Self-Adaptive Particle Swarm Optimization for Large-Scale Feature Selection in Classification

YU XUE, Nanjing University of Information Science and Technology
BING XUE and MENGJIE ZHANG, Victoria University of Wellington

Many evolutionary computation (EC) methods have been used to solve feature selection problems and they perform well on most small-scale feature selection problems. However, as the dimensionality of feature selection problems increases, the solution space increases exponentially. Meanwhile, there are more irrelevant features than relevant features in datasets, which leads to many local optima in the huge solution space. Therefore, the existing EC methods still suffer from the problem of stagnation in local optima on large-scale feature selection problems. Furthermore, large-scale feature selection problems with different datasets may have different properties. Thus, it may be of low performance to solve different large-scale feature selection problems with an existing EC method that has only one candidate solution generation strategy (CSGS). In addition, it is time-consuming to find a suitable EC method and corresponding suitable parameter values for a given large-scale feature selection problem if we want to solve it effectively and efficiently. In this article, we propose a self-adaptive particle swarm optimization (SaPSO) algorithm for feature selection, particularly for large-scale feature selection. First, an encoding scheme for the feature selection problem is employed in the SaPSO. Second, three important issues related to self-adaptive algorithms are investigated. After that, the SaPSO algorithm with a typical self-adaptive mechanism is proposed. The experimental results on 12 datasets show that the solution size obtained by the SaPSO algorithm is smaller than its EC counterparts on all datasets. The SaPSO algorithm performs better than its non-EC and EC counterparts in terms of classification accuracy not only on most training sets but also on most test sets. Furthermore, as the dimensionality of the feature selection problem increases, the advantages of SaPSO become more prominent. This highlights that the SaPSO algorithm is suitable for solving feature selection problems, particularly large-scale feature selection problems.

CCS Concepts: • **Computing methodologies** → **Feature selection**; *Genetic algorithms*;

Additional Key Words and Phrases: Feature selection, particle swarm optimization, large-scale, self-adaptive, classification

ACM Reference format:
Yu Xue, Bing Xue, and Mengjie Zhang. 2019. Self-Adaptive Particle Swarm Optimization for Large-Scale Feature Selection in Classification. *ACM Trans. Knowl. Discov. Data* 13, 5, Article 50 (September 2019), 27 pages.
https://doi.org/10.1145/3340848

The work of Y. Xue was supported by National Natural Science Foundation of China (Grant number 61403206, 61876089), by Natural Science Foundation of Jiangsu Province (Grant number BK20141005), by Natural Science Foundation of the Jiangsu Higher Education Institutions of China (Grant number 14KJB520025), and by Priority Academic Program Development of Jiangsu Higher Education Institutions.
Authors' addresses: Y. Xue, Nanjing University of Information Science and Technology, 219 Ningliu Rd, Nanjing, Jiangsu, China; email: xueyu@nuist.edu.cn; B. Xue and M. Zhang, Victoria University of Wellington, Gate 6, Kelburn Parade, Wellington, New Zealand; emails: {bing.xue, mengjie.zhang}@ecs.vuw.ac.nz.

1556-4681/2019/09-ART50 $15.00
https://doi.org/10.1145/3340848

1 INTRODUCTION

In machine learning and data mining, many classification datasets have a large number of features (Wu et al. 2017; Yu et al. 2016). The training time increases as the feature space dimensionality increases. The increasing dimensionality can cause the "curse of dimensionality" problem (Gheyas and Smith 2010). Feature extraction (Stuhlsatz et al. 2012), feature construction (Li and Tao 2012), and feature selection (Xue et al. 2016) are three important dimensionality reduction approaches to tackle with the "curse of dimensionality" problem. In fact, most of the features in the input datasets are redundant or irrelevant features, which not only increases the training time but also reduces the classification accuracy of the learnt classifier. Feature selection is an important data preprocessing technique to overcome such problems, which involves choosing a minimum subset of relevant features from the original large feature set so that the training time is reduced and the learning performance be improved (Yang et al. 2013).

The feature selection problem can be considered as a combinatorial optimization problem. The total number of possible solutions is $(2^n - 1)$ for a feature selection problem with n features. Thus, the number of solutions increases exponentially with respect to n. It might be time-acceptable to find the exact solution for a small-size feature selection problem using an exhaustive method. However, it is impractical to find the exact solution using an exhaustive method for a large-scale feature selection problem. For example, consider a feature selection problem with 57 dimensions. If we use a personal computer to implement the exhaustive method and assume the computer can evaluate 3 billion solutions in 1 second, it would take at least 1.5 years to find the exact solution. Moreover, for a feature selection problem with 100 dimensions, it would take more than 1.3585E+013 years, and for a feature selection problem with 1,000 dimensions, it would take more than 1.1483E+284 years. In this article, feature selection problems with more than 100 dimensions are termed as large-scale problems. Therefore, it is impractical to find the exact solutions by an exhaustive method for the large-scale feature selection problems.

In the past several decades, a variety of heuristic methods have been proposed to find acceptable solutions for the feature selection problems (Chang et al. 2016; Tang and Liu 2014). Based on evaluation methods, feature selection methods can be simply classified into two categories: filter and wrapper (Xue et al. 2016). Filter methods evaluate features independent of any classification method, which may limit the performance. Filter approaches are often computationally less expensive than wrapper approaches, but they are not as effective as wrapper approaches. Wrapper approaches include a classification method in the evaluation function; hence, they are more time-consuming. However, they can typically obtain better results than filter approaches (Xue et al. 2014a).

Wrapper methods evaluate candidate subsets using a classifier. Greedy search based sequential search methods, such as sequential forward selection (SFS) (Whitney 1971) and sequential backward selection (SBS) (Marill and Green 1963), are typical wrapper methods. Their main drawback is the "nesting effect." Because they add or reduce features into/from a feature subset one by one, if one feature is selected or removed, it will not be removed or selected again. Although a plus-l-take-r-away method has been proposed to alleviate the problem (Stearns 1976), the improved approach still suffers from stagnation in local optima. Moreover, the dimensionality in each repeated stage is fixed depending on the predefined values of l and r. Therefore, the sequential forward floating selection algorithm (SFFS) and sequential backward floating selection (SBFS) algorithm are proposed to overcome the above problems (Pudil et al. 1994). The SFFS adds features to an empty feature set using the basic SFS procedure, followed by a series of removing the worst feature from the updated feature set if the removing process can improve the feature set. The SBFS removes features from a complete feature set using the basic SBS procedure, followed by a series of adding the best feature to the updated feature set if the adding process can improve the feature set.

In datasets, some features may interact with each other (Gheyas and Smith 2010); hence, an individually relevant feature may become redundant when it works together with other features. On the other hand, an individually weakly relevant feature may become highly useful when it works with others. Therefore, the traditional filter and wrapper approaches, which evaluate and select features individually, cannot work well. Hence, to address the feature selection problems effectively, a global search technique is required. Evolutionary computation (EC) techniques have recently received much attention from the feature selection community because of their global optimization ability (Xue et al. 2016). The special population-based structure and efficient parallel search manner enable EC methods to have very good global search ability. Some famous EC methods used in feature selection are as follows: genetic algorithms (Ghareb et al. 2016; Holland 1975), genetic programming (Kamath et al. 2012; Koza 1990), ant colony optimization (Dorigo and Gambardella 1997; Neagoe and Neghina 2016), particle swarm optimization (PSO) (Bharti and Singh 2016; Kennedy and Eberhart 1995; Zhang et al. 2015, 2017a), differential evolution (DE) (Al-Ani et al. 2013; Storn and Price 1997), and the firefly algorithm (Yang 2008; Zhang et al. 2017b). A survey of all kinds of work for solving feature selection problems using EC methods can be found in Ref. (Xue et al. 2016).

Although many EC methods have been employed to solve small-scale feature selection problems, existing EC methods still suffer from the problem of stagnation in local optima on large-scale feature selection problems. The solution space of a feature selection problem increases exponentially with the dimensionality of the dataset increases. Thus, the increasing number of features results in a huge solution space. Besides, feature selection is a special problem that is different from other common combinatorial optimization problems, i.e., there are typically more irrelevant or redundant features than relevant features in the datasets, thus, the large number of irrelevant or redundant features generate many local optima in the huge solution space. Therefore, most EC methods still suffer from the problem of stagnation in local optima (Xue et al. 2014a). Another possible cause of this issue is that many of these methods lack the ability to explore and exploit the search space in a proper manner (Al-Ani et al. 2013). Moreover, different datasets may have different properties, and different positions may have different fitness landscape for the solution space of the same dataset. Thus, the suitable search manners for the EC method should be automatically employed according to the feature selection problem and its fitness landscape. However, we do not know the characteristics of the fitness landscape for a given feature selection problem. Hence, to solve a feature selection problem effectively, generally speaking, many different algorithms are tested in advance, and when we test one of the algorithms for a given problem, many experiments should be performed to look for a suitable parameter value set for the algorithm. Obviously, it costs a great deal of computational time to look for a suitable algorithm and its suitable parameter values. In recent years, some EC methods with a self-adaptive mechanism have been proposed for solving different optimization problems and the experimental results showed that they have obvious advantages on optimization problems (Harrison et al. 2018; Pornsing et al. 2016; Sudo et al. 2015; Xue et al. 2014b, 2017; Wang 2011). For example, in order to overcome the drawbacks in PSO, Wang et al. (Wang 2011) have imported a chaotic local search into PSO and proposed a chaotic self-adaptive particle swarm optimization (CSAPSO) algorithm. In the CSAPSO, the velocity was adjusted dynamically so as to deal with various constraints in real-world problems. In (Ying 2011), the CSAPSO was employed to solve the dynamic economic dispatch (DED) problem with value-point effects, and the experimental results indicated that CSAPSO can get a better solution in feasible time. Besides, Pornsing et al. (2016) have designed self-adaptive inertia weight and time varying adaptive swarm topology techniques for PSO. The designed techniques were used to avoid premature convergence by executing the exploration and exploitation stages simultaneously. The numerical experiments showed that the new algorithm with self-adaptive inertia weight and time varying adap-

tive swarm topology outperforms other competitive algorithms. Furthermore, Harrison et al. (2018) have reviewed many kinds of control parameter adaptation based PSO algorithms in a survey work. In (Harrison et al. 2018), they have empirically examined whether the adapted parameters reach a stable point and whether the final parameter values adhere to a well-known convergence criterion.

PSO and its many variant algorithms have been employed for feature selection problems. The frameworks of these variant algorithms are similar. They first initialize a population of particles to represent the possible solutions of an optimization problem and record some important information about the solutions such as the personal best solutions and the global best solution, then they use different formulas and the recorded information to generate a population of new possible solutions. Finally, they repeat the above process till a predefined stop criterion has been satisfied. There are many PSO variant algorithms, and it is the different updating formulas that make them different from each other. The function of these formulas is to generate new solutions; therefore, the formula(s) in each algorithm is (are) termed as candidate solution generation strategy (strategies), i.e., CSGS in this study. Although many PSO variant algorithms have been developed for feature selection problems, most of them employ only one CSGS to generate new solutions. Recently, Bharti and Singh (2016) proposed a binary particle swarm optimization (BPSO) algorithm with an adaptive inertia weight operator, and employed the proposed BPSO to solve the feature selection in text clustering. Their results indicate the proposed BPSO with an adaptive inertia weight operator can select more informative feature set compared to its competitive methods as it obtained better clustering performance. In EC algorithms, there exist many self-adaptive mechanisms. However, first, to our best knowledge, though EC methods with self-adaptive mechanisms have been employed for solving large-scale feature selection in clustering (Bharti and Singh 2016), they have not been tried for solving feature selection problems in classification, not to mention large-scale feature selection in classification. Second, the following two important questions should be answered when designing the strategy pool of the self-adaptive algorithm: (1) Which CSGSs should be used in the pool? (2) How many CSGSs should be used in the pool? To answer these questions, one basic problem should be solved first, i.e., how to identify whether a CSGS is effective? PSO is an effective technique for feature selection problems (Xue et al. 2014a). Therefore, motivated by these questions and methods analysis, we propose the methods to overcome the questions in designing strategy pool of self-adaptive algorithms, and develop a self-adaptive PSO (SaPSO) algorithm for solving large-scale feature selection problems in classification. In the SaPSO algorithm, several CSGSs with different characteristics are maintained, and the previous experiences of generating promising solutions are used to adaptively choose the suitable CSGSs to generate new solutions in the subsequent generations.

1.1 Goals

The overall goal of this article is to propose a new self-adaptive PSO algorithm for feature selection, particularly for large-scale feature selection. The more detailed objectives are described as follows:

(1) To design a new self-adaptive PSO algorithm with multiple CSGSs, which are used self-adaptively during evolutionary process.
(2) To carry out theoretical study of designing the strategy pool of the SaPSO algorithm.
(3) To investigate whether a self-adaptive PSO algorithm can achieve good performance for feature selection, especially for large-scale feature selection.

1.2 Organization

The remainder of this article is organized as follows. In Section 2, we provide background information. Section 3 describes the new algorithm with the representation of solutions, the strategy

pool designing methods, and the self-adaptive mechanism. Section 4 describes experiment design, and Section 5 presents results with discussions. Section 6 provides conclusions and future research work.

2 RELATED WORK

2.1 Initialization and Updating Mechanisms in PSO for Feature Selection

PSO has been widely used in feature selection problems (Xue et al. 2013, 2014a). In the framework of PSO, initialization is very important, and it significantly affects the performance of PSO. Recently, Xue et al. (2014a) improved the performance of PSO for feature selection by improving its initialization. In Ref. (Xue et al. 2014a), Xue et al. proposed three new initialization approaches, and they have found a very good initialization strategy for the feature selection. In this article, we employ the same approach as used in the final proposed algorithm in Ref. (Xue et al. 2014a), which is briefly described as follows.

Mixed initialization. In this initialization strategy, most particles are initialized using a small number of features and the remaining particles are initialized using large feature subsets (Xue et al. 2014a).

Actually, the feature selection problem has two objectives, where the first one is classification accuracy/classification error, and the second one is the number of selected features (solution size). For example, suppose a given dataset with four features and a decoded solution is $\{0, 1, 0, 1\}$. Obviously, the second objective of the solution is 2. To calculate the first objective, firstly, we use the second column, the fourth column, and the label column of the dataset to form a new dataset. Then, the k-nearest neighbor (kNN) method is used to obtain the classification accuracy on this new dataset, which is also the first objective of the solution. In order to satisfy the two objectives of feature selection, in addition to initialization, four updating mechanisms were investigated in Ref. (Xue et al. 2014a). Because it has been proved that the second one is a promising updating strategy, we use the second updating strategy in this study, and it is briefly presented as follows.

Classification performance as the first priority. pbest or *gbest* is updated in two situations, where *pbest* represents the best solution that is found by a particle so far while *gbest* means the best solution obtained by the swarm so far (Kennedy and Eberhart 1995). Use *pbest* as an example, for the first situation, if the classification performance of the particle's new position is better than *pbest*, *pbest* will be updated and replaced by the new position. In this case, the number of selected features will be ignored. For the second situation, if the classification performance is the same as *pbest* and the number of features is smaller, the current *pbest* will be replaced by the particle's new position.

2.2 Related Work on Self-Adaptive Algorithms

In the last decade, the self-adaptive mechanism in EC methods has attracted the attention of researchers, and powerful self-adaptive EC methods have been proposed. For example, Qin et al. (2009) introduced a self-adaptive mechanism into DE and proposed a self-adaptive DE (SaDE). Their experimental results show SaDE is more effective in obtaining better quality solutions. Additionally, Li et al. (2012) introduced an adaptive framework into PSO to develop a self-learning PSO (SLPSO) algorithm. Different from SaDE, their adaptive scheme was implemented at the individual level. Their experimental study on a set of 45 test functions and two real-world problems show that the self-adaptive mechanism is helpful for solving different types of problems, particularly the problems that have very complex fitness landscapes. Furthermore, Wang et al. (2011) proposed a self-adaptive learning based PSO (SLPSO) with a new probability model that was used

to describe the probability of a strategy. They compared their SLPSO with eight PSO variants on 26 numerical optimization problems with different characteristics and an economic load dispatch problem in power systems. Their results indicate that SLPSO can update the best solution records. In recent years, Xue et al. (2014b, 2017) proposed some improved self-adaptive EC techniques to solve the continuous and discrete optimization problems.

Recently, the EC methods with self-adaptive mechanisms have been proposed to solve large-scale continuous optimization problems, and the experimental results show that these algorithms have obvious advantages on the continuous numerical optimization problems with high dimensionality (Xue et al. 2014b). However, to our best knowledge, though EC methods with self-adaptive mechanisms have been employed for solving large-scale feature selection in clustering (Bharti and Singh 2016), they have not been tried for solving feature selection problems in classification, not to mention large-scale feature selection in classification. In this article, we investigate a self-adaptive PSO algorithm to see whether it can achieve good performance for feature selection in classification, especially for large-scale feature selection in classification.

3 SELF-ADAPTIVE PARTICLE SWARM OPTIMIZATION FOR FEATURE SELECTION

3.1 Representation of Solutions

There are several representation schemes for feature selection in the literature (Xue et al. 2016). In this article, feature selection is transformed into a "0" and "1" combinatorial optimization problem, in the same manner as that in (Xue et al. 2014a). Thus, the representation of a solution is a binary string. This string has D dimensions, where D means the total number of features. We use continuous encoding in PSO, and the range of each dimension of the position vector is limited in $[0, 1]$. To transfer a continuous position vector to a binary string, a threshold θ is set in advance. If the value of the dth dimension of the position is greater than θ, the corresponding value in the binary vector is set to 1, which represents that the dth feature is selected. Otherwise, the value in the binary vector is set to 0, which represents that the dth feature is not selected.

3.2 Methods for Designing Strategy Pool

Different from the other variant algorithms of PSO that use only one CSGS to generate new particles, the SaPSO algorithm uses multiple CSGSs to generate new particles. In the SaPSO algorithm, the multiple CSGSs are maintained in a specific component that is termed as strategy pool. In order to design the strategy pool for the SaPSO, we have firstly implemented 25 CSGSs that are commonly used and representative CSGSs in the literature about PSO (The detailed information of the 25 CSGSs can be seen in the complementary materials). The strategy pool is not constitute of all the 25 CSGS, i.e., only the suitable CSGSs from the 25 CSGSs are put in the strategy pool. In this subsection, a method for selecting CSGSs is introduced.

The choice of CSGSs to form the strategy pool has two aspects to consider. (1) How many CSGSs should be selected to form the pool? (2) Which CSGSs should be selected? There is a basic question here, i.e., how to identify which CSGSs are effective? We can identify which CSGSs are effective if there are only one dataset. However, there are a large number of datasets, and we expect the CSGSs can perform well on the large-scale datasets. The only information that can be obtained is the performance of the CSGSs on each dataset by doing experiments. Hence, we need a method to comprehensively evaluate the performance of the CSGSs.

Analytic hierarchy process (AHP) is a famous multicriteria decision making technique (Aguaron et al. 2016; Saaty 1990). The main characteristics of this approach are as follows: the modeling of the problem using a hierarchical structure that reflects all the relevant aspects of the problem; the use of pairwise comparisons to incorporate the preferences of decision makers; the derivation of

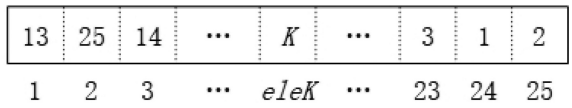

Fig. 1. An example of a *seq* with 25 CSGSs.

priority vector for the alternative properties. In this article, AHP is used to synthetically evaluate the CSGSs. According to the AHP, a score or order, which is used to evaluate the CSGSs, should be assigned to each CSGS. Because the total number of the CSGSs is greater than the allowed scale of the existing AHP, we first propose a relative permutation order based scaling method (RPOSM) to divide the CSGSs into small groups and assign an order/score to each CSGS according to their performance on each feature selection problem. Nevertheless, these CSGSs in the same group have the same order. Based on RPOSM, a new analytic hierarchy process (RPOSM-AHP) is proposed to comprehensively evaluate the CSGSs. In this section, we first introduce RPOSM and then describe RPOSM-AHP.

3.2.1 Relative Permutation Order Based Scaling Method. RPOSM is designed to get the order of each CSGS on each dataset. The input of RPOSM is a sequence of CSGSs sorted according to their fitness values on a feature selection problem. The output of RPOSM is the order of each CSGS.

RPOSM is described as follows. Suppose *seq* denotes a sorted sequence in decreasing according to the importance of its elements. The order o_K of the Kth element is calculated according to Equation (1) as follows.

$$o_K = OS - \lceil ele_K / GS \rceil + 1 \tag{1}$$

where ele_K represents the serial number of the Kth element in the *seq*, o_K is the score of the Kth element, $\lceil \cdot \rceil$ is the rounding up operator, GS represents the size of each group, and OS is the order size that is determined by real-world problems.

For example, suppose there are 25 CSGSs and they are sorted as in Figure 1. In addition, we suppose that OS = 9, GS = 3, and we want to calculate the orders for CSGS13 and CSGS2. Thus, according to Equation (1), they are calculated as follows:

$$\begin{aligned} o_{13} &= OS - \lceil eleK/GS \rceil + 1 \\ &= 9 - \lceil 1/3 \rceil + 1 \\ &= 9 \\ o_2 &= OS - \lceil eleK/GS \rceil + 1 \\ &= 9 - \lceil 25/3 \rceil + 1 \\ &= 1. \end{aligned}$$

3.2.2 RPOSM Based Analytic Hierarchy Process. RPOSM-AHP is proposed based on RPOSM. In this article, a hierarchical structure with three levels is constructed. RPOSM-AHP is presented in four steps as follows.

Step 1 Construct hierarchies

"To select the suitable CSGSs to form strategy pool" is the overall goal level. We use NI selected training sets to test the performance of NS CSGSs. The size of the datasets changes from small to large. The NI selected training sets are used as the criteria level. Because our solving preference is the large-scale problems, the datasets with large-scales are put in the front of criteria level. NS CSGSs are used as the attribute level. Since our selection preference is the CSGSs that perform better on the datasets, the CSGSs are sorted according to their performance on each dataset. The hierarchical structure is shown in Figure 2.

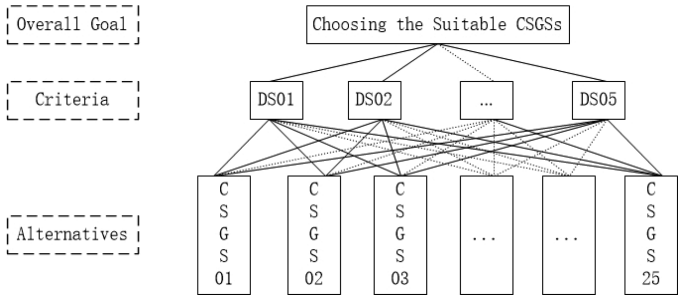

Fig. 2. The hierarchical structure of RPOSM-AHP.

Step 2 Construct pairwise comparison matrices and check consistency

Step 2.1 Construct pairwise comparison matrices

Firstly, sort the datasets at the criteria level according to the relative solving preference. Secondly, obtain the sorted sequence of CSGSs according to the experimental results on each dataset. Thirdly, use RPOSM to calculate the order number of each CSGS in each sorted sequence. Finally, we construct pairwise comparison matrices at the criteria level and attribute level as follows. Let's take attribute level for example, suppose A represents a comparison matrix of the CSGS on one dataset, and

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1s} \\ a_{21} & a_{22} & \dots & a_{2s} \\ \dots & \dots & \dots & \dots \\ a_{s1} & a_{s2} & \dots & a_{ss} \end{bmatrix}. \tag{2}$$

Suppose o_i and o_j represent the order numbers of CSGS i and CSGS j, respectively. Thus, for $\forall a_{ij} \in A\ (j = 1, 2, \dots s,\ i \le j)$, a_{ij} is calculated as follows:

$$a_{ij} = \begin{cases} o_i - o_j + 1 & if\ o_i \ge o_j \\ 1/(o_j - o_i + 1) & \text{otherwise} \end{cases}, \tag{3}$$

a_{ji} is calculated as follows.

$$a_{ji} = 1/a_{ij}, \tag{4}$$

where $i = 1, 2, \dots, s$, $j = 1, 2, \dots, s$. a_{ij} represents the ith row and jth element of matrix A. $a_{ii} = 1$, $a_{ij} = 1/a_{ji}$, $a_{ij} > 0$, and s is the total number of datasets/CSGSs in criteria level/attribute level.

All pairwise comparison matrices at the criteria level and attribute level can be obtained according to Formulas (1–4).

Step 2.2 Check consistency

Step 2.2.1 Calculate the maximum eigenvalue of each pairwise comparison matrix as follows:

$$\lambda \max = (1/s) \cdot \sum_{i=1}^{s} (AW)_i / W_i, \tag{5}$$

where λ max represents the maximum eigenvalue of the pairwise comparison matrix, W is the eigenvector of the pairwise comparison matrix A, and$(AW)_i$ is the ith element of vector AW.

Step 2.2.2 Calculate the consistence index (CI) *(Saaty 1990) as follows:*

$$CI = (\lambda \max - s)/(s - 1) \tag{6}$$

Step 2.2.3 Calculate the consistence ratio (CR) (Saaty 1990) as follows.

$$CR = CI/RI, \tag{7}$$

where RI is a random index (Saaty 1990) given in advance. When $CR < 0.1$, we consider that the consistency of the pairwise comparison matrix is acceptable.

Step 3 Get a priority vector at each single level

For each pairwise comparison matrix in Step 2.1, the relative priority at each level can be calculated as follows:

$$\overline{a}_{ij} = a_{ij} / \sum_{k=1}^{s} a_{kj} \quad i, j = 1, 2, \ldots, s \tag{8}$$

$$W_i = \sum_{j=1}^{s} \overline{a}_{ij} / \sum_{k=1}^{s} \sum_{j=1}^{s} \overline{a}_{kj} \qquad i = 1, 2, \ldots, s. \tag{9}$$

Thus, we obtain a priority vector that is represented by $C = (C_1, C_2, \ldots, C_{NI})$ at the criteria level, and we obtain the priority vectors which are represented by $W_i = (w_{i1}, w_{i2}, \ldots, w_{iNS})$, $i = 1, 2, \ldots, NI$, at the attribute level.

Step 4 Calculate the total priority vector

Based on the results in Step 3, the total priority vector can be calculated as follows:

$$W^{'} = \left(\sum_{i=1}^{NI} c_i w_{i1}, \sum_{i=1}^{NI} c_i w_{i2}, \ldots, \sum_{i=1}^{NI} c_i w_{iNS} \right). \tag{10}$$

Finally, the performance of the CSGSs can be evaluated comprehensively by RPOSM-AHP. Besides, RPOSM-AHP introduces relative solving preference at the criteria level, which can make RPOSM-AHP satisfy the requirement for solving large-scale problems.

3.3 Candidate Solution Generation Strategies

There are many PSO variant algorithms in the literature. It is impractical to investigate and implement all of their CSGSs. We only chose and implemented 25 CSGSs which are representative, frequently used, from high-quality articles or recently proposed. We have tried our best to choose and implement the CSGSs as more as possible, we believe the 25 CSGSs cover almost all different types of CSGSs for designing the strategy pool. Due to the space limit, the 25 CSGSs are provided in the complementary material. Based on the initial experimental results, five CSGSs are finally selected. The five CSGSs are described as follows.

(1) Standard PSO is an efficient EC method (Xue et al. 2014a), which is the main strategy used to generate new solutions. This strategy searches for the optimal solution by updating the position and velocity of each particle according to the following equations:

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{11}$$

$$v_{id}^{t+1} = w * v_{id}^{t} + c_1 * r_1 * \left(p_{id} - x_{id}^{t} \right) + c_2 * r_2 * \left(p_{gd} - x_{id}^{t} \right), \tag{12}$$

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

(2) A different velocity updating strategy was proposed in Ref. (Wang et al. 2013), which is described as follows:

$$x_{id}^{t+1} = r_1 * x_{id}^t + r_2 * p_{gd} + r_3 * \left(x_{ad}^t - x_{bd}^t\right), \tag{13}$$

where x_{ad}^t and x_{bd}^t are position vectors of two random particles. The position updating method and other variables have the same meaning as mentioned before.

(3) Estimation-based velocity updating strategy from Ref. (Wang et al. 2011) is selected in this article, which is described as follows:

$$c = \frac{(D-1)N(0,1)}{D} + \frac{C(0,1)}{D} \tag{14}$$

$$v_{id}^{t+1} = \left(mean_{id}^t - x_{id}^t\right) + \frac{c}{\sqrt{3}}\sqrt{\left(p_{id} - mean_{id}^t\right)^2 + \left(x_{id}^t - mean_{id}^t\right)^2 + \left(x_{ad}^t - mean_{id}^t\right)^2}, \tag{15}$$

where $N(0,1)$ and $C(0,1)$ represent two numbers randomly generated by the Gaussian distribution and Cauchy distribution, respectively. $mean_{id}^t$ is set to as same as that in Ref. (Xue et al. 2014b). The position updating method and other variables have the same meaning as mentioned before.

(4) The CLPSO velocity updating strategy from Ref. (Liang et al. 2006) is selected in this article, which is described as follows:

$$v_{id}^{t+1} = w * v_{id}^t + c_1 * r_1 * \left(pbest_{f_i(d)} - x_{id}^t\right), \tag{16}$$

where $f_i = [f_i(1), f_i(2), \ldots, f_i(D)]$ defines whose personal best 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 updating method and other variables have the same meaning as mentioned before.

(5) An improved CLPSO velocity updating strategy called the PSO-CL-pbest strategy from Ref. (Wang et al. 2011) is also selected in this article. The strategy is given as follows:

$$v_{id}^{t+1} = w * v_{id}^t + 0.5 * c_1 * r_1 * \left(pbest_{f_i(d)} - x_{id}^t + p_{gd} - x_{id}^t\right), \tag{17}$$

where the variates and the position updating method have the same meaning as mentioned before.

3.4 The Self-Adaptive Mechanism

The main objectives of the self-adaptive mechanism are to produce the probabilities for the CSGSs according to their performance, and choose a suitable CSGS for each particle based on these probabilities. There are two most important problems here, i.e., (1) How to generate new probabilities for the CSGSs? (2) How to select a suitable CSGS for each particle? In this article, we adopt relatively simple methods to solve the two problems because we focus mainly on solving the CSGS selection problem for the self-adaptive algorithms.

The CSGSs that are successfully used in recent generations should be continuously used in future generations. When a CSGS cannot perform well, it should be replaced by another CSGS that may perform well. We use a simple self-adaptive mechanism which is described as follows. All the CSGSs in the strategy pool are assigned an initial probability and the probabilities are changed during evolution. Let p_j represent the selection probability of the jth strategy (where $j = 1, 2, \ldots, Q$, and Q is the number of CSGSs. Thus the initial probability of each CSGS is $1/Q$). The sum of these probabilities is 1 and they are recalculated according to the performance of the CSGSs on generating new solutions. In this article, the roulette wheel method (Fogel 1994) is used to select a CSGS. Subsequently, a candidate solution is generated by applying the selected CSGS to

the corresponding particle. Then, the candidate solution is evaluated and the updating mechanism that was described in Section 2 is employed to determine whether the *pbest* and *gbest* should be updated. The information that whether the generated solution is better than the corresponding *pbest* is recorded by the elements $nsFlag_{i,j}$ and $nfFlag_{i,j}$ ($i = 1, 2, \ldots, ps$, $j = 1, 2, \ldots, Q$, where ps is the number of particles and Q is the number of CSGSs) in the binary matrices $nsFlag_{ps\times Q}$ and $nfFlag_{ps\times Q}$. It is implemented as follows.

At the beginning of a generation,

$$nsFlag = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{ps\times Q} \quad \text{and} \quad nfFlag = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{ps\times Q}, \tag{18}$$

suppose the jth strategy is selected for the ith particle, if the generated new solution is better than the corresponding *pbest*, then $nsFlag_{i,j} = 1$, otherwise, $nfFlag_{i,j} = 1$. After the evolution process of the current generation is finished, we sum all the rows in *nsFlag* and *nfFlag*, then the number of the new solutions that are generated by the jth CSGS and successfully enter into the next generation is recorded in the element $S_{k,j}$ ($k = 1, 2, \ldots, LP$, $j = 1, 2, \ldots, Q$, where LP represents a number of generations, it means the probabilities of CSGSs will be recalculated if the evolution process is repeated for LP generations) of another matrix $S_{LP\times Q}$. Similarly, the number of the new solutions generated by the jth CSGS but unsuccessfully enter into the next generation is recorded in the element $F_{k,j}$ of another matrix $F_{LP\times Q}$. Meanwhile, the matrices *nsFlag* and *nfFlag* are initialized as shown in Equation (18), so that they can record the information in the next generation.

The above process is repeated LP generations to learn the success and fail information for the CSGSs, after evolving for LP generations, the probability values of the CSGSs are recalculated.

At the first generation ($k = 1$) of each LP generations,

$$S = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{LP\times Q} \quad \text{and} \quad F = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{LP\times Q}. \tag{19}$$

After the evolutionary process repeats LP generations, the strategy selection probabilities of the CSGSs are recalculated based on the statistical data stored in matrices S and F. The probability for the jth ($j = 1, 2, \ldots, Q$) strategy is calculated as follows:

$$S_j^1 = \sum_{k=1}^{LP} S_{k,j} \tag{20}$$

$$S_j^2 = \begin{cases} \varepsilon, & if S_j^1 = 0 \\ S_j^1, & \text{otherwise} \end{cases} \tag{21}$$

$$S_j^3 = S_j^1 \Big/ \left(S_j^2 + \sum_{k=1}^{LP} F_{k,j} \right) \tag{22}$$

$$p_j = S_j^3 \Big/ \sum_{j=1}^{Q} S_j^3, \tag{23}$$

where S_j^3 is the rate of the new solutions generated by the jth strategy and replaced their corresponding *pbests* successfully within *LP* generations. Meanwhile, the matrices S and F are initialized as Equation (19). The small value $\varepsilon = 0.0001$ is used to avoid division by zero. The probabilities are normalized by Equation (23) to ensure that they always sum to 1.

Equations (18)–(23) are employed to generate new probabilities for the CSGSs according to their performance during *LP* generations evolution. The CSGSs are selected according to the new probabilities. It is obvious that the larger the probability value, the larger probability that the corresponding CSGS is selected to generate new solutions in the next *LP* generations.

Algorithm 1 shows the pseudo-code of the SaPSO algorithm.

ALGORITHM 1: Pseudo code of the SaPSO algorithm

Initialization:

Set parameter values including number of fitness evaluations (NFE), current number of fitness evaluations ($cFE = 0$), population size (ps), Q, $p_j = 1/Q$ for each $j \in \{1, 2, \ldots, Q\}$, $LP = 10$, $nsFlag$, $nfFlag$, $S_{LP\times Q}$, $F_{LP\times Q}$, $curiter = 0$, $flagiter = 0$, and and so on. Initialize the positions and velocities of each particle. Evaluate each particle and store the *pbest*s and *gbest*;

while ($cFE < NFE$) **do**
  for each $i < ps$ **do**
    Select one CSGS from the strategy pool for x_i by the roulette wheel selection method based on $\{p_1, p_2, \ldots, p_Q\}$. Suppose the jth CSGS is selected. Generate a new particle x_i^{new} by the selected CSGS, and calculate its fitness value;
    if x_i^{new} is better than x_i **then**
      $nsFlag_{i,j} = 1$;
      if x_i^{new} is better than $pbest_i$ **then**
        Update $pbest_i$ with x_i^{new};
        if x_i^{new} is better than *gbest* **then**
          Update *gbest* with x_i^{new};
        end if
      end if
    else
      $nfFlag_{i,j} = 1$;
    end if
    $cFE = cFE + 1$;
    Replace x_i with x_i^{new};
  end for
  $curiter = curiter + 1$;
  $k = curiter - flagiter$;
  Replace the kth row of $S_{LP\times Q}$ with the sum of all the rows in $nsFlag$;
  Replace the kth row of $F_{LP\times Q}$ with the sum of all the rows in $nfFlag$;
  Initial $nsFlag$ and $nfFlag$ as Equation (18);
  if ($curiter - flagiter$) $= LP$ **then**
    $flagiter = curiter$;
    Update $\{p_1, p_2, \ldots, p_Q\}$ based on $S_{LP\times Q}$ and $F_{LP\times Q}$ as Eqs. (20)–(23);
    Initial $S_{LP\times Q}$ and $F_{LP\times Q}$ as Eq. (19);
  end if
end while

Output:

gbest;

Table 1. Information of the Datasets

Datasets	NoE	NoF	NoC	Datasets	NoE	NoF	NoC
No.1	1,000	5,000	2	No.9	900	561	6
No.2	360	1,300	2	No.10	900	522	3
No.3	1,080	856	9	No.11	600	500	2
No.4	1,062	301	2	No.12	1,000	64	10
No.5	675	256	10	No.13	208	60	2
No.6	1,040	617	26	No.14	596	30	2
No.7	1,000	649	10	No.15	32	56	3
No.8	1,200	561	12				

4 EXPERIMENT DESIGN

4.1 Datasets and Classification Method

The datasets used in this article are shown in Table 1, which were chosen from the UCI Machine Learning Repository (Bache and Lichman 2016). They are available at http://archive.ics.uci.edu/ml/index.php. In Table 1, No.1 to No.15 represent Gisette, MicroMass, CNAE, GrammaticalFacial-Expression, SemeionHand writtenDigit, Isolet5, MultipleFeaturesDigit, HAPT, Har, UJIIndoorLoc, MadelonValid, OpticalRecognitionofHandWritten, ConnectionistBenchData, WDBC, LungCancer. Besides, NOE, NoF, and NoC are abbreviations of number of examples, number of features, and number of classes. Some instances of several datasets were randomly reduced for saving experimental time, i.e., 519, 1,000, 6,567, 2,852, 211, 2,823 instances were reduced for datasets No.6, No.7, No.8, No.9, No.10, No.12, respectively. The first five datasets in Table 1 were used for algorithm design, and for saving time, only the 4th to 15th datasets in Table 1 were used to extensively test the performance of the designed algorithm and the other non-EC methods and EC methods. The datasets consist of different numbers of examples, features, and classes. Each dataset was divided into two partitions: one was used as a training set, formed by randomly selecting 70% of the examples from the original dataset. The other partition was used as a test set, and it consists of the remaining examples. The *k*NN method was used as the classification method to evaluate the feature subsets generated by the non-EC methods and EC methods, where $k = 3$. In this article, *k*NN method is used as the classification method since it is one of the most popular classification algorithms used for feature selection due to its promising classification performance and simplicity (Xue et al. 2016). In *k*NN, 3 fold cross-validation is used to measure the classification accuracy.

4.2 Experiment Methods and Benchmark Algorithms

The first five datasets in Table 1 were used to test the performance of the 25 CSGSs. The approaches described in Section 3.2 were used to select CSGSs for the strategy pool of SaPSO. At this stage, we only ran the algorithms with different CSGSs twice in order to save time. The experimental results were used as the inputs of RPOSM and RPOSM-AHP. Then the sorted sequence of the 25 CSGSs on each dataset was obtained. From the first CSGS to the eleventh CSGS, by gradually adding one CSGS one time into the strategy pool, 10 algorithms with different strategy pool sizes were designed. The five datasets were employed again to test the performance of the 10 algorithms. Using RPOSM and RPOSM-AHP on the experimental results, the sorted sequence of the 10 algorithms was obtained. By analyzing CSGCs of the algorithms at the front of the sequence, we selected CSGS1, CSGS11, CSGS13, CSGS14, and CSGS15, which were described in Section 3.2, to constitute the final strategy pool, and termed the algorithm that with the final strategy pool as SaPSO.

Table 2. The Parameter Values of the Comparison Algorithms

Algorithms	Parameter values	Algorithms	Parameter values
LRS21	$l = 2; r = 1$		$F = 0.5; CR = 0.1;$
LRS31	$l = 3; r = 1$	DE	$NFE = 1{,}000{,}000$
LRS32	$l = 3; r = 2$		
	$CR = 0.7; MR = 0.1;$		Initial $CR = 0.5;$
GA	$SR = 0.5;$	SaDE	$F \in N(0.5, 0.3^2);$
	$NFE = 1{,}000{,}000$		$LP = 10;$
			$NFE = 1{,}000{,}000$
	$C_1 = C_2 = 1.49618;$		
Original PSO	$w = 0.7298;$		$p_j = 0.2; LP = 10;$
	$NFE = 1{,}000{,}000$		$ps = 100; \theta = 0.6;$
	$C_1 = C_2 = 1.49618;$	SaPSO	$Ub = 1; Lb = 0;$
Standard PSO	$w \in [0.9, 0.4];$		$Ubv = 0.5; Lbv = -0.5;$
	$NFE = 1{,}000{,}000$		$NFE = 1{,}000{,}000$

To further show the effectiveness of SaPSO, non-EC methods (including SFS (Whitney 1971), SBS (Marill and Green 1963), LRS21 ($l = 2, r = 1$) (Pudil et al. 1994; Stearns 1976), LRS31 ($l = 3$, $r = 1$) (Pudil et al. 1994; Stearns 1976), LRS32 ($l = 3$, $r = 2$) (Pudil et al. 1994; Stearns 1976) SFFS, and SBFS (Pudil et al. 1994)) and EC methods (including GA (Yang and Honavar 1998), standard PSO (Kennedy and Eberhart 1995; Xu et al. 2007; Xue et al. 2014a), original PSO (Xue et al. 2014a), DE (Storn and Price 1997), and SaDE (Qin et al. 2009)) were employed for comparisons. Different from original PSO, standard PSO has an inertia weight that decreases from 0.9 to 0.4. There are two important objectives for feature selection, one is the classification accuracy, the other one is the number of features, i.e., solution size. In fact, a small solution size represents a small training set and a small test set. So, a small solution size means less training time and less test time will be spent. Thus, we will compare the solution sizes, training accuracies, and test accuracies obtained by these algorithms.

4.3 Parameter Settings

The following parameter values were used: for LRS, three groups of parameters were set according to Ref. (Pudil et al. 1994), i.e., $l = 2$, $r = 1$; $l = 3$, $r = 1$; $l = 3$, $r = 2$. The population size of the algorithm with only one CSGS or EC methods was set to 100. The dimensionality of particles was equal to the features in the corresponding dataset in Table 1. The threshold $\theta = 0.6$ in the encoding scheme. The upper boundary of each dimensionality was 1. The low boundary of each dimensionality was 0. The upper/lower boundary of the velocity is $0.5/-0.5$. Moreover, for fair comparisons, we use NFE as a stop criterion. For saving time, we set the NFE to 500,000 to test the performance of SaPSO with each of the 25 CSGSs, But in order to thoroughly test the performance of the 10 different SaPSO algorithms by combining the CSGSs, we set the NFE to 1,200,000. For saving time, the NFE was set to 1,000,000 for the final SaPSO, GA, standard PSO, original PSO, DE, and SaDE. For the 25 CSGSs, the parameters were set as provided in the original papers. The detailed parameter values of the comparison algorithms are presented in Table 2. Some particular parameters are explained as follows: For standard PSO, $w \in [0.9, 0.4]$ and it decreased gradually with respect to the current number of NFE. For each CSGS in SaDE, initial $CR = 0.5$, F is selected from normal distribution with $\mu = 0.5$ and $\sigma = 0.3$, LP was empirically set to 10. For SaPSO, the initial $p_j = 0.2$, the LP was empirically set to 10. The 12 datasets which are from the 4th to the 15th

Table 3. Sorted CSGSs on the First Five Datasets

	1	2	3	4	5	6	7
No.1	CSGS15	CSGS 14	CSGS 13	CSGS 6	CSGS 1	CSGS 8	CSGS 25
No.2	CSGS 13	CSGS 15	CSGS 3	CSGS 14	CSGS 11	CSGS 1	CSGS 6
NO.3	CSGS 15	CSGS 13	CSGS 14	CSGS 6	CSGS 3	CSGS 7	CSGS 2
NO.4	CSGS 13	CSGS 15	CSGS 1	CSGS 14	CSGS 25	CSGS 8	CSGS 6
NO.5	CSGS 15	CSGS 14	CSGS 13	CSGS 6	CSGS 8	CSGS 3	CSGS 2

Table 4. Sorted CSGSs

NO.	1	2	3	4	5	6	7	8	9	10	11
Sorted CSGSs	CSGS13	CSGS15	CSGS14	CSGS1	CSGS3	CSGS6	CSGS8	CSGS25	CSGS2	CSGS7	CSGS11

datasets of Table 1 were used to test SaPSO and its counterparts. Each algorithm was run 30 times on each dataset.

5 RESULTS

5.1 Results for the Strategy Pool Design of SaPSO

We have done preliminary experiments of using the 25 CSGSs on the first five datasets and obtained the sorted CSGSs on each dataset by comparing the quality of the solutions found by these CSGSs. Table 3 shows the first seven CSGSs in the sorted sequence on each dataset. Limited by the page space, only seven CSGSs are listed in each sorted sequence.

Because our focus is solving large-scale feature selection problems, we set the sorted sequence of datasets as <No.1 No.2 No.3 No.4 No.5>. Therefore, using the experimental results in Table 3 and <No.1 No.2 No.3 No.4 No.5> as inputs for the approaches described in Section 3.2.2, we obtained a sorted sequence of the CSGSs. For short, only the used 11 CSGSs are listed in Table 4.

The output values of RPOSM-AHP are shown as follows. λ max of A_1 is 5.06, where A_1 is the pairwise comparison matrix on criteria level, CI is 0.017. According to AHP, the results are reliable only when $CR < 0.1$, and RI is an experimental value, the value can be obtained by finding in table (Saaty 1990). RI of five order matrix is 1.12, CR is $0.015 < 0.1$; λ max of A_l ($l = 2, 3, \ldots, 6$) is 26.01, where A_l is the pairwise comparison matrix on attribute level, and CI is 0.04, RI of a 9 order matrix is 1.45, $CR = 0.027 < 0.1$. Hence, the results are reliable.

Based on the experimental results shown in Table 4, we designed 10 algorithms and numbered them from ALG 1 to ALG 10 through adding the number of CSGSes. Thus, the strategy pool in ALG1 consists of the first two strategies in Table 4. The strategy pool in ALG2 consists of the first three strategies in Table 4, and so on. Hence, the strategy pool of the 10th algorithm consists of all the strategies in Table 4. We also employed the five datasets to test the 10 algorithms. Table 5 summarizes the mean value of the best fitness values obtained by the 10 algorithms on each dataset.

We employed RPOSM and RPOSM-AHP again to obtain the sorted vector of these 10 algorithms. The detailed process was omitted to save space, we only list the final results as follows. (ALG1, ALG2, ALG3, ALG10, ALG4, ALG6, ALG5, ALG8, ALG7, ALG9). All the first five algorithms includes CSGS13, CSGS15, CSGS14, and CSGS1. ALG10 was in the fourth position, and CSGS11 was employed in ALG10, which shows that although CSGS11 is ranked the lowest, it works well together with the top 4 CSGSs to improve the performance. Thus, in the final algorithm, CSGS13, CSGS15, CSGS14, CSGS1, and CSGS11 were employed to constitute strategy pool of the SaPSO algorithm. Since the strategy pool of SaPSO is comprised of CSGS13, CSGS15, CSGS14, CSGS1,

Table 5. The Mean Fitness Values of the Best Solutions Obtained by Different Algorithms on the Five Datasets

	Gisette	MicroMass	CNAE	GrammaticalFacialExpression	SemeionHandwrittenDigit
ALG 1	0.9933	0.9676	0.9342	0.9357	0.9192
ALG 2	0.9933	0.9583	0.928	0.9264	0.9162
ALG 3	0.9933	0.9491	0.9296	0.9262	0.9014
ALG 4	0.9783	0.9398	0.9295	0.9295	0.8843
ALG 5	0.9733	0.926	0.9295	0.9232	0.8867
ALG 6	0.97	0.9583	0.9248	0.928	0.9012
ALG 7	0.9683	0.9398	0.9295	0.9216	0.8894
ALG 8	0.9583	0.9352	0.9296	0.9216	0.8916
ALG 9	0.9567	0.9306	0.9247	0.9263	0.8964
ALG 10	0.9817	0.9537	0.9248	0.9233	0.8963

Table 6. The Mean Fitness Values of the Best Solutions Obtained by SaPSO and its CSGSs on the Five Datasets

	Gisette	MicroMass	CNAE	GrammaticalFacialExpression	SemeionHandwrittenDigit
CSGS13	0.9675	**0.9676**	0.9318	0.9311	0.9069
CSGS15	**0.9975**	0.963	0.9319	0.9295	**0.9251**
CSGS14	0.975	0.9491	0.9288	0.9271	0.9067
CSGS1	0.955	0.9468	0.9193	0.9281	0.8804
CSGS11	0.9325	0.9329	0.9083	0.9107	0.8213
SaPSO	**0.9975**	0.9653	**0.9336**	**0.9311**	0.9165

Bold face indicates the best accuracy values.

and CSGS11, we employed the same methods to compare SaPSO with CSGS13, CSGS15, CSGS14, CSGS1, and CSGS11. For saving time, each method was run four times on each dataset. The mean values of the best classification accuracy values of the methods on the five datasets are given as in Table 6.

Using the methods in Section 3.2, the sorted sequence was obtained as follows: <SaPSO, CSGS15, CSGS13, CSGS14, CSGS1, CSGS11>. It is demonstrated that the algorithm with multiple strategies is better than that only with its separate strategies.

5.2 Computational Results and Comparisons

In this section, we show that SaPSO performs better than non-EC methods (including SFS, SBS, LRS21, LRS32, SFFS, and SBFS) and other EC methods (including GA, PSO, DE, and SaDE) when solving feature selection problems. To further test the performance, we tested them on 12 datasets that are the fourth to the fifteenth datasets in Table 1.

5.2.1 Results of Solution Sizes. The results of solution sizes obtained by the SaPSO algorithm, SFS, SBS, LRS21, LRS32, SFFS, SBFS, GA, standard PSO, original PSO, DE, and SaDE on the 12 training sets are shown in Tables 7 and 8, in terms of mean values (Mean) of solution sizes, standard deviations (Std), and reduction rate (%).

In these Tables, "SZ" represents the solution size, i.e., the number of selected features; a statistically significance test, i.e., *T*-test, with a confidence level of 95% is used; "*T*–*E*" describes whether existing statistically significant differences between the SaPSO algorithm and its counterparts,

Table 7. Solution Sizes of Non-EC Methods and SaPSO on Training Sets

DS		SFS		SBS		LRS21		LRS32		SFFS		SBFS		SaPSO	
		Mean ± Std	%	Mean ± Std	%	Mean ± Std	%	Mean ± Std	%	Mean ± Std	%	Mean ± Std	%	Mean ± Std	%
DS1	SZ	4.9 ± 1.4	98.3	298.3 ± 1.4	0.8	**4.8** ± 1.1	98.4	5.6 ± 1.3	98.1	5.7 ± 1.6	98.1	298.4 ± 0.9	0.8	77.1 ± 10.1	74.3
	T–E,P	–,<0.001	↓24.1	+,<0.001	↑73.4	–,<0.001	↓24.0	–,<0.001	↓23.7	–,<0.001	↓23.7	+,<0.001	↑73.5		
DS2	SZ	14.8 ± 5.0	94.2	254.4 ± 0.9	0.6	14.0 ± 4.0	94.5	14.1 ± 3.8	94.4	**11.8** ± 2.4	95.3	253.9 ± 0.3	0.8	107.5 ± 4.8	58
	T–E,P	–,<0.001	↓36.2	+,<0.001	↑57.4	–,<0.001	↓36.5	–,<0.001	↓36.4	–,<0.001	↓37.3	+,<0.001	↑57.1		
DS3	SZ	16.0 ± 3.9	97.4	615.2 ± 0.9	0.2	15.9 ± 3.0	97.4	16.3 ± 3.5	97.3	**13.6** ± 2.0	97.7	614.8 ± 0.3	0.3	159.3 ± 8.1	74.1
	T–E,P	–,<0.001	↓23.2	+,<0.001	↑73.9	–,<0.001	↓23.2	–,<0.001	↓23.1	–,<0.001	↓23.6	+,<0.001	↑73.8		
DS4	SZ	10.2 ± 2.5	98.4	646.1 ± 0.9	0.4	10.1 ± 2.7	98.4	**9.5** ± 2.4	98.5	10.8 ± 1.6	98.3	646.5 ± 0.6	0.3	147.4 ± 14.9	77.2
	T–E,P	–,<0.001	↓21.1	+,<0.001	↑76.8	–,<0.001	↓21.1	–,<0.001	↓21.2	–,<0.001	↓21.0	+,<0.001	↑76.9		
DS5	SZ	8.5 ± 2.0	98.4	559.2 ± 1.1	0.3	**8.3** ± 1.6	98.5	8.7 ± 2.1	98.4	9.6 ± 2.3	98.2	558.6 ± 0.6	0.4	122.9 ± 15.6	78
	T–E,P	–,<0.001	↓20.4	+,<0.001	↑77.7	–,<0.001	↓20.4	–,<0.001	↓20.3	–,<0.001	↓20.2	+,<0.001	↑77.6		
DS6	SZ	7.7 ± 2.3	98.6	559.3 ± 1.0	0.3	**7.1** ± 1.6	98.7	7.1 ± 2.1	98.7	9.0 ± 2.3	98.3	558.9 ± 0.3	0.3	123.0 ± 15.8	78
	T–E,P	–,<0.001	↓20.5	+,<0.001	↑77.7	–,<0.001	↓20.6	–,<0.001	↓20.6	–,<0.001	↓20.3	+,<0.001	↑77.6		
DS7	SZ	**1.8** ± 0.4	99.6	521.0 ± 0.0	0.1	2.0 ± 0.0	99.6	2.0 ± 0.0	99.6	2.0 ± 0.4	99.6	519.8 ± 0.6	0.4	3.4 ± 4.1	99.3
	T–E,P	–,0.0419	↓0.3	+,<0.001	↑99.1	=,0.072	↓0.2	=,0.072	↓0.2	=,0.073	↓0.2	+,<0.001	↑98.9		
DS8	SZ	**3.1** ± 2.4	99.3	498.1 ± 0.9	0.3	5.7 ± 2.7	98.8	3.7 ± 2.5	99.2	6.5 ± 2.0	98.6	497.8 ± 0.4	0.4	111.8 ± 10.8	77.6
	T–E,P	–,<0.001	↓21.7	+,<0.001	↑77.2	–,<0.001	↓21.2	–,<0.001	↓21.6	–,<0.001	↓21.0	+,<0.001	↑77.1		
DS9	SZ	15.3 ± 2.5	76	62.4 ± 0.8	2.4	15.4 ± 3.1	75.8	14.9 ± 2.6	76.6	**9.7** ± 2.5	84.7	61.7 ± 0.6	3.5	32.8 ± 1.7	48.6
	T–E,P	–,<0.001	↓27.4	+,<0.001	↑46.2	–,<0.001	↓27.2	–,<0.001	↓28.0	–,<0.001	↓36.0	+,<0.001	↑45.0		
DS10	SZ	4.1 ± 1.6	93.1	58.1 ± 0.9	3	4.5 ± 1.5	92.3	**3.9** ± 1.2	93.3	4.4 ± 1.6	92.6	57.7 ± 0.5	3.7	18.2 ± 2.5	69.6
	T–E,P	–,<0.001	↓23.5	+,<0.001	↑66.6	–,<0.001	↓22.7	–,<0.001	↓23.7	–,<0.001	↓23.0	+,<0.001	↑65.8		
DS11	SZ	3.4 ± 1.0	88.5	28.4 ± 0.7	5.2	3.5 ± 1.0	88.2	3.3 ± 1.1	89	**3.1** ± 1.0	89.5	27.8 ± 0.3	7.1	9.9 ± 2.3	67
	T–E,P	–,<0.001	↓21.5	+,<0.001	↑61.7	–,<0.001	↓21.2	–,<0.001	↓22.0	–,<0.001	↓22.5	+,<0.001	↑59.8		
DS12	SZ	3.1 ± 1.4	94.4	54.1 ± 0.8	3.2	**2.9** ± 1.0	94.7	3.3 ± 1.4	94.1	3.5 ± 1.7	93.7	53.8 ± 0.4	3.8	11.5 ± 2.5	79.4
	T–E,P	–,<0.001	↓15.0	+,<0.001	↑76.1	–,<0.001	↓15.3	–,<0.001	↓14.7	–,<0.001	↓14.3	+,<0.001	↑75.5		

Note: DS represents datasets. DS1 to DS12 represent the fourth to the fifteenth datasets in Table 1. The real number in the *T–E* row means the percentage of the reduced features that is enhanced by the SaPSO algorithm.

and "*P*" represents the *p*-values obtained in the *T*-tests; "+"/"–" means the result of the SaPSO algorithm is better/worse than the corresponding algorithm with a significant difference, and "=" means there is no significant difference between the SaPSO algorithm and the corresponding algorithm; ↑/↓ indicates the reduction rate, which is increased/decreased by the SaPSO algorithm comparing to the corresponding algorithm. The best results in terms of solution size are typed in bold.

Because the results of LRS31 and LRS32 are almost the same and space limitation, only the results of LRS32 are presented. It is shown in Table 7 that SFS, LRS21, LRS32, and SFFS can obtain smaller solution sizes than SaPSO on almost all the training sets while the solution sizes obtained by SBS and SBFS are much bigger than SaPSO. Generally speaking, SFS, LRS21, LRS32, SFFS can reduce more than 90% features while SBF and SBFS can only reduce less than 10% features. The bigger the number of the features, the smaller the reduction ratio of SBF and SBFS. It means that the "top down" search techniques are not suitable for solving the large-scale feature selection problems. Moreover, Table 7 indicates that non-EC methods are comparatively better on solution sizes. Maybe, this advantage can be used to develop the EC techniques for feature selection in the future.

Table 8. Solution Sizes of EC Methods on Training Sets

DS		GA		Original PSO		Standard PSO		DE		SaDE		SaPSO	
		Mean ± Std	%	Mean ± Std	%	Mean ± Std	%	Mean ± Std	%	Mean ± Std	%	Mean ± Std	%
DS1	SZ	124.6 ± 13.8	58.6	123.7 ± 10.1	58.8	121.0 ± 8.0	59.8	121.7 ± 8.4	59.5	114.0 ± 7.7	62.1	**77.1** ± 10.1	74.3
	T–E,P	+,<0.001	↑15.7	+,<0.001	↑15.4	+,<0.001	↑14.5	+,<0.001	↑14.8	+,<0.001	↑12.2		
DS2	SZ	188.1 ± 22.5	26.5	150.0 ± 13.9	41.3	165.3 ± 15.8	35.4	110.9 ± 17.2	56.6	108.3 ± 7.2	57.6	**107.5** ± 4.8	58
	T–E,P	+,<0.001	↑31.4	+,<0.001	↑16.6	+,<0.001	↑22.5	=,0.3	↑1.3	=,0.61	↑0.3		
DS3	SZ	339.3 ± 51.6	45	262.3 ± 20.8	57.4	286.9 ± 36.8	53.4	244.5 ± 11.4	60.3	233.6 ± 9.9	62.1	**159.3** ± 8.1	74.1
	T–E,P	+,<0.001	↑29.1	+,<0.001	↑16.7	+,<0.001	↑20.6	+,<0.001	↑13.8	+,<0.001	↑12.0		
DS4	SZ	333.8 ± 48.5	48.5	294.3 ± 24.6	54.6	299.9 ± 24.0	53.7	252.7 ± 12.7	61	249.3 ± 11.2	61.5	**147.4** ± 14.9	77.2
	T–E,P	+,<0.001	↑28.7	+,<0.001	↑22.6	+,<0.001	↑23.5	+,<0.001	↑16.2	+,<0.001	↑15.7		
DS5	SZ	324.9 ± 54.8	42	273.5 ± 24.4	51.2	286.8 ± 31.2	48.8	227.0 ± 11.6	59.5	220.2 ± 11.7	60.7	**122.9** ± 15.6	78
	T–E,P	+,<0.001	↑36.0	+,<0.001	↑26.8	+,<0.001	↑29.2	+,<0.001	↑18.5	+,<0.001	↑17.3		
DS6	SZ	342.1 ± 49.9	39	289.3 ± 29.2	48.4	308.9 ± 34.7	44.9	224.1 ± 11.6	60	216.5 ± 10.8	61.3	**123.0** ± 15.8	78
	T–E,P	+,<0.001	↑39.0	+,<0.001	↑29.6	+,<0.001	↑33.1	+,<0.001	↑18.0	+,<0.001	↑16.6		
DS7	SZ	225.4 ± 45.0	56.8	85.4 ± 8.8	83.6	23.0 ± 6.7	95.5	171.2 ± 2.9	67.1	155.7 ± 4.5	70.1	**3.4** ± 4.1	99.3
	T–E,P	+,<0.001	↑42.5	+,<0.001	↑15.7	+,<0.001	↑3.7	+,<0.001	↑32.1	+,<0.001	↑29.1		
DS8	SZ	290.6 ± 54.1	41.8	228.9 ± 20.6	54.2	255.9 ± 38.4	48.8	201.0 ± 11.6	59.7	185.8 ± 10.5	62.8	**111.8** ± 10.8	77.6
	T–E,P	+,<0.001	↑35.7	+,<0.001	↑23.4	+,<0.001	↑28.8	+,<0.001	↑17.8	+,<0.001	↑14.8		
DS9	SZ	42.2 ± 4.3	34	39.9 ± 2.7	37.6	42.8 ± 3.3	33.1	36.4 ± 7.2	43	34.5 ± 3.1	46	**32.8** ± 1.7	48.6
	T–E,P	+,<0.001	↑14.6	+,<0.001	↑10.9	+,<0.001	↑15.5	+,0.0119	↑5.6	+,0.0115	↑2.5		
DS10	SZ	23.1 ± 3.7	61.4	21.5 ± 3.7	64.1	23.4 ± 3.0	60.8	21.4 ± 2.7	64.2	20.2 ± 3.2	66.2	**18.2** ± 2.5	69.6
	T–E,P	+,<0.001	↑8.2	+,<0.001	↑5.5	+,<0.001	↑8.7	+,<0.001	↑5.3	+,0.00926	↑3.4		
DS11	SZ	13.4 ± 3.6	55.1	13.4 ± 2.3	55.3	15.0 ± 3.4	50	11.3 ± 2.1	62.1	11.3 ± 1.8	62.3	**9.9** ± 2.3	67
	T–E,P	+,<0.001	↑11.8	+,<0.001	↑11.6	+,<0.001	↑17.0	+,0.0168	↑4.8	+,0.0112	↑4.6		
DS12	SZ	17.0 ± 4.5	69.5	18.3 ± 3.6	67.2	17.2 ± 3.9	69.2	19.6 ± 3.8	65	18.6 ± 4.0	66.7	**11.5** ± 2.5	79.4
	T–E,P	+,<0.001	↑9.8	+,<0.001	↑12.1	+,<0.001	↑10.1	+,<0.001	↑14.4	+,<0.001	↑12.6		

It can be observed from Table 8 that the solution sizes obtained by the SaPSO algorithm on almost all the 12 datasets are smaller than those obtained by GA, standard PSO, original PSO, DE, and SaDE with statistical significant difference. Moreover, on each dataset, there is a statistically significant difference between the SaPSO algorithm and the other algorithms. As seen from Table 8 that, for one dataset (DS7), the SaPSO algorithm can reduce more than 90% of features; for seven datasets (DS1, DS3, DS4, DS5, DS6, DS8, DS12), the feature reduction rate is 70% to 80%; for two datasets (DS10, DS11), the reduced features take up 60% to 70%; and for the remaining two datasets (DS2, DS9), the feature reduction rate is 50% to 60% and 40% to 50%, respectively. Therefore, the SaPSO algorithm can reduce 70% to 80% of features in most cases. By comparing the reduction rate between SaPSO and the other EC methods, it can be observed that the ratios are enhanced by 10% to 30% on most datasets.

5.2.2 Results of Classification Accuracy on Training Sets and Test Sets. The training classification accuracies and test classification accuracies obtained by the non-EC and EC methods on the 12 training datasets and the corresponding test datasets are presented in Tables 9–12 in terms of mean values and standard deviations of classification accuracies. The best mean value on each dataset is typed in bold. In these tables, "CA" represents the classification accuracy.

It is shown in Table 9 that SaPSO can get better classification accuracies than all of SFS, SBS, LRS21, LRS32, SFFS, and SBFS on 11 training sets with statistical significant difference. SaPSO and

Table 9. Classification Accuracies of Non-EC Methods and SaPSO on Training Sets

DS		SFS	SBS	LRS21	LRS32	SFFS	SBFS	SaPSO
		Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std
DS1	CA	0.8929 ± 0.0082	0.8147 ± 0.0113	0.8924 ± 0.0078	0.8965 ± 0.0077	0.9002 ± 0.0051	0.8170 ± 0.0108	**0.9159** ± 0.0026
	T-E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS2	CA	0.6832 ± 0.0492	0.8471 ± 0.0041	0.6772 ± 0.0511	0.6822 ± 0.0453	0.6396 ± 0.0350	0.8475 ± 0.0034	**0.9031** ± 0.0049
	T-E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS3	CA	0.7481 ± 0.0392	0.7615 ± 0.0041	0.7460 ± 0.0277	0.7475 ± 0.0361	0.6949 ± 0.0363	0.7599 ± 0.0033	**0.8899** ± 0.0025
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS4	CA	0.8449 ± 0.0208	0.9493 ± 0.0017	0.8453 ± 0.0209	0.8373 ± 0.0193	0.9194 ± 0.0150	0.9478 ± 0.0030	**0.9831** ± 0.0015
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS5	CA	0.9244 ± 0.0111	0.9192 ± 0.0022	0.9261 ± 0.0081	0.9266 ± 0.0120	0.9282 ± 0.0074	0.9185 ± 0.0020	**0.9693** ± 0.0043
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS6	CA	0.9200 ± 0.0110	0.9134 ± 0.0031	0.9196 ± 0.0091	0.9170 ± 0.0114	0.9268 ± 0.0119	0.9137 ± 0.0031	**0.9745** ± 0.0048
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS7	CA	**1.0000** ± 0.0000	**1.0000** ± 0.0000	**1.0000** ± 0.0000	**1.0000** ± 0.0000	**1.0000** ± 0.0000	**1.0000** ± 0.0000	**1.0000** ± 0.0000
	T–E,P	=,NaN	=,NaN	=,NaN	=,NaN	=,NaN	=,NaN	
DS8	CA	0.6595 ± 0.0858	0.7155 ± 0.0072	0.7392 ± 0.1062	0.6803 ± 0.0804	0.8160 ± 0.0399	0.7145 ± 0.0060	**0.8722** ± 0.0050
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS9	CA	0.9482 ± 0.0108	0.9675 ± 0.0021	0.9490 ± 0.0116	0.9470 ± 0.0121	0.8185 ± 0.0737	0.9676 ± 0.0028	**0.9863** ± 0.0009
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS10	CA	0.8476 ± 0.0230	0.8095 ± 0.0093	0.8492 ± 0.0190	0.8444 ± 0.0181	0.8540 ± 0.0243	0.8080 ± 0.0111	**0.9537** ± 0.0086
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS11	CA	0.9419 ± 0.0052	0.9474 ± 0.0023	0.9452 ± 0.0046	0.9449 ± 0.0051	0.9442 ± 0.0060	0.9466 ± 0.0024	**0.9682** ± 0.0033
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS12	CA	0.7300 ± 0.0657	0.6440 ± 0.0223	0.7167 ± 0.0460	0.7419 ± 0.0514	0.7470 ± 0.0481	0.6315 ± 0.0253	**0.9310** ± 0.0192
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	

all the other non-EC algorithms obtain the same results on one training set. The performance of SaPSO is better than the non-EC algorithms in terms of training accuracy. In fact, SFS, LRS21, LRS32, and SFFS belong to the "bottom-up" search technique while SBS and SBFS belong to the "top-down" search technique. The "bottom-up" search technique adds better features to an empty set while the "top-down" search technique removes worse features from a complete set. It means the classification accuracy increases as either increasing the solution size from zero or decreasing the solution size from the total number of features. Nevertheless, many local optima will be experienced during this process. However, both "bottom-up" and "top-down" techniques can only obtain few solutions because they are heuristic search techniques. They add or remove a feature separately and greedily. In fact, the individually good features may not be included in the optimal set, and vice versa, the individually bad features may not be excluded from the optimal set. Hence, the greedy search techniques easily get trapped in local optima from different directions, and the "bottom-up" search techniques always get the solutions with small sizes while the "top-down" search techniques always get the solutions with big solution sizes. Starting the search with one or multiple medium solution sizes, and searching along different directions may be a better search technique for feature selection problems. This may be a reason why SaPSO performs more efficient than the other algorithms.

Table 10 shows that the SaPSO algorithm performs better than the other EC methods in terms of classification accuracy with statistical significant difference on most of the training sets. Besides,

Table 10. Classification Accuracies of EC Mothods on Training Sets

DS		GA	Original PSO	Standard PSO	DE	SaDE	SaPSO
		Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std
DS1	CA	0.9078 ± 0.0035	0.9107 ± 0.0022	0.9127 ± 0.0040	0.8988 ± 0.0024	0.9029 ± 0.0023	**0.9159** ± 0.0026
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS2	CA	0.8702 ± 0.0057	0.8753 ± 0.0041	0.8809 ± 0.0060	0.8464 ± 0.0034	0.8510 ± 0.0044	**0.9031** ± 0.0049
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS3	CA	0.8205 ± 0.0076	0.8446 ± 0.0089	0.8495 ± 0.0138	0.7941 ± 0.0035	0.8059 ± 0.0047	**0.8899** ± 0.0025
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS4	CA	0.9688 ± 0.0029	0.9705 ± 0.0034	0.9734 ± 0.0041	0.9585 ± 0.0017	0.9610 ± 0.0017	**0.9831** ± 0.0015
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS5	CA	0.9467 ± 0.0032	0.9510 ± 0.0027	0.9529 ± 0.0024	0.9351 ± 0.0020	0.9391 ± 0.0020	**0.9693** ± 0.0043
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS6	CA	0.9379 ± 0.0056	0.9429 ± 0.0056	0.9476 ± 0.0064	0.9262 ± 0.0028	0.9296 ± 0.0025	**0.9745** ± 0.0048
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS7	CA	**1.0000** ± 0.0000	**1.0000** ± 0.0000	**1.0000** ± 0.0000	**1.0000** ± 0.0000	**1.0000** ± 0.0000	**1.0000** ± 0.0000
	T–E,P	=,NaN	=,NaN	=,NaN	=,NaN	=,NaN	
DS8	CA	0.7837 ± 0.0116	0.8157 ± 0.0110	0.8221 ± 0.0169	0.7608 ± 0.0082	0.7724 ± 0.0046	**0.8722** ± 0.0050
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS9	CA	0.9860 ± 0.0018	0.9838 ± 0.0018	0.9849 ± 0.0019	0.9693 ± 0.0023	0.9748 ± 0.0016	**0.9863** ± 0.0009
	T–E,P	=,0.42	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS10	CA	0.9447 ± 0.0108	0.9467 ± 0.0114	0.9461 ± 0.0146	0.9024 ± 0.0055	0.9124 ± 0.0062	**0.9537** ± 0.0086
	T–E,P	+,<0.001	+0.00962	+,0.0178	+,<0.001	+,<0.001	
DS11	CA	0.9657 ± 0.0035	0.9670 ± 0.0037	0.9665 ± 0.0042	0.9644 ± 0.0013	0.9658 ± 0.0014	**0.9682** ± 0.0033
	T–E,P	+,0.00611	=,0.19	=,0.087	+,<0.001	+,<0.001	
DS12	CA	0.9355 ± 0.0289	0.9431 ± 0.0232	**0.9484** ± 0.0303	0.8718 ± 0.0245	0.8855 ± 0.0211	0.9310 ± 0.0192
	T–E,P	=,0.48	−,0.0319	−,0.0106	+,<0.001	+,<0.001	

it is observed from Table 10 that the robustness of SaPSO is good since the standard deviations of SaPSO on most datasets are small. The results indicate that the self-adaptive mechanism and the efficient CSGSs in the strategy pool make SaPSO efficient in solving feature selection problems.

As can be seen from Table 11, for most test sets SaPSO performs better than SFS, SBS, LRS21, LRS32, SFFS, and SBFS respectively. Occasionally, SaPSO performs worse that SBS and SBFS. It can be seen from Table 12 that the classification accuracy of the SaPSO algorithm is higher than that of GA on five test sets, similar to that of GA on seven test sets; the classification accuracy of the SaPSO algorithm is higher than that of original PSO on seven test sets, similar to that of original PSO on five test sets. Similarly, the classification accuracy of the SaPSO algorithm is higher than that of standard PSO on five test sets, similar to that of standard PSO on seven test sets; the classification accuracy of the SaPSO algorithm is higher than that of DE on seven test sets, similar to that of DE on five test sets; the classification accuracy of the SaPSO algorithm is higher than that of SaDE on nine test sets, similar to that of SaDE on three test sets. The standard deviations of SaPSO on the test sets are small. Overall, the performance of the SaPSO is better than the other EC methods on the test sets.

5.2.3 Convergence Performance of the EC Methods on the Training Sets. The convergence performance of the six EC methods on training sets is shown in Figure 3. (Note: the scales along Y axis are different for different graphs, and the experimental results in Figure 3 are statistical, i.e., for

Table 11. Classification Accuracies of Non-EC Methods and SaPSO on Test Sets

DS		SFS	SBS	LRS21	LRS32	SFFS	SBFS	SaPSO
		Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std
DS1	CA	0.8447 ± 0.0133	0.7301 ± 0.0313	**0.8471** ± 0.0150	0.8446 ± 0.0169	0.8165 ± 0.0211	0.7319 ± 0.0333	0.8448 ± 0.0157
	T–E,P	=,0.98	+,<0.001	=,0.56	=,0.96	+,<0.001	+,<0.001	
DS2	CA	0.5478 ± 0.0663	**0.7994** ± 0.0167	0.5495 ± 0.0613	0.5542 ± 0.0587	0.4355 ± 0.0548	0.7991 ± 0.0188	0.7795 ± 0.0216
	T–E,P	+,<0.001	−,<0.001	+,<0.001	+,<0.001	+,<0.001	−,<0.001	
DS3	CA	0.6568 ± 0.0498	0.7144 ± 0.0162	0.6562 ± 0.0482	0.6586 ± 0.0505	0.4620 ± 0.0578	0.7139 ± 0.0167	**0.8106** ± 0.0167
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS4	CA	0.7666 ± 0.0261	0.9308 ± 0.0090	0.7598 ± 0.0242	0.7609 ± 0.0250	0.8444 ± 0.0374	0.9308 ± 0.0087	**0.9388** ± 0.0092
	T–E,P	+,<0.001	+,0.00121	+,<0.001	+,<0.001	+,<0.001	+,0.00102	
DS5	CA	0.8520 ± 0.0220	0.8479 ± 0.0130	0.8531 ± 0.0185	0.8516 ± 0.0219	0.7764 ± 0.0253	0.8449 ± 0.0127	**0.8814** ± 0.0121
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS6	CA	0.8702 ± 0.0204	0.8545 ± 0.0144	0.8779 ± 0.0165	0.8720 ± 0.0267	0.8017 ± 0.0397	0.8554 ± 0.0162	**0.9140** ± 0.0212
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS7	CA	0.9981 ± 0.0032	**0.9996** ± 0.0011	0.9974 ± 0.0038	0.9989 ± 0.0021	0.9979 ± 0.0029	0.9993 ± 0.0022	0.9986 ± 0.0026
	T–E,P	=,0.51	=,0.06	=,0.16	=,0.62	=,0.33	=,0.27	
DS8	CA	0.5400 ± 0.0728	0.6173 ± 0.0291	0.6029 ± 0.1033	0.5797 ± 0.0779	0.6482 ± 0.0673	0.6141 ± 0.0325	**0.6863** ± 0.0351
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,0.00865	+,<0.001	
DS9	CA	0.8833 ± 0.0216	0.9317 ± 0.0084	0.8843 ± 0.0244	0.8831 ± 0.0287	0.7248 ± 0.0789	0.9319 ± 0.0111	**0.9364** ± 0.0107
	T–E,P	+,<0.001	=,0.064	+,<0.001	+,<0.001	+,<0.001	=,0.12	
DS10	CA	0.6634 ± 0.0498	0.6513 ± 0.0624	0.6608 ± 0.0581	0.6456 ± 0.0391	0.6649 ± 0.0737	0.6494 ± 0.0595	**0.7005** ± 0.0511
	T–E,P	+,0.00608	+,0.00149	+,0.00677	+,<0.001	+,0.0343	+,<0.001	
DS11	CA	0.8622 ± 0.0248	0.8887 ± 0.0148	0.8774 ± 0.0293	0.8719 ± 0.0300	0.8881 ± 0.0513	0.8881 ± 0.0141	**0.9099** ± 0.0135
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+0.0312	+,<0.001	
DS12	CA	0.4870 ± 0.2037	**0.6204** ± 0.0990	0.4417 ± 0.1694	0.4139 ± 0.1892	0.4269 ± 0.1658	0.5926 ± 0.1085	0.5102 ± 0.1442
	T–E,P	=,0.61	−,0.00113	=,0.097	+,0.0308	+,0.0424	−,0.0155	

each algorithm, the mean values of the results of its 30 independent runs were firstly calculated then plotted.) By comparing the convergence curves of the algorithms, it is observed that most algorithms have similar performance at the initial stage because the same initialization method is used for all the algorithms. However, at the early stage, the SaPSO algorithm converges faster than GA, original PSO, standard PSO, DE, and SaDE to the solutions which with both small sizes and higher classification accuracies on most datasets. At the later stages, although the convergence performance of all the algorithms decreases significantly, the classification accuracy obtained by the SaPSO algorithm become higher and higher while the classification accuracy of other algorithms changes very little on most datasets, meanwhile the solution size obtained by SaPSO is much smaller than that of the other algorithms. Thus, the SaPSO algorithm has a better diversity property instead of stagnating into a local optimum. Besides, it is observed that there exist obvious distinctions between SaPSO and the other algorithms.

For small-scale training sets, i.e., the last four datasets, although the difference of classification accuracy among the six algorithms is not obvious, SaPSO can obtain the solutions that with much smaller sizes. For most large-scale feature selection problems, SaPSO can obtain the solutions with a high classification accuracy and a small size. The advantage of the SaPSO algorithm becomes increasingly prominent as the dimensionality of the datasets increases. Perhaps this is because there are few local optima in the small solution space of a feature selection problem. Hence, this kind of problem is not very difficult to solve by a common EC method. Thus, the SaPSO algorithm,

Table 12. Classification Accuracies of EC Methods on Test Sets

DS		GA	Original PSO	Standard PSO	DE	SaDE	SaPSO
		Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std	Mean ± Std
DS1	CA	**0.8513** ± 0.0151	0.8449 ± 0.0131	0.8507 ± 0.0126	0.8506 ± 0.0133	0.8470 ± 0.0117	0.8448 ± 0.0157
	T–E,P	=,0.11	=,0.98	=,0.11	=,0.13	=,0.54	
DS2	CA	0.7872 ± 0.0201	0.7821 ± 0.0272	**0.7892** ± 0.0254	0.7690 ± 0.0207	0.7685 ± 0.0183	0.7795 ± 0.0216
	T–E,P	=,0.16	=,0.68	=,0.12	=,0.059	+,0.0377	
DS3	CA	0.7487 ± 0.0258	0.7875 ± 0.0232	0.7777 ± 0.0202	0.7375 ± 0.0214	0.7534 ± 0.0223	**0.8106** ± 0.0167
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS4	CA	0.9356 ± 0.0091	0.9327 ± 0.0105	0.9348 ± 0.0125	0.9300 ± 0.0106	0.9334 ± 0.0104	**0.9388** ± 0.0092
	T–E,P	=,0.18	+,0.02	=,0.16	+0.00112	+,0.0375	
DS5	CA	0.8530 ± 0.0159	0.8585 ± 0.0146	0.8599 ± 0.0152	0.8496 ± 0.0199	0.8504 ± 0.0172	**0.8814** ± 0.0121
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS6	CA	0.8679 ± 0.0146	0.8727 ± 0.0156	0.8713 ± 0.0174	0.8630 ± 0.0212	0.8673 ± 0.0243	**0.9140** ± 0.0212
	T–E,P	+,<0.001	+,<0.001	+,<0.001	+,<0.001	+,<0.001	
DS7	CA	0.9962 ± 0.0053	0.9901 ± 0.0064	0.9952 ± 0.0043	0.9908 ± 0.0082	0.9919 ± 0.0103	**0.9986** ± 0.0026
	T–E,P	+,0.0314	+,<0.001	+,<0.001	+,<0.001	+,0.00155	
DS8	CA	0.6421 ± 0.0291	0.6598 ± 0.0352	0.6503 ± 0.0402	0.6400 ± 0.0357	0.6387 ± 0.0346	**0.6863** ± 0.0351
	T–E,P	+,<0.001	+,0.00498	+,<0.001	+,<0.001	+,<0.001	
DS9	CA	0.9312 ± 0.0097	0.9278 ± 0.0098	0.9324 ± 0.0078	0.9187 ± 0.0135	0.9247 ± 0.0136	**0.9364** ± 0.0107
	T–E,P	=,0.053	+,0.00195	=,0.1	+,<0.001	+,<0.001	
DS10	CA	0.6908 ± 0.0556	0.6972 ± 0.0462	0.6804 ± 0.0423	0.6779 ± 0.0442	0.6841 ± 0.0553	**0.7005** ± 0.0511
	T–E,P	=,0.48	=,0.79	=,0.1	=,0.072	=,0.24	
DS11	CA	0.9125 ± 0.0232	**0.9162** ± 0.0191	0.9092 ± 0.0213	0.9021 ± 0.0200	0.9021 ± 0.0127	0.9099 ± 0.0135
	T–E,P	=,0.6	=,0.15	=,0.88	=,0.083	+,0.0248	
DS12	CA	0.4750 ± 0.1602	0.4565 ± 0.1211	0.4565 ± 0.1348	0.4852 ± 0.1523	0.4481 ± 0.1446	**0.5102** ± 0.1442
	T–E,P	=,0.37	=,0.12	=,0.14	=,0.52	=,0.1	

GA, original PSO, standard PSO, DE, and SaDE have similar performance on small-scale feature selection problems. However, with the dimensionality of the datasets increasing, an increasing number of local optima appear in the increasingly huge solution space. Therefore, the problem becomes increasingly difficult to solve, and a technique such as the SaPSO algorithm may be more useful to solve feature selection problems with a large-scale dimensionality.

We have already got a conclusion from Table 8 that more than 70% of the features are irrelevant or redundant in most cases. If the irrelevant or redundant features are added into a feature set, many solutions with different sizes but with the same accuracy will be generated in the solution space. For example, suppose that there are three relevant features and seven irrelevant features. The seven irrelevant features generate more than 1,000 redundant solutions or local optima for the three relevant features. Therefore, in most cases, optima exist in a small solution space, while the space is separated or surrounded by many local optima that occupy huge space. Therefore, it is observed that the six algorithms find many solutions that have different solution sizes but the same classification accuracy. Regardless of the classification accuracy, the solution size obtained by the SaPSO algorithm is obviously smaller than that of the other algorithms. It is because the SaPSO algorithm not only has a better exploration property but also a better exploitation property, which is due to the multiple efficient CSGSs, and they are used adaptively in the SaPSO algorithm. Therefore, the SaPSO algorithm is an efficient technique not only for small-scale feature selection

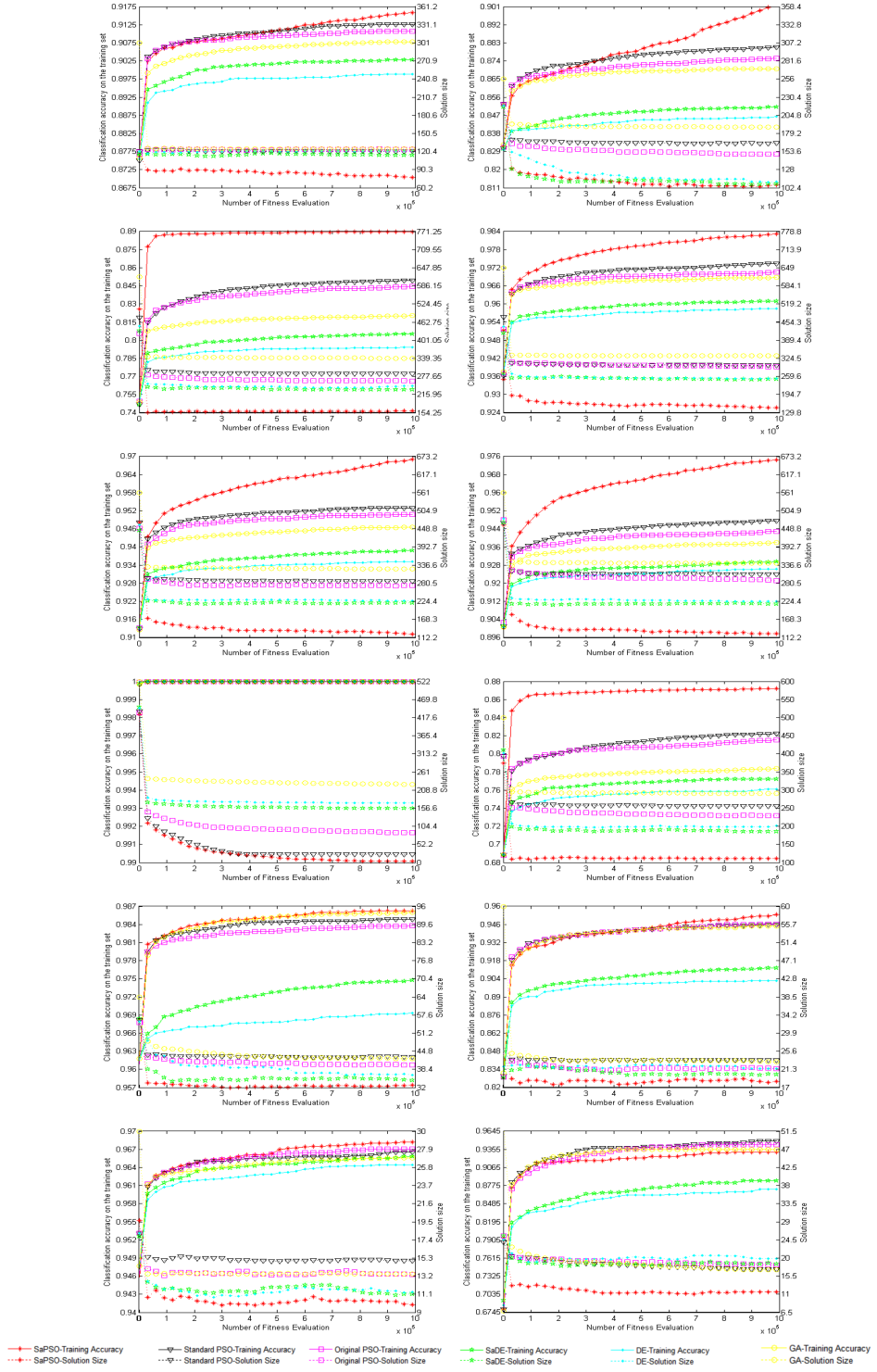

Fig. 3. Convergence curves on DS1–DS12.

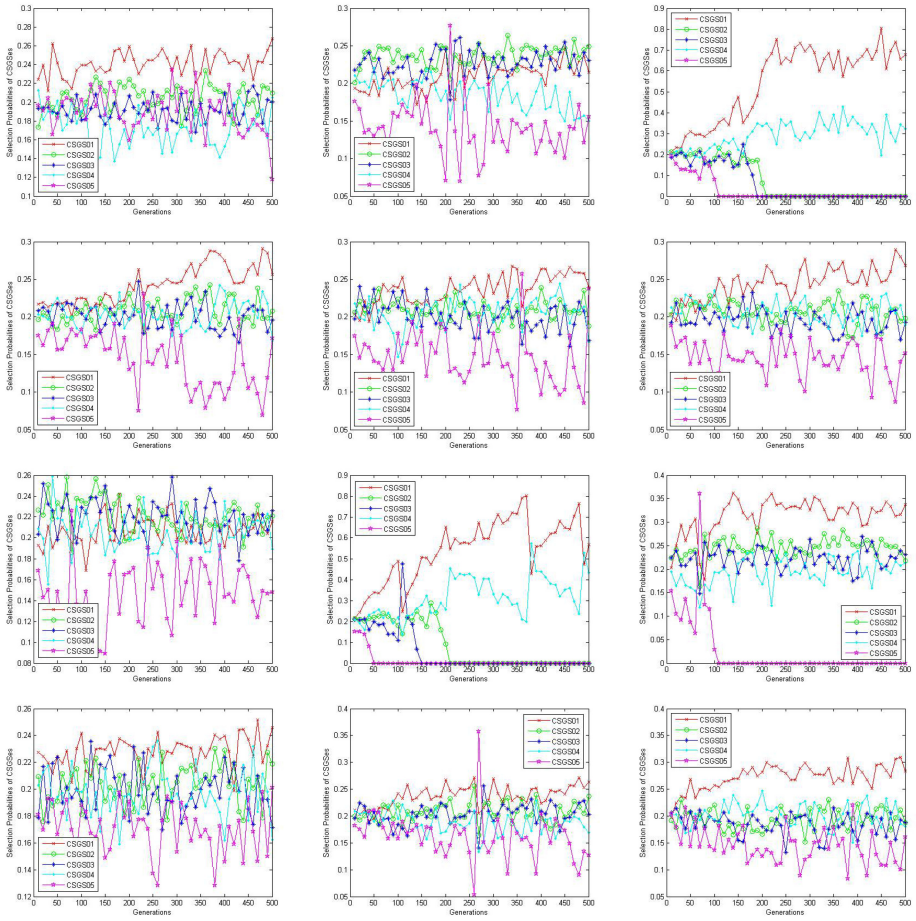

Fig. 4. Changing curves of strategy selection probabilities on DS1–DS12.

problems but also for large-scale feature selection problems. Particularly on large-scale feature selection problems, its advantages become more and more prominent.

By comparing the classification accuracy and solution size of the other algorithms, we observed that, in most cases, the classification accuracy of GA, original PSO and standard PSO is higher than that of DE and SaDE. Unfortunately, the solution size of GA, original PSO, and standard PSO is worse than that of DE and SaDE. Our goal is to obtain the solutions with not only small solution size but also high classification accuracy. However, most existing algorithms are good at only one objective. By comparing SaPSO with SaDE, it is obviously observed that SaPSO is better than SaDE on most datasets. The results indicate that the SaPSO algorithm is an effective technique to solve feature selection problems, particularly large-scale feature selection problems. This is because effective CSGSs are employed in SaPSO, and it indicates that the strategy selecting method is useful for strategy pool design.

5.2.4 Changing Tendency of Selection Probabilities of the CSGSs. To investigate the effectiveness of the self-adaptive mechanism in the SaPSO algorithm, the changing curves of the strategy selection probabilities are plotted as shown in Figure 4. Each subgraph means the changing tendency of the selection probabilities of the CSGSs on each dataset. In each subgraph, the x axis

represents the number of generations, the y axis represents the value of the selection probability, and one curve means the changing selection probability of one CSGS. Taking the third subgraph, for example, the red curve represents the selection probability of CSGS01. At the beginning, it is selected at the probability of about 0.2. After 200 generations, it is selected at the probability of more than 0.5, and the selection probability sometimes arrives at 0.8. Similarly, the blue curve represents the selection probability of CSGS03. At the beginning, it is selected at the probability of 0.2 as well. But it is never selected after 200 generations since its selection probability becomes zero. For observing convenience, only the first 500 generations of the curves are plotted. It is observed from Figure 4 that all the probabilities change during all the evolutionary processes on all datasets. This phenomenon indicates not only that all CSGSs are used but also that the performance of the CSGSs changes at different evolutionary stages. The self-adaptive mechanism can select the most suitable CSGS at different evolution stages. We can see that the probability of the first CSGS is greater than the other CSGSs on some datasets, whereas it is less than most of the others on other datasets. It is because different feature selection problems have different properties. However, the self-adaptive mechanism can select the most suitable CSGS during different evolution stages.

6 CONCLUSIONS

The objectives of this article were to design a self-adaptive algorithm, to carry out theoretical study of designing the strategy pool, and to investigate whether a self-adaptive PSO algorithm can achieve good performance for feature selection, especially for large-scale feature selection. These objectives have been successfully achieved as shown by a set of experiments on datasets of varying difficulties.

In this article, we analyzed the properties of the feature selection. We found that feature selection is different from other common combinatorial optimization problems, i.e., there are more irrelevant features than relevant features in the datasets, and the large number of irrelevant features generate more local optima in the huge solution space. Thus, a feature selection problem becomes more difficult to solve as the dimensionality increases. In addition, large-scale feature selection problems with different datasets often have different properties. In order to solve the large-scale feature selection problems effectively, we proposed an improved AHP approach to solve the basic problems of designing a self-adaptive based EC method. The SaPSO algorithm was proposed using a simple self-adaptive framework. The experiments were conducted on 15 datasets. A total of 12 of the them were further employed to test the performance of SaPSO and its counterparts. The experimental results showed that the SaPSO algorithm could reduce 70% to 80% of the features in most cases. By comparing the SaPSO algorithm with the other four EC methods, the reduction ratios were enhanced by 10% to 30% on most datasets. Moreover, the SaPSO algorithm is better than the other algorithms in the terms of classification accuracy on both the training set and the test set. Furthermore, the performance of the SaPSO algorithm becomes increasingly better than the other algorithms as the dimensionality of the datasets increases.

In the future work, we will implement more state-of-the-art EC based feature selection methods, and do more comparison experiments on more datasets to investigate the performance of SaPSO. In addition, because feature selection can be treated as a two-objective or three-objective problem, we will investigate the properties of the multi-objective feature selection and propose a new multi-objective algorithm based on SaPSO to solve large-scale multi-objective feature selection problems.

REFERENCES

J. Aguaron, M. T. Escobar, and J. M. Moreno-Jimenez. 2016. The precise consistency consensus matrix in a local AHP-group decision making context. *Annals of Operations Research* 245, 12 (2016), 245–259.

A. Al-Ani, A. Alsukker, and R. Khushaba. 2013. Feature subset selection using differential evolution and a wheel based search strategy. *Swarm and Evolutionary Computation* 9 (2013), 15–26.

K. Bache and M. Lichman. 2016. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/index.php.

K. K. Bharti and P. K. Singh. 2016. Opposition chaotic fitness mutation based adaptive inertia weight BPSO for feature selection in text clustering. *Applied Soft Computing* 43 (2016), 20–34.

X. J. Chang, F. P. Nie, Y. Yang, C. Q. Zhang, and H. Huang. 2016. Convex sparse PCA for unsupervised feature learning. *ACM Transactions on Knowledge Discovery from Data* 11, 1 (2016), 16.

M. Dorigo and L. M. Gambardella. 1997. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 53–66.

D. B. Fogel. 1994. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks* 5, 1 (1994), 3–14.

A. S. Ghareb, B. A. Abu, and A. R. Hamdan. 2016. Hybrid feature selection based on enhanced genetic algorithm for text categorization. *Expert Systems with Applications* 49 (2016), 31–47.

I. A. Gheyas and L. S. Smith. 2010. Feature subset selection in large dimensionality domains. *Pattern Recognition* 43, 1 (2010), 5–13.

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

J. H. Holland. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

U. Kamath, J. Compton, R. I. Dogan, K. D. Jong, and A. Shehu. 2012. An evolutionary algorithm approach for feature generation from sequence data and its application to DNA splice site prediction. *IEEE-ACM Transactions on Computational Biology and Bioinformatics* 9, 5 (2012), 1387–1398.

J. Kennedy and R. Eberhart. 1995. Particle swarm optimization. In *IEEE International Conference on Neural Networks*. 1942–1948.

J. R. Koza. 1990. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*, Vol. 34. Stanford University.

C. H. Li, S. X. Yang, and T. T. Nguyen. 2012. A self-learning particle swarm optimizer for global optimization problems. *IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics* 42, 3 (2012), 627–646.

J. Li and D. C. Tao. 2012. On preserving original variables in Bayesian PCA with application to image analysis. *IEEE Transactions on Image Processing* 21, 12 (2012), 4830–4843.

J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar. 2006. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation* 10, 3 (2006), 281–295.

T. Marill and D. Green. 1963. On the effectiveness of receptors in recognition systems. *IEEE Transactions on Information Theory* 9, 1 (1963), 11–17.

V. E. Neagoe and E. C. Neghina. 2016. Feature selection with ant colony optimization and its applications for pattern recognition in space imagery. In *International Conference on Communications*. 101–104.

C. Pornsing, M. S. Sodhi, and B. F. Lamond. 2016. Novel self-adaptive particle swarm optimization methods. *Soft Computing* 20, 9 (2016), 3579–3593.

P. Pudil, J. Novovičová, and J. Kittler. 1994. Floating search methods in feature selection. *Pattern Recognition Letters* 15, 11 (1994), 1119–1125.

A. K. Qin, V. L. Huang, and P. N. Suganthan. 2009. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation* 13, 2 (2009), 398–417.

T. L. Saaty. 1990. How to make a decision: The analytic hierarchy process. *European Journal of Operational Research* 48, 1 (1990), 9–26.

S. D. Stearns. 1976. On selecting features for pattern classifiers. In *3rd International Joint Conference on Pattern Recognition*. 71–75.

R. Storn and K. Price. 1997. Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 11, 4 (1997), 341–359.

A. Stuhlsatz, J. Lippel, and T. Zielke. 2012. Feature extraction with deep neural networks by a generalized discriminant analysis. *IEEE Transactions on Neural Networks and Learning Systems* 23, 4 (2012), 596–608.

T. Sudo, K. Goto, Y. Nojima, and H. Ishibuchi. 2015. Effects of ensemble action selection with different usage of player's memory resource on the evolution of cooperative strategies for iterated prisoner's dilemma game. In *IEEE Congress on Evolutionary Computation*. 1505–1512.

J. L. Tang and H. Liu. 2014. Feature selection for social media data. *ACM Transactions on Knowledge Discovery from Data* 8, 4 (2014), 1–27.

H. Wang, H. Sun, C. H. Li, S. Rahnamayan, and J. S. Pan. 2013. Diversity enhanced particle swarm optimization with neighborhood search. *Information Sciences* 223 (2013), 119–135.

Y. Wang. 2011. Chaotic self-adaptive particle swarm optimization algorithm for dynamic economic dispatch problem with valve-point effects. *Expert Systems with Applications* 38, 11 (2011), 14231–14237.

Y. Wang, B. Li, T. Weise, J. Y. Wang, B. Yuan, and Q. J. Tian. 2011. Self-adaptive learning based particle swarm optimization. *Information Sciences* 181, 20 (2011), 4515–4538.

A. W. Whitney. 1971. A direct method of nonparametric measurement selection. *IEEE Transactions on Computers* 100, 9 (1971), 1100–1103.

Y. Wu, S. C. H. Hoi, T. Mei, and N. H. Yu. 2017. Large-scale online feature selection for ultra-high dimensional sparse data. *ACM Transactions on Knowledge Discovery from Data* 11, 4 (2017), 1–22.

R. Xu, G. C. Anagnostopoulos, and D. C. Wunsch. 2007. Multiclass cancer classification using semisupervised ellipsoid ARTMAP and particle swarm optimization with gene expression data. *IEEE-ACM Transactions on Computational Biology and Bioinformatics* 4, 1 (2007), 65–77.

B. Xue, W. N. Browne, M. J. Zhang, and X. Yao. 2016. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation* 20, 4 (2016), 606–626.

B. Xue, M. J. Zhang, and W. N. Browne. 2013. Particle swarm optimization for feature selection in classification: A multi-objective approach. *IEEE Transactions on Cybernetics* 43, 6 (2013), 1656–1671.

B. Xue, M. J. Zhang, and W. N. Browne. 2014a. Particle swarm optimisation for feature selection in classification: Novel initialisation and updating mechanisms. *Applied Soft Computing* 18 (2014), 261–276.

Y. Xue, J. Jiang, B. Zhao, and T. H. Ma. 2017. A self-adaptive artificial bee colony algorithm based on global best for global optimization. *Soft Computing* 22, 9 (2017), 2935–2952.

Y. Xue, S. M. Zhong, Y. Zhuang, and B. Xu. 2014b. An ensemble algorithm with self-adaptive learning techniques for high-dimensional numerical optimization. *Applied Mathematics and Computation* 231 (2014), 329–346.

H. Q. Yang, M. R. Lyu, and I. King. 2013. Efficient online learning for multitask feature selection. *ACM Transactions on Knowledge Discovery from Data* 7, 2 (2013), 1–27.

J. H. Yang and V. Honavar. 1998. Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems and their Applications* 13, 2 (1998), 44–49.

X. S. Yang. 2008. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press.

K. Yu, X. D. Wu, W. Ding, and J. Pei. 2016. Scalable and accurate online feature selection for big data. *ACM Transactions on Knowledge Discovery from Data* 11, 2 (2016), 39.

Y. Zhang, D. W. Gong, and J. Cheng. 2017a. Multi-objective particle swarm optimization approach for cost-based feature selection in classification. *IEEE-ACM Transactions on Computational Biology and Bioinformatics* 14, 1 (2017), 64–75.

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

Y. Zhang, X. F. Song, and D. W. Gong. 2017b. A return-cost-based binary firefly algorithm for feature selection. *Information Sciences* 418 (2017), 561–574.

Received March 2018; revised April 2019; accepted June 2019

TABLE I
REFERENCES AND FORMULAS OF THE 25 CSGSS

No.	Reference	Formula		Brief information
1	[1]	$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1}$ $v_{id}^{t+1} = w \times v_{id}^{t} + c_1 \times r_{1i} \times (p_{id} - x_{id}^{t}) + c_2 \times r_{2i} \times (p_{gd} - x_{id}^{t})$	(1) (2)	w=0.7298.
2	[1]	$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1}$ $v_{id}^{t+1} = v_{id}^{t} + c_1 \times r_{1i} \times (p_{id} - x_{id}^{t}) + c_2 \times r_{2i} \times (p_{gd} - x_{id}^{t})$	(3) (4)	
3	[1]	$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1}$ $v_{id}^{t+1} = w \times v_{id}^{t} + c_1 \times r_{1i} \times (p_{id} - x_{id}^{t}) + c_2 \times r_{2i} \times (p_{gd} - x_{id}^{t})$	(5) (6)	$w \in [0.9, 0.5]$.
4	[2]	$v_k^d = wv_k^d + \eta \cdot (pbest_k^d - x_k^d)$	(7)	$w = [0.9, 0.5]$, $\eta = 2.0$.
5	[2]	$v_k^d = v_{avg}^d \cdot N(0,1)$	(8)	where $v_{avg}^d = \sum_{k=1}^{N} \left\| v_k^d \right\| / N$, $N(0,1)$ is a random number generated from a normal distribution.
6	[2]	$v_k^d = wv_k^d + \eta \cdot (pbest_{rand}^d - x_k^d)$	(9)	$pbest_{rand}^d$ is the personal best of a random particle.
7	[2]	$v_k^d = wv_k^d + \eta \cdot r_k^d \cdot (gbest^d - x_k^d)$	(10)	This strategy is used to perform learning from the global best.
8	[3]	$\overrightarrow{p_{lbesti}} = (p_{lbesti}^1, p_{lbesti}^2, \ldots, p_{lbesti}^n)$ $v_i^d(t+1) = w(t) \times v_i^d(t) + C_1(t) \times rand_1 \times (p_{lbesti}^d(t) - x_i^d(t)) +$ $c_2(t) \times rand_2 \times (p_{lbesti}^d(t) - x_i^d(t))$	(11) (12)	
9	[4]	$v_{id}(t+1) = \mathrm{sgn}(r_1 - 0.5)(wv_{id}(t) + c_1 r_2 (p_{id} - x_{id}(t)) +$ $c_2 r_3 (p_{gd} - x_{id}(t))$ $\mathrm{sgn}(t) = \begin{cases} -1, & t < 0 \\ 0, & t = 0 \\ 1, & t > 0 \end{cases}$	(13) (14)	
10	[5]	$X_{i+1} = r_1 \cdot X_i + r_2 \cdot pbest_i + r_3 \cdot (X_c - X_d)$	(15)	where X_c and X_d are the position vectors of two random particles.
11	[5]	$X_{i+1} = r_1 \cdot X_i + r_2 \cdot gbest_i + r_3 \cdot (X_c - X_d)$	(16)	
12	[6]	$Viad_i^d \leftarrow X_k^d - X_j^d$ $c = N(0.5, 0.2)$ $V_i^d \leftarrow c \times Viad_i^d + c \times (pbest_i^d - X_i^d)$	(17) (18) (19)	
13	[6]	$c = \frac{(D-1)N(0,1)}{D} + \frac{C(0,1)}{D}$ $V_i^d \leftarrow (mean_i^d - X_i^d) +$ (7) $\frac{c}{\sqrt{3}} \sqrt{(pbest_i^d - mean_i^d)^2 + (X_i^d - mean_i^d)^2 + (X_k^d - mean_i^d)^2}$	(20) (21)	
14	[7]	$V_i^d \leftarrow w \times V_i^d + c \times rand_i^d \times (pbest_{f_i(d)}^d - X_i^d)$	(22)	
15	[6]	$V_i^d \leftarrow w \times V_i^d + 0.5 \times c \times rand_i \times (pbest_{f_i(d)}^d - X_i^d + pbest_i^d - X_i^d)$	(23)	
16	[8]	$F = N(0.5, 0.2)$ $\mathbf{Ab}_i{}' \leftarrow \mathbf{Ab}_{r_1} + F \times (\mathbf{Ab}_{r_2} - \mathbf{Ab}_{r_3})$ $\mathbf{Ab}_i^d = \begin{cases} \mathbf{Ab}_i^{d\prime}, & if\ d \in \mathrm{randSel}(D) \\ \mathbf{Ab}_i^d, & otherwise \end{cases}$ $\mathbf{Ab}_i^d = \begin{cases} \min(2l_i^d - \mathbf{Ab}_i^d, u_i^d), & if\ \mathbf{Ab}_i^d < l_i^d \\ \max(2u_i^d - \mathbf{Ab}_i^d, l_i^d), & if\ \mathbf{Ab}_i^d > u_i^d \end{cases}$	(24) (25) (26) (27)	where $N(0.5, 0.2)$ is a random number.
17	[8]	$\mathbf{Ab}_i{}' \leftarrow \mathbf{Ab}_{r_1} + rand \times (\mathbf{Ab}_{r_2} - \mathbf{Ab}_{r_3}) + F \times (\mathbf{Ab}_{r_4} - \mathbf{Ab}_{r_5})$	(28)	
18	[8]	$F = U(0.6, 1)$ $\mathbf{Ab}_i' \leftarrow \mathbf{Ab}_i + rand \times (\mathbf{Ab}_{r_1} - \mathbf{Ab}_{r_2}) + F \times (\mathbf{Ab}_{r_3} - \mathbf{Ab}_{r_4})$	(29) (30)	where $U(0.6, 1)$is a uniformly distributed random number between 0.6 and 1.
19	[8]	$c = U(0, 2)$ $\mathbf{Ab}_i^d \leftarrow \mathbf{Ab}_i^d + c \times (\mathbf{Ab}_{r_1}^d - \mathbf{Ab}_i^d)$	(31) (32)	
20	[9]	$u_{i,j} = \begin{cases} x_{r_1,j} + F \cdot (x_{r_2,j} - x_{r_3,j}), & \text{if } rand[0,1) < CR \text{ or } j = j_{rand} \\ x_{i,j}, & \text{otherwise} \end{cases}$	(33)	
21	[9]	$u_{i,j} = \begin{cases} x_{i,j} + F \cdot (x_{best,j} - x_{i,j}) + F \cdot (x_{r_1,j} - x_{r_2,j}) + F \cdot (x_{r_3,j} - x_{r_4,j}), & \text{if } rand[0,1) < CR \text{ or } j = j_{rand} \\ x_{i,j}, & \text{otherwise} \end{cases}$	(34)	
22	[9]	$u_{i,j} = \begin{cases} x_{r_1,j} + F \cdot (x_{r_2,j} - x_{r_3,j}) + F \cdot (x_{r_4,j} - x_{r_5,j}), & \text{if } rand[0,1) < CR \text{ or } j = j_{rand} \\ x_{i,j}, & \text{otherwise} \end{cases}$	(35)	
23	[9]	$\mathbf{U}_{i,G} = \mathbf{X}_{i,G} + k \cdot (\mathbf{X}_{r_1,G} - \mathbf{X}_{i,G}) + F \cdot (\mathbf{X}_{r_2,G} - \mathbf{X}_{r_3,G})$	(36)	
24	[7]	$v_i^d = w * v_i^d + c * rand_i^d * (pbest_{fi(d)}^d - X_i^d)$	(37)	
25	[10]	$v_{id} = \chi[v_{id} + c_1 r_{1d}(p_{id} - x_{id}) + c_2 r_{2d}(p_{nd} - x_{id})]$	(38)	

REFERENCES

[1] B. Xue, M. Zhang, and W. N. Browne, "Particle swarm optimisation for feature selection in classification: novel initialisation and updating mechanisms," *Applied Soft Computing*, vol. 18, pp. 261–276, 2014.

[2] C. H. Li, S. X. Yang, and T. T. Nguyen, "A self-learning particle swarm optimizer for global optimization problems," *IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics*, vol. 42, no. 3, pp. 627–646, 2012.

[3] Z. Beheshti, S. M. Shamsuddin, and S. Hasan, "Memetic binary particle swarm optimization for discrete optimization problems," *Information Sciences*, vol. 299, pp. 58–84, 2015.

[4] W. Du and B. Li, "Multi-strategy ensemble particle swarm optimization for dynamic optimization," *Information Sciences*, vol. 178, no. 15, pp. 3096–3109, 2008.

[5] H. Wang, H. Sun, C. H. Li, S. Rahnamayan, and J. S. Pan, "Diversity enhanced particle swarm optimization with neighborhood search," *Information Sciences*, vol. 223, pp. 119–135, 2013.

[6] Y. Wang, B. Li, T. Weise, J. Y. Wang, B. Yuan, and Q. J. Tian, "Self-adaptive learning based particle swarm optimization," *Information Sciences*, vol. 181, no. 20, pp. 4515–4538, 2011.

[7] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 281–295, 2006.

[8] Y. Xue, S. M. Zhong, Y. Zhuang, and B. Xu, "An ensemble algorithm with self-adaptive learning techniques for high-dimensional numerical optimization," *Applied Mathematics and Computation*, vol. 231, pp. 329–346, 2014.

[9] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[10] Z. H. Zhan, J. Zhang, Y. Li, and Y. H. Shi, "Orthogonal learning particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 6, pp. 832–847, 2011.