v_{i,d}^{t} + c_1 * r_1 * \left(pbest_{f_i(d)} - x_{i,d}^{t} \right), \tag{6}$$

where $f_i = [f_i(1), f_i(2), ..., f_i(d)]$ defines which *pbest* should be used by the current particle. $pbest_{f_i(d)}$ can be the corresponding dimensionality of any particle's *pbest* including its own *pbest*. The position update method and other variables have the same meaning as described previously.

5) The strategy from [56] is used in SPS-PSO as one CSGS. It is described as follows:

$$v_{i,d}^{t+1} = w * v_{i,d}^{t} + 0.5 * c_1 * r_1 * \left(pbest_{f_i(d)} - x_{i,d}^{t} + p_{g,d} - x_{i,d}^{t} \right), \tag{7}$$

where the position update method and other variables have the same meaning as described previously.

2.4. Strategy Self-adaptive Mechanism

In the strategy self-adaptive mechanism, each CSGS is assigned a selection probability. The selection probability of each CSGS is increased or decreased based on its performance in the evolution process. Two main problems should be considered here: (1) which CSGS should be selected, and (2) how to update the selection probabilities according to the performance of the CSGS?

An update cycle is set for all CSGSs, and in each cycle, the selection probability for each CSGS is fixed. If a CSGS successfully improves a solution, i.e., the new solution generated by this CSGS is better than the old one, the strategy selection probability of this CSGS will be increased; otherwise, it will be decreased. At the beginning of the algorithm, all CSGSs are assigned the same initial selection probability, which is set to $1/N_q$, where N_q is the number of CSGSs in the strategy pool. The sum of all probabilities of CSGSs is 1. P_q is then used to represent the selection probability of the q^{th} $(q = 1, 2, 3, ..., N_q)$ strategy.

The roulette wheel method [58] is used to select a CSGS based on the selection probabilities. The selected CSGS is then used by the current particle to generate a new solution. The newly generated solution is then evaluated by a classifier, and the update mechanism mentioned in Section 2.1 is utilized to decide whether to update *pbest* and *gbest*. Subsequently, this information (whether the new solution is better than the original one) is recorded by the elements $nsFlag_{i,q}$ and $nfFlag_{i,q}$ $(i = 1, 2, ..., N_{ps}, q = 1, 2, ..., N_q)$, as shown below:

$$nsFlag = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{N_{ps} \times N_q} \quad and \quad nfFlag = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{N_{ps} \times N_q} \tag{8}$$

where N_{ps} is the number of particles, and N_q is the number of CSGSs.

For example, at some point, the q^{th} strategy is selected for the i^{th} particle. If the new solution is better than the old one, then $nsFlag_{i,q}$ will be set to

1; otherwise, $nfFlag_{i,q}$ will be set to 1. If the evolution of the current generation is completed, the sum of each column from $nsFlag_{i,q}$ and $nfFlag_{i,q}$ is calculated, respectively. The results are then recorded as two elements, $S_{k,q}$ and $F_{k,q}\,(k = 1, 2, ..., N_g, q = 1, 2, ..., N_q)$, as shown in Formula (9), where N_g represents the number of generations in a cycle, indicating that the selection probability will be updated once N_g generations are completed, and $S_{k,q}$ means the number of successful evolutions that the q^{th} strategy has in the k^{th} generation. Meanwhile, the matrices $nsFlag_{N_{ps}\times N_q}$ and $nfFlag_{N_{ps}\times N_q}$ are reset according to Formula (8) after the selection probabilities are updated, so that new information can be recorded in the next cycle.

$$S = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{N_g \times N_q} \quad and \quad F = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{N_g \times N_q} \tag{9}$$

After the process is repeated for N_g generations, the total number of successes and failures of all CSGSs during the N_g generations is calculated, and the selection probability of each CSGS is updated. First, sum each column from S and F in Formula (9) is calculated to get the total number of successful instances or failed instances of each CSGS during the whole N_g generations, and the results are recorded as S_q^1. If the total number of successful evolution of a CSGS is 0, i.e., the value of S_q^1 is 0, a small enough value ε is assigned to S_q^1. The small value ε=0.01 is used to avoid being divided by zero.

$$S_q^1 = \sum_{k=1}^{N_g} S_{k,q} \tag{10}$$

$$S_q^2 = \begin{cases} \varepsilon, & if \;\; S_q^1 = 0 \\ S_q^1, & otherwise \end{cases} \tag{11}$$

Then, S_q^2 is divided by their total number of times that a strategy was selected, as shown in Formula (12). This step is designed to obtain the probability that a given CSGS succeeds in N_g generations of evolution. This probability will determine the likelihood of the CSGS being selected in the next cycle. At the same time, to guarantee that the summed probabilities of all CSGS is 1, S_q^3 is normalized; the calculation process is shown in Formula (13). Finally, p_q refers to the probability of strategy q being selected.

$$S_q^3 = S_q^2 \Bigg/ \left(S_q^2 + \sum_{k=1}^{N_g} F_{k,q} \right) \tag{12}$$

$$p_q = S_q^3 \Bigg/ \sum_{q=1}^{N_q} S_q^3 \tag{13}$$

Algorithm 1: The Self-adaptive Mechanism for SPS-PSO

Input: the number of fitness evaluations (NFE), current number of fitness evaluations (cFE), population size (N_{ps}), the number of strategies (N_q), $p_q = \frac{1}{N_q}\,(q = 1, 2, 3, ..., N_q)$, $N_g = 10$, $cur\_iter = 0$, $flag\_iter = 0$;

Output: *gbest*

```
while cFE < NFE do
    i = 0;
    for i < N_ps do
        Select a CSGS from the strategy pool based on the selection
        probabilities using the roulette wheel method to get the new
        particle x_i^new and its fitness;
        if x_i^new is better than x_i then
            nsFlag_{i,q} = 1;
            if x_i^new is better than pbest then
                Replace pbest with x_i^new;
                if x_i^new is better than gbest then
                    Replace gbest with x_i^new;
                end
            end
        end
        else
            nfFlag_{i,q} = 1;
        end
        cFE = cFE + 1;
        Replace x_i with x_i^new;
        i = i + 1;
    end
    cur_iter = cur_iter + 1;
    Calculate the sum of each column in nsFlag_{i,q} and record it in
    S_{N_g × N_q};
    Calculate the sum of each column in nfFlag_{i,q} and record it in
    F_{N_g × N_q};
    Reset nsFlag_{i,q} and nfFlag_{i,q} to zero matrices;
    if cur_iter − flag_iter = N_g then
        Use S_{N_g × N_q} and F_{N_g × N_q} to update p_q (q = 1, 2, 3, ..., N_q);
        Reset S_{N_g × N_q} and F_{N_g × N_q} to zero matrices;
        flag_iter=cur_iter;
    end
end
```

2.5. Parameter Self-adaptive Mechanism

Parameter adaptation has been used in many EC algorithms. Qin *et al.* [59] proposed a self-adaptive differential evolution (SaDE) algorithm, in which the trial vector generation strategies and their related control parameter values are gradually adapted to a given problem. In addition, Fan *et al.* [60] proposed a self-adaptive differential evolution algorithm with zoning evolution of control parameters and adaptive mutation strategies (ZEPDE). The overall performance of ZEPDE is better than those of other similar algorithms for solving optimization problems, especially when the test function has a higher dimensionality. In this research, in order to improve PSO for feature selection problems, the parameter adaptation mechanism is introduced and further developed into the proposed algorithm.

2.5.1. Parameter Initialization

In the self-adaptive process for parameters, a parameter starts from an initial value, and after undergoing the process, finally reaches its optimal value, which can make the corresponding CSGS perform well in the optimization process. In [59], a parameter self-adaptive mechanism is employed in a differential evolution (DE) algorithm, and all the parameters are given the same initial value. However, in SPS-PSO, it is not suitable for all parameters to have the same initial value. Because the appropriate range of each parameter may vary considerably, evolution will take more time if the initial value of a parameter is far from its optimal values. At the same time, because of the increase in gap between the initial value and the optimal value, the probability that the parameter falls into local optima also greatly increases. In this case, the parameters have a greater probability of falling into the local optima instead of reaching their global optimal value(s).

To improve the performance of CSGSs through adjusting their parameters in SPS-PSO, it is necessary to make the parameters reach their optimal values more quickly while reducing their probability of falling into local optima. Although the specific optimal parameter value is not determined, each parameter can be given a good enough value that will serve as the initial value. Therefore, the initial value for each parameter is set individually. More specifically, the initial value of each parameter is set to a value that is proven to result in a good performance.

2.5.2. Parameter Update Procedure

The traditional PSO algorithm uses fixed parameter values. A good set of parameters can improve the performance of the algorithm [59]. For a specific problem, the parameters can reach the optimal values that can make the CSGSs perform well. In SPS-PSO, the Gaussian transformation is used to adjust the parameter values to better values.

In SPS-PSO, to make the parameters more quickly approach the suitable values that would cause the corresponding CSGSs to generate better solutions (i.e., with higher classification accuracy and smaller solution sizes), values from

existing literature were assigned to be the initial values for the parameters. $PVM_{N_p} = \begin{bmatrix} P_1 & P_2 & \cdots & P_{N_p} \end{bmatrix}$, where N_p is the total number of parameters of all the CSGSs, and P_n is the n^{th} parameter in PVM_{N_p}. At the beginning of a generation, all parameter values are set using Formula (14).

$$PVMemory = \begin{bmatrix} P_1 & P_2 & \cdots & P_{N_p} \\ P_1 & P_2 & \cdots & P_{N_p} \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}_{N_{ps} \times N_p} \tag{14}$$

If a strategy is chosen to generate a particle, its parameter values will be selected from the corresponding elements in $PVMemory_{N_{ps} \times N_p}$. For example, if the i^{th} particle selects CSGS 1 to generate a new particle, then the needed parameters are taken out from the i^{th} row and corresponding columns in $PVMemory_{N_{ps} \times N_p}$. The new parameters values are then generated from the old ones through Gaussian transformation, in which the initial value is used as the mean value while the standard deviation is 0.3. These newly generated parameter values will be used to generate a new particle. If the new particle performs better than the old one, these new parameter values will replace the corresponding old values in $PVMemory_{N_{ps} \times N_p}$. After a generation of evolution, the mean values of all columns of $PVMemory_{N_{ps} \times N_p}$ are calculated and then recorded in $TempPVM_{N_g \times N_p}$.

$$TempPVM = \begin{bmatrix} TP_1 & TP_2 & \cdots & TP_{N_p} \\ TP_1 & TP_2 & \cdots & TP_{N_p} \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}_{N_g \times N_p} \tag{15}$$

N_g is set as an evolutionary cycle. After every N_g generations, the new parameter values that are used in the next cycle will be generated by counting the values in all $TempPVM_{N_g \times N_p}$. The counting method is to get the average value of each generation for every parameter in $TempPVM_{N_g \times N_p}$, i.e., the mean value of every column in $TempPVM_{N_g \times N_p}$ is calculated. These new parameter values will replace PVM_{N_p} and will be used in the next evolution cycle. In the evolution process, each of the related parameter values are gradually adapted by learning the valid values that they generated before. In this process, the algorithm can adaptively determine more suitable parameter values according to different stages of the search process.

3. Experiments

In this section we present the experimental methods. Nine datasets with more than 600 dimensions were used to perform the experiments. To investigate whether the evolutionary computation based feature selection method is effective and to test the performance of SPS-SPO for feature selection, two groups of experiments were conducted. In the two groups of experiments, four classifiers and four other evolutionary algorithms were employed. The detailed descriptions about the datasets, experimental methods, and parameter settings are given as follows.

Algorithm 2: The Parameter Update Procedure

Input: The number of fitness evaluations (NFE), current number of fitness evaluations (cFE), population size (N_{ps}), $PVM_{N_p} = \begin{bmatrix} P_1 & P_2 & \cdots & P_{N_p} \end{bmatrix}$, $cur\_iter = 0$, $flag\_iter = 0$;

while $cFE < NFE$ **do**
  Set PVM_{N_p} to $PVMemory_{N_{ps} \times N_p}$;
  $i = 0$;
  for $i < N_{ps}$ **do**
    Get the parameters needed from $PVMemory_{N_{ps} \times N_p}$, and generate a new parameter value p_i by Gaussian transformation;
    Generate a new particle with the corresponding CSGS and p_i;
    if x_i^{new} *is better than* x_i **then**
      $PVMemory_{i,n} = p_i$;
    end
    $i = i + 1$;
    $cFE = cFE + 1$;
  end
  $cur\_iter = cur\_iter + 1$;
  Calculate the mean of each column in $PVMemory_{N_{ps} \times N_p}$ and record it in $TempPVM_{N_g \times N_p}$;
  Replace PVM_{N_p} with $TempPVM_{l,N_p}$ ($l = cur\_iter - flag\_iter$);
  if $cur\_iter - flag\_iter = N_g$ **then**
    Calculate the mean of each column in $TempPVMy_{N_{ps} \times N_p}$ and replace PVM_{N_p};
    Reset $TempPVMemory_{N_{ps} \times N_p}$ to zero matrix;
    $flag\_iter$=cur_iter;
  end
end

Table 1: The detailed Information of Datasets

ID	Datasets	NoE	NoF	NoC
DS1	isolet5	1559	617	26
DS2	MultipleFeaturesDigit	2000	649	10
DS3	CNAE	1080	856	9
DS4	reged01	500	999	2
DS5	tied	750	999	4
DS6	marti	500	1024	2
DS7	MicroMass	360	1300	2
DS8	gisette_valid	1000	5000	2
DS9	drivFaceD	606	6400	3

3.1. Datasets

The datasets used in the experiments are chosen from the University of California Irvine (UCI) Machine Learning Repository [51] and Causality Workbench [52], as shown in Table 1. Among them, DS 1, DS 2, DS 3, DS 6, DS 7, DS 8, and DS 9 are from UCI whereas DS 4 and DS 5 are from Causality Workbench. To assess the performance of SPS-PSO on large-scale feature selection problems, the dimensions of all datasets are greater than 600. A brief description of all datasets is shown in Table 1.

In Table 1, "DSn" represents the n^{th} dataset, NoE means the number of examples, NoF represents the number of features (dimensions), and NoC is the number of classes. Furthermore, all the datasets were divided into training sets and test sets. Among them, the training sets account for 70%, while the remaining 30% is used as test sets. For the classifiers, 3-fold cross-validation is used to obtain the classification accuracy.

3.2. Algorithms and Classifiers for Comparison

Two groups of experiments were performed: (1) Different classifiers may have different levels of performance when being used in feature selection problems. When different classifiers are used as evaluation functions, the accuracy and sizes of the solutions obtained by SPS-PSO may also be different, too. To prove that this proposed method is effective for more classifiers, four different classifiers, namely KNN [46], LDA [47], SVM [49], and ELM [50], were used as evaluation functions for SPS-PSO, respectively. To investigate the performance of different EC algorithms, a relatively suitable classifier was used as the final evaluation function. (2) To further show the superiority of the SPS-PSO algorithm compared with the other four EC techniques, four EC algorithms, namely PSO [41], GA [61], DE [62], and SaDE [59], were used as comparison algorithms for the same classifier, which was KNN.

3.3. Parameter Settings

Detailed information on the parameter settings of SPS-PSO is given in Table 2, where θ is a threshold used to determine whether the feature should be

Table 2: Parameter Values of SPS-PSO

Operating parameters				Initial values of parameters				
θ	NFE	N_g	N_{ps}	PV_1	PV_2	PV_3	PV_4	PV_5
0.6	100,000	10	100	1.49445	1.49445	1.49618	1.49618	0.7298

selected. NFE represents the maximum number of fitness evaluations. N_g is the number of the leaning periods, which is the number of generations between two update cycles. N_{ps} represents the number of the particles, and PV_n shows the initial parameter values for CSGSs. At the same time, the parameter values of the compared algorithms are set to be the same as those used in their original references. The parameter values of the four different classifiers are set as follows: KNN: number of neighbors K=3; LDA: type is diagLinear; ELM: the number of hidden layer nodes is 100, and the kernel function is sigmoid function; SVM: the kernel function is RBF, the loss function C=1, and the gamma function G=0.07.

To compare the performance of different algorithms as fairly as possible, NFE was used as stop criteria, and the same $NFE = 100,000$ was used for each comparison algorithm. The qualities of the final solutions obtained by the different algorithms were compared. Meanwhile, to obtain more valuable and statistical conclusions, each experiment was repeated 26 times.

4. Results and Analysis

The experimental results are shown in Tables 3∼8. In these tables, "mean" represents the average value of the accuracy or the size of the final solution obtained by each algorithm, and "std" represents the standard deviation. In Table 5 and Table 8, the %-column lists the proportion of features that is reduced by an algorithm from the complete feature sets. A statistical significance test (T-test) with a confidence level of 95% is employed [63]. Therefore, the symbol "+" means that the benchmark method is better than the corresponding method with a significant difference, "-" means that the benchmark method is worse than the corresponding method with a significant difference, and "=" means that there is no significant difference between the benchmark method and the corresponding method. The symbols are followed by the p values from the statistical significance test. Taking Table 4 as an example, "+" indicates that KNN is better than ELM with a significant difference, and "=" indicates that there is no significant difference between KNN and ELM.

4.1. Comparison of the Four Classifiers

Table 3 and Table 4 show the classification accuracy of SPS-PSO on the training sets and test sets with the four types of classifiers: KNN, LDA, SVM, and ELM. Table 5 shows the solution sizes (the number of finally selected features) obtained by SPS-PSO on the training sets when the four classifiers are used as evaluation functions, respectively.

Table 3: CLASSIFICATION ACCURACY OF THE FOUR CLASSIFIERS USING SPS-PSO ON TRAINING SETS

Datasets	KNN		LDA		SVM		ELM	
	mean	std	mean	std	mean	std	mean	std
DS 1	9.06E-01	7.30E-03	**9.41E-01**	2.70E-03	9.36E-01	4.10E-03	8.79E-01	1.12E-02
			-,<0.001		-,<0.001		+,<0.001	
DS 2	9.79E-01	1.10E-03	**9.93E-01**	1.00E-03	1.17E-01	0	8.04E-01	4.70E-03
			-,<0.001		+,<0.001		+,<0.001	
DS 3	9.30E-01	5.20E-03	**9.63E-01**	2.50E-03	8.04E-01	4.10E-03	9.42E-01	4.60E-03
			-,<0.001		+,<0.001		-,<0.001	
DS 4	9.94E-01	2.00E-03	**9.97E-01**	0	8.97E-01	0	9.26E-01	2.80E-03
			-,<0.001		+,<0.001		+,<0.001	
DS 5	9.94E-01	5.10E-03	9.99E-01	9.00E-04	**1.00E+00**	0	9.51E-01	2.36E-02
			-,<0.001		-,<0.001		+,<0.001	
DS 6	**8.95E-01**	2.40E-03	6.33E-01	5.50E-03	8.89E-01	0	8.88E-01	3.50E-03
			+,<0.001		+,<0.001		+,<0.001	
DS 7	9.85E-01	5.30E-03	**1.00E+00**	8.00E-04	5.93E-01	5.17E-02	9.28E-01	1.14E-02
			-,<0.001		+,<0.001		+,<0.001	
DS 8	9.73E-01	3.50E-03	**9.83E-01**	5.30E-03	5.09E-01	0	8.51E-01	5.40E-03
			-,<0.001		+,<0.001		+,<0.001	
DS 9	**9.85E-01**	1.40E-03	8.86E-01	4.20E-03	9.10E-01	0	9.54E-01	2.30E-03
			+,<0.001		+,<0.001		+,<0.001	

Table 4: CLASSIFICATION ACCURACY OF THE FOUR CLASSIFIERS USING SPS-PSO ON TEST SETS

Datasets	KNN		LDA		SVM		ELM	
	mean	std	mean	std	mean	std	mean	std
DS 1	8.25E-01	1.32E-02	**8.76E-01**	1.21E-02	7.21E-01	2.22E-02	7.50E-01	1.66E-02
			-,<0.001		+,<0.001		+,<0.001	
DS 2	9.48E-01	5.90E-03	**9.78E-01**	3.60E-03	1.21E-01	1.60E-03	9.36E-01	6.10E-03
			-,<0.001		+,<0.001		+,<0.001	
DS 3	8.13E-01	2.88E-02	**8.85E-01**	1.86E-02	1.92E-01	2.16E-02	7.25E-01	4.15E-02
			-,<0.001		+,<0.001		+,<0.001	
DS 4	8.94E-01	2.02E-02	**9.45E-01**	1.70E-02	8.47E-01	0	8.60E-01	1.94E-02
			-,<0.001		+,<0.001		+,<0.001	
DS 5	9.36E-01	1.58E-02	**9.74E-01**	7.70E-03	9.21E-01	1.56E-02	8.79E-01	2.03E-02
			-,<0.001		+,<0.001		+,<0.001	
DS 6	8.48E-01	1.04E-02	5.54E-01	4.32E-02	**8.67E-01**	0	8.50E-01	9.90E-03
			+,<0.001		-,<0.001		=,0.45	
DS 7	7.89E-01	4.07E-02	**8.39E-01**	3.15E-02	5.29E-01	3.23E-02	7.04E-01	3.77E-02
			-,<0.001		+,<0.001		+,<0.001	
DS 8	8.96E-01	1.36E-02	**9.22E-01**	1.18E-02	5.20E-01	0	8.74E-01	1.54E-02
			-,<0.001		+,<0.001		+,<0.001	
DS 9	**9.17E-01**	6.90E-03	7.92E-01	4.84E-02	8.79E-01	0	9.14E-01	7.60E-03
			+,<0.001		+,<0.001		=,0.11	

It can be seen from Table 3 that KNN is better than SVM on 7 training sets with significant difference, and that KNN is better than ELM on 8 training sets with a significant difference. Similarly, it can be seen from Table 4 that KNN outperforms SVM on 8 test sets with a significant difference, whereas KNN outperforms ELM on 7 test sets with a significant difference. Table 4 also shows that KNN outperforms SVM and ELM in terms of classification accuracy on most training sets and test sets. From Table 3 and Table 4 we can see that KNN outperforms LDA on only 2 training sets and 2 test sets, respectively, i.e., the classification accuracy of LDA is higher than that of KNN on most training sets and test sets. It would therefore seems that LDA is better. However, evaluation in terms of classification performance is only one of the two objectives of this study. Thus, it is necessary to further analyze the sizes of the final solutions.

From Table 5 we see that feature selection can reduce most of the features for all the classifiers. For example, it can reduce 70%-80% features for KNN.

Table 5: The SOLUTION SIZEs OF THE FOUR CLASSIFIERS USING SPS-PSO ON TRAINING SETS

Datasets	KNN			LDA			SVM			ELM		
	mean	std	%	mean	std	%	mean	std	%	mean	std	%
DS 1	1.56E+02	1.10E+01	74.73	1.55E+02 =,0.72	1.07E+01	74.86	1.46E+02 -,<0.001	9.81E+00	76.29	**1.04E+02** -,<0.001	1.65E+01	83.13
DS 2	1.71E+02	2.16E+01	73.66	1.59E+02 -,0.008	9.47E+00	75.52	**1.71E+02** =,1	2.52E+01	73.59	2.75E+02 +,<0.001	4.06E+01	57.62
DS 3	2.54E+02	3.02E+01	70.28	3.03E+02 +,<0.001	3.21E+01	64.65	**1.33E+02** -,<0.001	8.22E+00	84.48	1.95E+02 -,<0.001	1.92E+01	77.25
DS 4	1.96E+02	3.85E+01	80.41	1.13E+02 -,<0.001	2.17E+01	88.72	**1.45E+01** -,<0.001	5.76E+00	98.55	2.89E+02 +,<0.001	4.69E+01	71.05
DS 5	1.89E+02	2.57E+01	81.13	2.33E+02 +,<0.001	4.56E+01	76.66	**1.13E+02** -,<0.001	5.82E+00	88.70	1.36E+02 -,<0.001	4.20E+01	86.40
DS 6	3.04E+02	3.81E+01	70.35	3.47E+02 +,<0.001	4.54E+01	66.15	**1.53E+01** -,<0.001	5.14E+00	98.50	3.46E+02 +,0.002	5.96E+01	66.23
DS 7	3.21E+02	2.87E+01	75.31	2.43E+02 -,<0.001	3.41E+01	81.28	**7.11E+01** -,<0.001	1.75E+01	94.53	3.01E+02 =,0.033	4.10E+01	76.83
DS 8	1.26E+03	1.14E+02	74.87	1.17E+03 -,<0.001	7.00E+01	76.58	**2.49E+02** -,<0.001	4.13E+01	95.01	1.29E+03 =,0.41	1.63E+02	74.14
DS 9	1.88E+03	2.51E+02	70.67	1.84E+03 =,0.53	2.43E+02	71.33	**3.25E+02** -,<0.001	5.07E+01	94.92	1.71E+03 -,0.004	1.83E+02	73.33

This indicates that feature selection is an effective method for simplifying the classification systems. It can be seen from Table 5 that SVM achieves the smallest solution sizes on eight datasets. However, the classification accuracy of SVM on the training and test sets are too low on DS 2, DS 7, and DS 8. The accuracy of SVM on training sets DS 2, is only 11.7%, in particular. This results indicates that SVM is unstable. It seems that KNN, LDA, and ELM have advantages in terms of the solution size. It can be seen from Table 5 that, LDA only reduced 64% and 66% of all features on DS 3 and DS 6, respectively, and ELM only reduced 57% and 66% of all features on DS 2 and DS 6, respectively. By contrast, KNN can consistently reduce more than 70% of all features on all training sets. Besides, the standard deviation of KNN is slightly smaller than those of LDA and ELM. Therefore, although KNN is not the best classifier on all datasets, its performance is the most stable. The experimental results reveal that SPS-PSO can achieve good performance for feature selection when different classifiers are used as evaluation functions, respectively. In addition, it can be concluded that feature selection is an effective approach for improving the performance of classification systems. Nevertheless, some different classification results are different when different classifiers are employed. In the four classifiers, KNN's performance is more stable, although the solution size obtained is not as good as SVM and the classification accuracy is not as good as that of LDA, KNN performs the most stably among the four classifiers, i.e., KNN can consistently reduce more than 70% features while ensuring a good classification accuracy. Thus, KNN is chosen as the final classifier for comparing SPS-PSO with the other EC algorithms.

4.2. Comparison of SPS-PSO with Other EC Algorithms

Table 6 and Table 7 show classification accuracy of SPS-PSO and the other EC algorithms on training sets and test sets, respectively. Meanwhile, Table 8 shows the final solution sizes on training sets obtained by SPS-PSO and the other algorithms.

It can be observed from Table 6 that the classification accuracy of SPS-PSO is higher than that of PSO on 8 training sets with a significant difference. Similarly, SPS-PSO is better than GA, DE, and SaDE on 7, 9, and 9 datasets, respectively. On DS 6 and DS 9, GA has a higher classification accuracy than that of SPS-PSO. However, the difference between them is so small that it can even be neglected. Besides, considering the significant difference, SPS-PSO is the same as that of GA on DS 6. All in all, the performance of SPS-PSO in terms of classification accuracy on the training sets is better than those of the other EC algorithms. It can be seen from Table 7 that SPS-PSO is obviously better than the other EC algorithms with a significant difference on the test sets. It can be observed from Table 8 that the solutions obtained by SPS-PSO are much smaller than those obtained by the other EC algorithms on all datasets with a significant difference. On these nine datasets, SPS-PSO removed over 70% of the features. Particularly, SPS-PSO can remove over 80% of features on DS 3 and DS 4. To sum up, SPS-PSO can remove a significant number of redundant and irrelevant features. According to Tables 6∼8, it can be observed that SPS-PSO can not only reduce the sizes of the solutions but also obtain higher classification accuracy compared with other algorithms. Therefore, the strategy and parameter self-adaptive mechanisms are very effective techniques for improving the performance of evolutionary algorithms on large-scale feature selection problems.

4.3. Convergence Performance of the EC Methods on the Training Sets

Fig.1 and Fig.2 illustrate the convergence performance of SPS-PSO and the other EC algorithms on the nine training sets in terms of classification accuracy and solution sizes (it is noted that the scales along Y axis are different for different graphs).To produce graphs that can be read more clearly, the convergence performance curves on the nine datasets are provided in Fig.1 and Fig.2 separately.

Table 6: CLASSIFICATION ACCURACY OF METHODS ON TRAINING SETS(KNN)

Data sets	PSO		GA		DE		SaDE		SPS-PSO	
	mean	std	mean	std	mean	std	mean	std	mean	std
DS 1	8.52E-01 +,<0.001	8.00E-03	8.32E-01 +,<0.001	6.70E-03	8.04E-01 +,<0.001	3.70E-03	8.11E-01 +,<0.001	4.20E-03	**9.06E-01**	7.30E-03
DS 2	9.75E-01 +,<0.001	1.30E-03	9.71E-01 +,<0.001	1.90E-03	9.67E-01 +,<0.001	1.10E-03	9.68E-01 +,<0.001	1.10E-03	**9.79E-01**	1.10E-03
DS 3	8.71E-01 +,<0.001	1.71E-02	8.78E-01 +,<0.001	1.40E-02	8.19E-01 +,<0.001	9.40E-03	8.17E-01 +,<0.001	9.10E-03	**9.30E-0**1	5.20E-03
DS 4	9.71E-01 +,<0.001	7.20E-03	9.49E-01 +,<0.001	8.90E-03	9.39E-01 +,<0.001	2.20E-03	9.44E-01 +,<0.001	2.80E-03	**9.94E-01**	2.00E-03
DS 5	9.53E-01 +,<0.001	1.14E-02	9.39E-01 +,<0.001	8.00E-03	9.07E-01 +,<0.001	3.70E-03	9.16E-01 +,<0.001	3.00E-03	**9.94E-01**	5.10E-03
DS 6	8.95E-01 =,1	1.40E-03	**8.95E-01** =,1	3.20E-03	8.92E-01 +,<0.001	1.40E-03	8.92E-01 +,<0.001	2.10E-03	8.95E-01	2.40E-03
DS 7	9.72E-01 +,<0.001	1.40E-03	9.54E-01 +,<0.001	1.09E-02	9.33E-01 +,<0.001	4.80E-03	9.39E-01 +,<0.001	6.70E-03	**9.85E-01**	5.30E-03
DS 8	9.57E-01 +,<0.001	6.40E-03	9.54E-01 +,<0.001	3.00E-03	9.47E-01 +,<0.001	1.60E-03	9.49E-01 +,<0.001	1.70E-03	**9.73E-01**	3.50E-03
DS 9	9.84E-01 +,0.004	1.20E-03	**9.85E-01** =,1	1.80E-03	9.83E-01 +,<0.001	1.10E-03	9.83E-01 +,<0.001	1.10E-03	9.85E-01	1.40E-03

Table 7: CLASSIFICATION ACCURACY OF METHODS ON TEST SETS(KNN)

Data sets	PSO		GA		DE		SaDE		SPS-PSO	
	mean	std	mean	std	mean	std	mean	std	mean	std
DS 1	7.83E-01 +,<0.001	1.98E-02	7.59E-01 +,<0.001	2.03E-02	7.53E-01 +,<0.001	1.38E-02	7.50E-01 +,<0.001	1.66E-02	**8.25E-01**	1.32E-02
DS 2	9.44E-01 +,0.031	8.00E-03	9.38E-01 +,<0.001	6.00E-03	9.38E-01 +,<0.001	8.00E-03	9.36E-01 +,<0.001	6.10E-03	**9.48E-01**	5.90E-03
DS 3	7.60E-01 +,<0.001	4.74E-02	7.85E-01 +,<0.001	2.86E-02	7.43E-01 +,<0.001	3.27E-02	7.25E-01 +,<0.001	4.15E-02	**8.13E-01**	2.88E-02
DS 4	8.66E-01 +,<0.001	1.92E-02	8.58E-01 +,<0.001	1.26E-02	8.64E-01 +,<0.001	2.02E-02	8.60E-01 +,<0.001	1.94E-02	**8.94E-01**	2.02E-02
DS 5	8.86E-01 +,<0.001	1.51E-02	8.90E-01 +,<0.001	1.54E-02	8.74E-01 +,<0.001	1.31E-02	8.79E-01 +,<0.001	2.03E-02	**9.36E-01**	1.58E-02
DS 6	**8.57E-01** -,0.002	1.15E-02	8.47E-01 =,0.7	9.50E-03	8.48E-01 =,1	1.16E-02	8.50E-01 =,0.45	9.90E-03	8.48E-01	1.04E-02
DS 7	6.88E-01 +,<0.001	4.86E-02	6.93E-01 +,<0.001	3.78E-02	6.86E-01 +,<0.001	5.03E-02	7.04E-01 +,<0.001	3.77E-02	**7.89E-01**	4.07E-02
DS 8	8.83E-01 +,<0.001	9.80E-03	8.82E-01 +,<0.001	1.45E-02	8.79E-01 +,<0.001	1.44E-02	8.74E-01 +,<0.001	1.54E-02	**8.96E-01**	1.36E-02
DS 9	9.12E-01 +,0.032	1.04E-02	9.16E-01 =,0.65	1.00E-02	**9.18E-01** =,0.65	1.00E-02	9.14E-01 =,0.11	7.60E-03	9.17E-01	6.90E-03

Table 8: SOLUTION SIZE OF METHODS ON TRAINING SETS (KNN)

Datasets	PSO			GA			DE			SaDE			SPS-PSO		
	mean	std	%	mean	std	%	mean	std	%	mean	std	%	mean	std	%
DS 1	1.72E+02 +,<0.001	1.87E+01	72.20	3.48E+02 +,<0.001	3.34E+01	43.67	2.51E+02 +,<0.001	1.41E+01	59.32	2.33E+02 +,<0.001	1.36E+01	62.27	**1.56E+02**	1.10E+01	74.73
DS 2	1.98E+02 +,<0.001	1.80E+01	69.45	3.38E+02 +,<0.001	5.41E+01	47.86	2.64E+02 +,<0.001	1.25E+01	59.34	2.56E+02 +,<0.001	1.22E+01	60.54	**1.71E+02**	2.16E+01	73.66
DS 3	3.91E+02 +,<0.001	6.21E+01	54.35	6.79E+02 +,<0.001	3.73E+01	20.64	6.70E+02 +,<0.001	9.74E+01	21.71	4.76E+02 +,<0.001	1.73E+02	44.41	**2.54E+02**	3.02E+01	70.28
DS 4	2.91E+02 +,<0.001	3.94E+01	70.90	4.84E+02 +,<0.001	8.46E+01	51.59	4.01E+02 +,<0.001	1.46E+01	59.82	3.79E+02 +,<0.001	1.99E+01	62.10	**1.96E+02**	3.85E+01	80.41
DS 5	3.31E+02 +,<0.001	6.95E+01	66.86	4.83E+02 +,<0.001	7.03E+01	51.64	4.03E+02 +,<0.001	1.66E+01	59.66	3.87E+02 +,<0.001	1.65E+01	61.27	**1.89E+02**	2.57E+01	81.13
DS 6	3.14E+02 =,0.37	4.78E+01	69.36	5.23E+02 +,<0.001	9.67E+01	48.88	4.13E+02 +,<0.001	5.21E+01	59.67	3.95E+02 +,<0.001	2.25E+01	61.40	**3.04E+02**	3.81E+01	70.35
DS 7	4.12E+02 =,0.08	2.73E+02	68.31	7.34E+02 +,<0.001	1.65E+02	43.57	5.21E+02 +,<0.001	1.89E+01	59.92	5.02E+02 +,<0.001	2.10E+01	61.37	**3.21E+02**	2.87E+01	75.31
DS 8	1.69E+03 +,<0.001	1.24E+02	66.27	2.71E+03 +,<0.001	4.50E+02	45.88	2.01E+03 +,<0.001	5.30E+01	59.80	1.95E+03 +,<0.001	5.23E+01	60.96	**1.26E+03**	1.14E+02	74.87
DS 9	1.91E+03 +,0.54	9.42E+01	70.11	3.23E+03 +,<0.001	5.30E+02	49.47	2.57E+03 +,<0.001	7.26E+01	59.82	2.48E+03 +,<0.001	6.59E+01	61.23	**1.88E+03**	2.51E+02	70.67

Through a comparison of the convergence curves of these algorithms, it is found that the performance levels of the four algorithms are relatively close at the beginning, because all the algorithms use the same initialization method. At the early stages of the algorithms, SPS-PSO converges significantly faster than PSO, GA, DE, and SaDE. At the later stages, the convergence of the five algorithms is significantly reduced. Through observations on the convergence curves in terms of the classification accuracy and solution sizes, SPS-PSO outperforms the other four algorithms on all the datasets except DS 6 and DS 9. On DS 6 and DS 9, although the classification performance levels of SPS-PSO and GA are similar, the solution size curves of GA converged very early and quickly stopped. Similarly, although the convergence curve of PSO is relatively close to that of SPS-PSO, it does not perform as well as SPS-PSO in terms of the final classification accuracy and solution sizes. Feature selection can be seen as an optimization problem that requires both high classification accuracy and small solution size. When compared with SPS-PSO, the other four EC

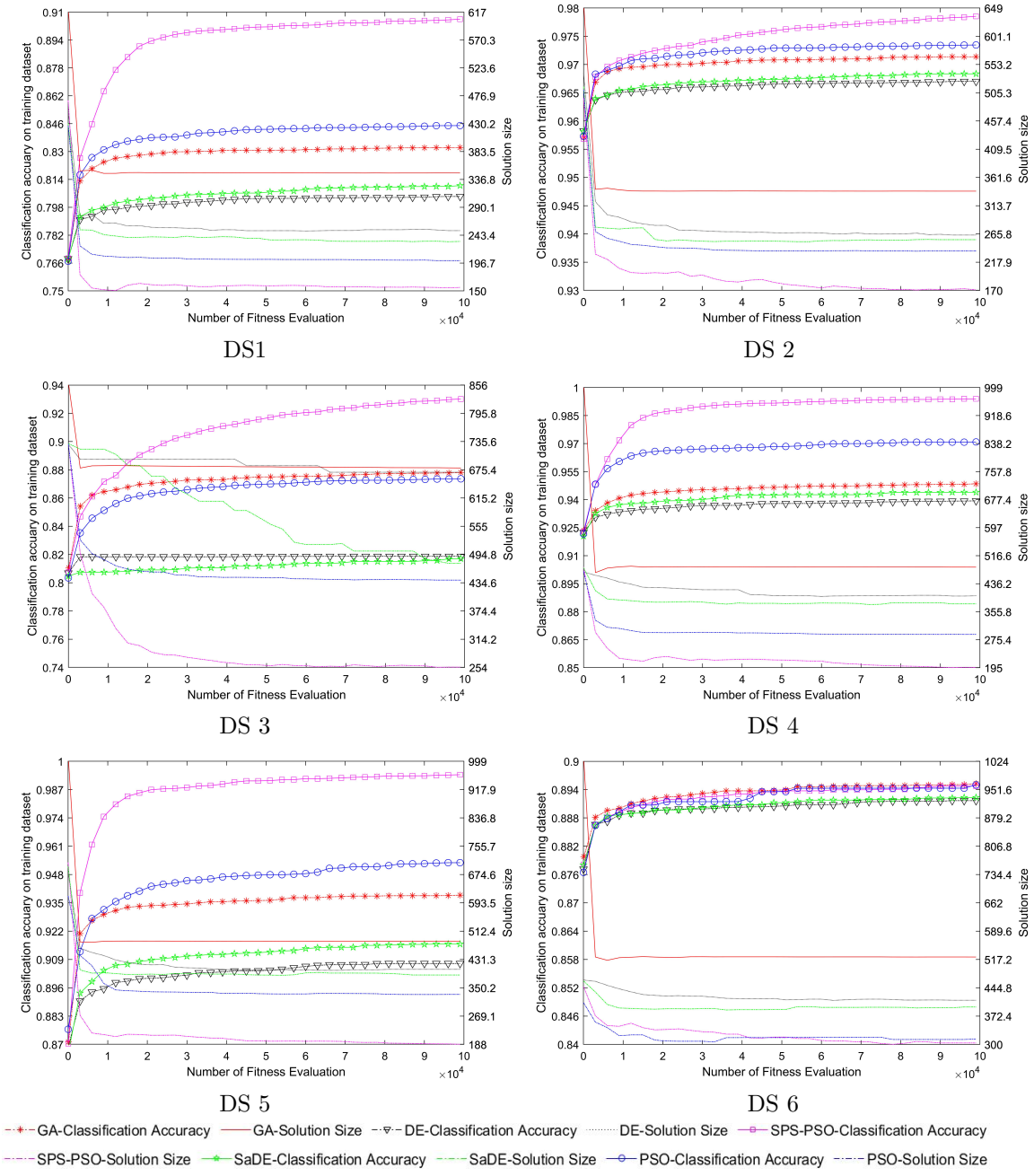

Figure 1: The classification accuracy and solution sizes of different algorithms on training sets (DS 1- DS 6).

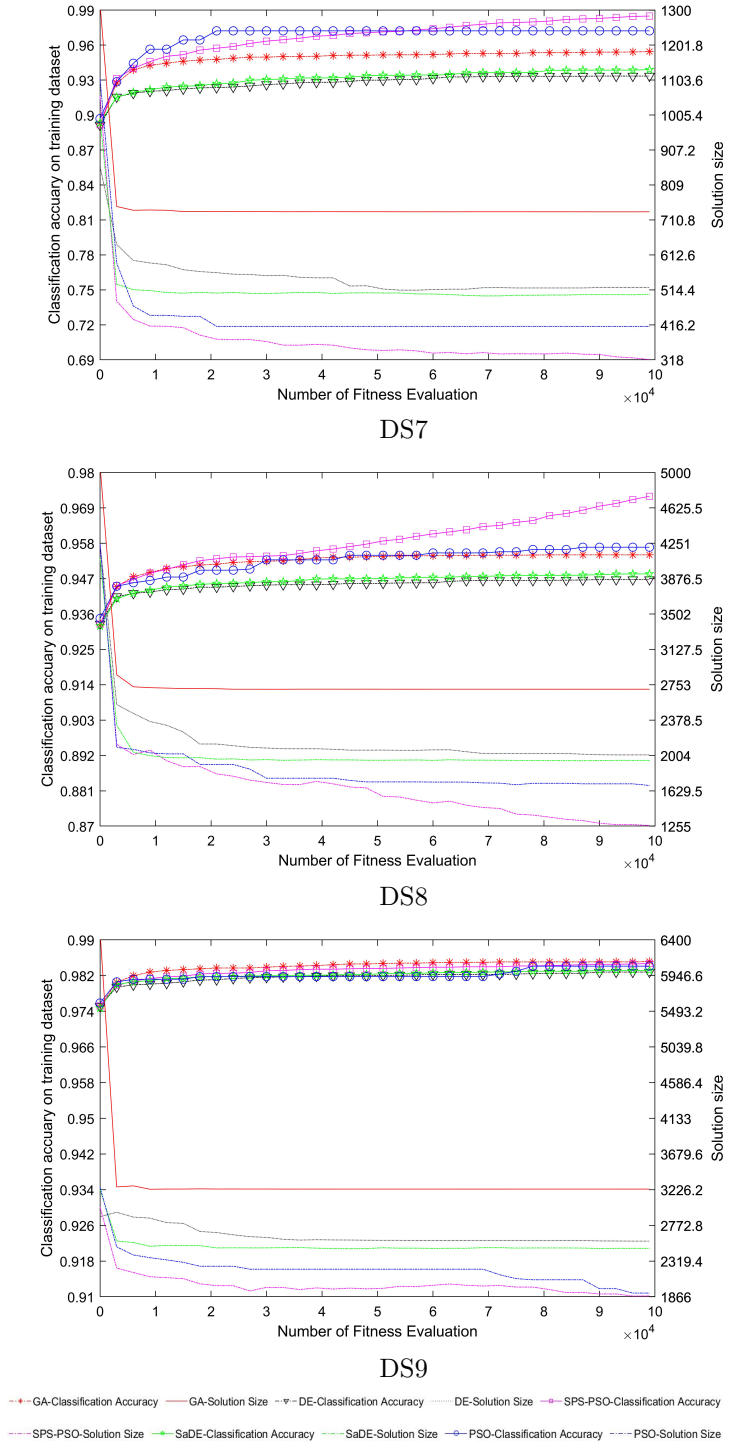

Figure 2: The classification accuracy and solution sizes of different algorithms on training sets (DS 7- DS 9).

algorithms cannot optimize the classification accuracy and the solution size as well for large-scale feature selection problems at the same time. The other EC algorithms fall into local optima more easily. In addition, if NFE is set to a small number, a solution with lower quality will be obtained. If NFE is set to a big number, a better solution, or even the best solution, can be obtained. Therefore, there is a trade-off between computation time and performance. This trade-off can be made by adjusting the value of NFE.

5. Conclusions and Future Work

In this research, in order to effectively solve large-scale feature selection problems, the strategy and parameter self-adaptive mechanisms were introduced into PSO, and a new algorithm named SPS-PSO was designed. Furthermore, in order to investigate the effect of feature selection for different classifiers, four classifiers, i.e., KNN, LDA, SVM, and ELM, are used as evaluation functions for SPS-PSO. Experimental results on nine datasets show that the feature selection technique is effective for improving the classification accuracy and reducing the calculation time when different classifiers are used as evaluation functions. It is also demonstrated that SPS-PSO can achieve higher classification accuracy and smaller solution sizes than those of the other four EC methods on large-scale feature selection problems. This result indicates that the adaptive mechanisms can significantly improve the performance of EC methods on feature selection problems. In future work, we will try to further improve the efficiency of SPS-PSO, and more classifiers will be used as evaluation functions for SPS-PSO to further assess its application potential.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (61403206, 61876089,61876185), the Natural Science Foundation of Jiangsu Province (BK20141005), the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (14KJB520025), the Engineering Research Center of Digital Forensics, Ministry of Education, and the Priority Academic Program Development of Jiangsu Higher Education Institutions.

References

[1] B. Xue, M. J. Zhang, W. N. Browne, X. Yao, A survey on evolutionary computation approaches to feature selection, IEEE Trans. Evolut. Comput. 20 (4) (2016) 606–626.

[2] S. Jadhav, H. He, K. Jenkins, Information gain directed genetic algorithm wrapper feature selection for credit rating, Appl. Soft Comput. 69 (2018) 541–553.

[3] I. Jain, V. K. Jain, R. Jain, Correlation feature selection based improved-binary particle swarm optimization for gene selection and cancer classification, Appl. Soft Comput. 62 (2018) 203–215.

[4] G. Jothi, H. H. Inbarani, Hybrid tolerance rough set-firefly based supervised feature selection for mri brain tumor image classification, Appl. Soft Comput. 46 (2016) 639–651.

[5] M. A. Ambusaidi, X. He, P. Nanda, Z. Tan, Building an intrusion detection system using a filter-based feature selection algorithm, IEEE Trans. Comput. 65 (10) (2016) 2986–2998.

[6] P. Mohapatra, S. Chakravarty, P. K. Dash, Microarray medical data classification using kernel ridge regression and modified cat swarm optimization based gene selection system, Swarm Evol. Comput. 28 (2016) 144–160.

[7] S. Nakariyakul, D. P. Casasent, Improved forward floating selection algorithm for feature subset selection, in: Int. Conf. Wav. Anal. and Pattern Recog., Vol. 2, 2008, pp. 793–798.

[8] S. D. Strearns, On selecting features for pattern classifiers, in: Int. Conf. Pattern Recog., 1976, pp. 71–75.

[9] D. Ververidis, C. Kotropoulos, Fast sequential floating forward selection applied to emotional speech features estimated on des and susas data collections, in: 14th Eur. Sig. Proc. Conf., 2006, pp. 1–5.

[10] R. Leardi, L. Ngaard, Sequential application of backward interval partial least squares and genetic algorithms for the selection of relevant spectral regions, J. Chemometr. 18 (11) (2004) 486–497.

[11] C. Dudeja, Fuzzy-based modified particle swarm optimization algorithm for shortest path problems, Soft Comput. 23 (17) (2019) 1–11.

[12] Y. Zhang, D. Gong, Y. Hu, W. Zhang, Feature selection algorithm based on bare bones particle swarm optimization, Neurocomputing 148 (2015) 150–157.

[13] Y. Zhang, X. F. Song, D. W. Gong, A return-cost-based binary firefly algorithm for feature selection, Inform. Sci. 418 (2017) 561–574.

[14] M. Mafarja, I. Aljarah, A. A. Heidari, H. Faris, P. Fournier-Viger, X. Li, S. Mirjalili, Binary dragonfly optimization for feature selection using time-varying transfer functions, Knowl-Based Syst. 161 (2018) 185–204.

[15] M. Mafarja, I. Aljarah, A. A. Heidari, A. I. Hammouri, H. Faris, A. M. Al-Zoubi, S. Mirjalili, Evolutionary population dynamics and grasshopper optimization approaches for feature selection problems, Knowl-Based Syst. 145 (2018) 25–45.

[16] M. Mafarja, S. Mirjalili, Whale optimization approaches for wrapper feature selection, Appl. Soft Comput. 62 (2018) 441–453.

[17] H. Faris, M. M. Mafarja, A. A. Heidari, I. Aljarah, A.-Z. AlaM, S. Mirjalili, H. Fujita, An efficient binary salp swarm algorithm with crossover scheme for feature selection problems, Knowl-Based Syst. 154 (2018) 43–67.

[18] I. Aljarah, M. Mafarja, A. A. Heidari, H. Faris, Y. Zhang, S. Mirjalili, Asynchronous accelerating multi-leader salp chains for feature selection, Appl. Soft Comput. 71 (2018) 964–979.

[19] M. M. Mafarja, S. Mirjalili, Hybrid whale optimization algorithm with simulated annealing for feature selection, Neurocomputing 260 (2017) 302–312.

[20] A. K. Das, S. Sengupta, S. Bhattacharyya, A group incremental feature selection for classification using rough set theory based genetic algorithm, Appl. Soft Comput. 65 (2018) 400–411.

[21] H. Dong, T. Li, R. Ding, J. Sun, A novel hybrid genetic algorithm with granular information for feature selection and optimization, Appl. Soft Comput. 65 (2018) 33–46.

[22] E. Hancer, B. Xue, M. Zhang, D. Karaboga, B. Akay, Pareto front feature selection based on artificial bee colony optimization, Inform. Sci. 422 (2018) 462–479.

[23] S. Arslan, C. Ozturk, Multi hive artificial bee colony programming for high dimensional symbolic regression with feature selection, Appl. Soft Comput. 78 (2019) 515–527.

[24] Y. Zhang, D.-W. Gong, X.-Z. Gao, T. Tian, X.-Y. Sun, Binary differential evolution with self-learning for multi-objective feature selection, Inform. Sci. 507 (2020) 67–85.

[25] U. Singh, S. N. Singh, A new optimal feature selection scheme for classification of power quality disturbances based on ant colony framework, Appl. Soft Comput. 74 (2019) 216–225.

[26] B. O. Alijla, C. P. Lim, L.-P. Wong, A. T. Khader, M. A. Al-Betar, An ensemble of intelligent water drop algorithm for feature selection optimization problem, Appl. Soft Comput. 65 (2018) 531–541.

[27] K.-C. Lin, J. C. Hung, J.-t. Wei, Feature selection with modified lion's algorithms and support vector machine for high-dimensional data, Appl. Soft Comput. 68 (2018) 669–676.

[28] Q. Tu, X. Chen, X. Liu, Multi-strategy ensemble grey wolf optimizer and its application to feature selection, Appl. Soft Comput. 76 (2019) 16–30.

[29] B. Hu, Y. Dai, Y. Su, P. Moore, X. Zhang, C. Mao, J. Chen, L. Xu, Feature selection for optimized high-dimensional biomedical data using an improved shuffled frog leaping algorithm, IEEE ACM Trans. Comput. Bi. 15 (6) (2018) 1765–1773.

[30] F. Pourpanah, Y. Shi, C. P. Lim, Q. Hao, C. J. Tan, Feature selection based on brain storm optimization for data classification, Appl. Soft Comput. 80 (2019) 761–775.

[31] R. Sheikhpour, M. A. Sarram, R. Sheikhpour, Particle swarm optimization for bandwidth determination and feature selection of kernel density estimation based classifiers in diagnosis of breast cancer, Appl. Soft Comput. 40 (2016) 113–131.

[32] K. R. Harrison, A. P. Engelbrecht, B. M. Ombuki-Berman, Self-adaptive particle swarm optimization: a review and analysis of convergence, Swarm Intelligence 12 (3) (2018) 187–226.

[33] Y. Wang, B. Li, T. Weise, J. Wang, B. Yuan, Q. Tian, Self-adaptive learning based particle swarm optimization, Inform. Sci. 181 (20) (2011) 4515–4538.

[34] C. H. Li, S. X. Yang, T. T. Nguyen, A self-learning particle swarm optimizer for global optimization problems, IEEE Trans. Syst. Man Cyb. 42 (3) (2012) 627–646.

[35] T. Niknam, H. D. Mojarrad, H. Z. Meymand, Non-smooth economic dispatch computation by fuzzy and self adaptive particle swarm optimization, Appl. Soft Comput. 11 (2) (2011) 2805–2817.

[36] M. Chih, Self-adaptive check and repair operator-based particle swarm optimization for the multidimensional knapsack problem, Appl. Soft Comput. 26 (2015) 378–389.

[37] M. Karimi-Nasab, M. Modarres, S. M. Seyedhoseini, A self-adaptive PSO for joint lot sizing and job shop scheduling with compressible process times, Appl. Soft Comput. 27 (2015) 137–147.

[38] A. Banitalebi, M. I. A. Aziz, Z. A. Aziz, A self-adaptive binary differential evolution algorithm for large scale binary optimization problems, Inform. Sci. 367 (2016) 487–511.

[39] Y. Xue, J. M. Jiang, B. P. Zhao, T. H. Ma, A self-adaptive artificial bee colony algorithm based on global best for global optimization, Soft Comput. 22 (9) (2018) 2935–2952.

[40] Y. Xue, B. Xue, M. Zhang, Self-adaptive particle swarm optimization for large-scale feature selection in classification, ACM Trans. Knowl. Discov. D. 13 (5) (2019) 1–27.

[41] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Int. Conf. Neural Networks, Vol. 4, 1995, pp. 1942–1948.

[42] Y. Zhang, H. G. Li, Q. Wang, C. Peng, A filter-based bare-bone particle swarm optimization algorithm for unsupervised feature selection, Applied Intelligence 49 (8) (2019) 2889–2898.

[43] B. Tran, B. Xue, M. J. Zhang, Variable-length particle swarm optimization for feature selection on high-dimensional classification, IEEE Trans. Evolut. Comput. 23 (3) (2019) 473–487.

[44] C. Y. Qiu, Bare bones particle swarm optimization with adaptive chaotic jump for feature selection in classification, Int. J. of Comput. Int. Sys. 11 (1) (2018) 1–14.

[45] Y. H. Lu, M. H. Liang, Z. Y. Ye, L. C. Cao, Improved particle swarm optimization algorithm and its application in text feature selection, Appl. Soft Comput. 35 (2015) 629–636.

[46] L. E. Peterson, K-nearest neighbor, Scholarpedia 4 (2) (2009) 1883.

[47] M. Li, B. Yuan, 2D-LDA: A statistical linear discriminant analysis for image matrix, Pattern Recogn. Lett. 26 (5) (2005) 527–532.

[48] V. Franc, V. Hlavac, Multi-class support vector machine, in: Obj. Recog. Supp. User Inter. Serv. Rob., 2002, pp. 236–239.

[49] Y. Hou, X. Ding, R. Hou, Support vector machine classification prediction model based on improved chaotic differential evolution algorithm, in: 13th Int. Conf. on Natural Comput., Fuzzy Sys. and Know. Disc., 2017, pp. 123–129.

[50] G. B. Huang, Q. Y. Zhu, C. K. Siew, Extreme learning machine: Theory and applications, Neurocomputing 70 (1) (2006) 489–501.

[51] K. Bache, M. Lichman, UCI machine learning repository.
URL `http://archive.ics.uci.edu/ml/index.php`

[52] C. workbench Team, Causality workbench data-repository.
URL `http://www.causality.inf.ethz.ch`

[53] B. Xue, M. Zhang, W. N. Browne, Particle swarm optimisation for feature selection in classification: Novel initialisation and updating mechanisms, Appl. Soft Comput. 18 (4) (2014) 261–276.

[54] H. Wang, H. Sun, C. Li, S. Rahnamayan, J. S. Pan, Diversity enhanced particle swarm optimization with neighborhood search, Inform. Sci. 223 (2) (2013) 119–135.

[55] J. J. Liang, A. K. Qin, P. N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evolut. Comput. 10 (3) (2006) 281–295.

[56] Y. Wang, B. Li, T. Weise, J. Wang, B. Yuan, Q. Tian, Self-adaptive learning based particle swarm optimization, Inform. Sci. 181 (20) (2011) 4515–4538.

[57] Y. Xue, S. Zhong, Y. Zhuang, B. Xu, An ensemble algorithm with self-adaptive learning techniques for high-dimensional numerical optimization, Appl. Math. Comput. 231 (1) (2014) 329–346.

[58] D. B. Fogel, An introduction to simulated evolutionary optimization, IEEE Trans. Neural Networ. 5 (1) (1994) 3–14.

[59] A. K. Qin, V. L. Huang, P. N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evolut. Comput. 13 (2) (2009) 398–417.

[60] Q. Fan, X. Yan, Self-adaptive differential evolution algorithm with discrete mutation control parameters, Expert Syst.with Appl. 42 (3) (2015) 1551–1572.

[61] J. Yang, V. Honavar, Feature subset selection using a genetic algorithm, IEEE Intel. Sys. The. Appl. 13 (2) (2002) 44–49.

[62] X. Qiu, K. C. Tan, J. Xu, Multiple exponential recombination for differential evolution, IEEE Trans. Cybernetics 47 (4) (2017) 995–1006.

[63] J. Derrac, S. Garcia, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evol. Comput. 1 (1) (2011) 3–18.