A Forward Search Inspired Particle Swarm Optimization Algorithm for Feature Selection in Classification

An-Da Li
School of Management
Tianjin University of Commerce
Tianjin, China
adli@tjcu.edu.cn

Bing Xue
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Bing.Xue@ecs.vuw.ac.nz

Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Mengjie.Zhang@ecs.vuw.ac.nz

***Abstract*—Particle swarm optimization (PSO) has been widely used for feature selection (FS) in classification. However, FS is still a challenging optimization task for PSO when the dimensionality of data is high. In this paper, we propose a forward search inspired PSO (FSIPSO) algorithm to build a wrapper-based FS method. In FSIPSO, the search space dynamically changes during the evolutionary process. Specifically, we rank the features according to their single-feature classification performance and divide the search space into several sub-spaces. A forward search scheme is proposed to sequentially select the sub-spaces. The selected sub-spaces construct the search space for FSIPSO. With this scheme, FSIPSO first searches in a small space to quickly find candidate solutions (feature subsets) with relatively good performance. Then, the search space expands with the selection of more sub-spaces, and FSIPSO can further select informative features in the expanded search space. Moreover, mutation operations are used in FSIPSO to avoid the premature problem. The experimental results on 8 UCI datasets have shown that FSIPSO obtains better FS results with less computation time compared with benchmark PSO-based FS methods. FSIPSO also obtains better convergence performance than these methods.**

***Index Terms*—Particle swarm optimization, feature selection, classification, sequential forward selection.**

I. Introduction

With the development of data acquisition technologies, the data with numerous features are more frequently collected in real-world applications. This brings significant challenges for machine learning tasks, including classification, since there are a large number of noisy or redundant features in the collected data. Feature selection (FS) has shown to be an effective dimensionality reduction technique for classification, which aims to select the informative features with respect to the class label. The potential benefits of FS include improving the classification performance and interpretability of the learned model, as well as reducing the computation time of the learning algorithm [1].

This work was supported in part by the Humanities and Social Sciences Youth Fund of Ministry of Education of China under Grants 19YJC630071 and 19YJC630221, the Marsden Fund of New Zealand Government under Contracts VUW1509, VUW1615, VUW1913 and VUW1914, the Science for Technological Innovation Challenge (SfTI) fund under grant E3603/2903, the University Research Fund at Victoria University of Wellington grant number 223805/3986, MBIE Data Science SSIF Fund under the contract RTVU1914, and National Natural Science Foundation of China (NSFC) under Grant 61876169.

FS can be depicted as an optimization problem that maximizes the classification performance by selecting the best feature subset. This problem has shown to be NP-hard, and its solution space substantially expands with the increase of the data dimensionality (i.e., the number of original features) [2]. According to different evaluation criteria, FS can be classified into filter methods and wrapper methods [3]. Filter methods evaluate the potential classification performance of a feature subset with measures based on distance, the information theory, etc. In comparison, wrapper methods straightforwardly use a classification performance measure to evaluate a feature subset. Thus, wrappers generally obtain better classification performance while requiring more computation time than filters. Due to the high performance of wrappers, this paper aims to build a wrapper-based FS method.

The search strategies play an important role in wrapper-based FS methods since FS is an optimization problem. Two generally used heuristics for wrapper-based FS are sequential forward selection (SFS) and sequential backward selection (SBS) [4], which are based on the hill-climbing search strategies. In recent years, evolutionary computation (EC) techniques, such as genetic algorithms [5], [6] and particle swarm optimization (PSO) [7], [8], have been increasingly used as the search strategy for FS due to their good global search capabilities.

PSO [9], inspired by the birds flocking behavior, is a powerful EC technique with competent global search performance. A large number of PSO-based FS methods have been developed in recent years. For example, Xue et al. [8] proposed several PSO-based FS variants, where different swarm initialization and particle updating mechanisms are studied. Nguyen et al. [10] proposed an FS method based on the sticky binary PSO (SBPSO) algorithm, which adopts a dynamic mechanism to balance the exploration and exploitation capabilities of the algorithm. Tran et al. [11] proposed a bare-bone PSO-based FS method that can simultaneously select and discretize features for effective and efficient classification. Pratama et al. [12] proposed an FS method that combines PSO with sequential forward floating selection (SFFS), where SFFS is used as a local search method to update the solutions. However, these methods focus mainly on improving the FS performance, not simultaneously the computational cost.

To improve the FS performance on high-dimensional data, Gu et al. [13] used a PSO variant, called competitive swarm optimizer (CSO), that adopts a pairwise competition solution updating mechanism for FS. Li et al. [14] proposed an improved binary PSO algorithm that adopts a feature-weighting-directed initialization mechanism and a bits-masking-based search space reduction mechanism to improve the FS performance for high-dimensional data. Xue et al. [15] proposed a PSO algorithm that adopts the feature weighting results of ReliefF to guide the swarm initialization for FS. One limitation of this method is that the final FS results are highly dependent on the performance of ReliefF. Moreover, some studies handle the high-dimensional FS problems with PSO using the sub-space strategy. Tran et al. [16] proposed a PSO algorithm with variable lengths of particles, which represent different sub-spaces of the full search space, for high-dimensional FS. Song et al. [17] proposed a cooperative coevolutionary PSO algorithm that uses several sub-swarms in different sub-spaces for high-dimensional FS. However, these two algorithms need several user-defined parameters to dynamically manipulate the sub-spaces during the evolutionary process. Thus, extensive experiments for tuning the parameters are needed.

A. Goals

According to the above analyses, FS is a challenging optimization problem especially when there are a large number of features. SFS uses a forward search scheme that sequentially selects features into the final feature subset. This heuristic scheme substantially reduces the number of search steps and reduces the required computational resources. However, one limitation of SFS is that the global search ability is limited. PSO is an EC technique with promising global search performance. However, there are still limitations in existing PSO-based methods on solving FS problems since the search space significantly increases with the increase of data dimensionality. To inherit the advantages of both SFS and PSO for FS, we aim to build a PSO algorithm by embedding the forward search scheme into the evolutionary process. This proposed algorithm is called forward search inspired PSO (FSIPSO).

FSIPSO first divides the original search space into several sub-spaces, where each sub-space is formed of a subset of the features. Then, during the evolutionary process, a forward search scheme is used to sequentially select the sub-spaces to dynamically construct the search space. The solutions are only allowed to be updated in the constructed search space. FSIPSO also adopts the mutation operations to further improve the global search performance. The performance of FSIPSO is examined on 8 UCI datasets with more than 100 features. Specifically, the following research points will be addressed in this paper:

1) Design an algorithm (i.e., FSIPSO) that properly combines the forward search scheme with PSO to improve the FS performance,
2) Introduce mutation operations to the above algorithm to improve the global search ability,
3) Compare the FS performance of FSIPSO with benchmark PSO-based FS methods as well as SFS and SBS, and
4) Compare the search performance of FSIPSO with benchmark PSO algorithms.

II. Background

This section briefly introduces the preliminaries of our proposed method, including the main idea of PSO and how PSO is applied to address FS problems.

A. Particle Swarm Optimization

In (standard) PSO [9], a swarm of particles is initialized, where each particle's position represents a solution in the continuous search space. Specifically, the position of the ith particle in the swarm can be denoted by a vector $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{iD})$, where D is the dimensionality of the search space. In PSO, it is assumed that each particle is continuously moving during the evolutionary process. Thus, each particle i has a velocity, denoted by $\mathbf{v}_i = (v_{i1}, v_{i2}, ..., v_{iD})$. Each particle updates its velocity $\mathbf{v}_i$ and position $\mathbf{x}_i$ according to its personal experience and the global experience of the swarm during the evolutionary process. Generally, the personal experience is reflected by the best position (denoted by $pbest_i$) this particle has moved to, and the global experience is shown by the best position (denoted by $gbest$) found by the swarm so far. Based on these notations, the velocity $\mathbf{v}_i$ and position $\mathbf{x}_i$ for the particle i at the tth generation is updated to

$$v_{id}^{t+1} = w * v_{id}^{t} + c_1 * r_1 * (p_{id} - x_{id}^{t}) + c_2 * r_2 * (g_d - x_{id}^{t}) \quad (1)$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \quad (2)$$

where $d = 1, 2, ..., D$ indicates the index of the considered dimension in the search space, and p_{id} and g_d denote the dth element of the $pbest_i$ and $gbest$. w, c_1, and c_2 are three user-defined parameters controlling the evolutionary behavior of PSO. Specifically, w is the inertia weight reflecting the exploration intensity, and c_1 and c_2 reflect the impacts of $pbest_i$ and $gbset$. r_1 and r_2 are two random values from the uniform distribution $U(0, 1)$.

B. Application of PSO to FS

Let the number of original features in the dataset be D. For an FS problem, a particle position $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{iD})$ in PSO can be used to represent a feature subset (solution), where each element x_{id} ($d = 1, 2, ..., D$) denotes whether the ith feature is selected or not by comparing it with a predefined threshold parameter η. Specifically, $x_{id} > \eta$ (or $x_{id} \leq \eta$) denotes the dth feature is selected (or eliminated).

The objective of an FS problem is to obtain a small feature subset with good classification performance. It is suggested in literature [18], [19] to use a combined fitness function considering both the classification performance and the feature subset size for an FS problem. In this paper, as suggested in [19], we adopt the following combined fitness function

$$F(\mathbf{x}_i) = \alpha * acc(\mathbf{x}_i) + (1-\alpha) * (1 - \frac{\#\mathbf{x}_i}{D}) \tag{3}$$

where $acc(\mathbf{x}_i)$ denotes the accuracy obtained by $\mathbf{x}_i$, $\#\mathbf{x}_i$ denotes the the number of selected features by $\mathbf{x}_i$, and D is the total number of original features. $\alpha \in (0, 1)$ and $1-\alpha \in (0, 1)$ reflects the weights of classification performance and feature number to evaluate a feature subset's fitness. In this paper, we set $\alpha = 0.9$ as suggested in [19], meaning that the classification performance is much more important than the number of selected features.

III. The Proposed Approach

This section describes the proposed FSIPSO algorithm in detail, including solution representation, the search space construction strategy (described in Sections III-B, III-C, and III-D), mutation operations, and overall algorithm.

A. Solution Representation

In FSIPSO, the position (which denotes a feature subset) of a particle i is represented by a real vector $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{iD})$, where each $x_{id} \in [0, 1], d = 1, ..., D$. Given a predefined parameter $\eta \in (0, 1)$, the dth feature is selected (eliminated) by $\mathbf{x}_i$ if $x_{id} > \eta$ ($x_{id} \leq \eta$).

B. Ranking Features

In FSIPSO, we first rank the features according to their importance in descending order. This is beneficial that we can construct sub-spaces with different utilities, where a sub-space with a higher utility means that this sub-space has a potentially higher probability containing more informative features. Thus, we can design a search space construction scheme to make the algorithm pay more effort on sub-spaces with higher utilities. The feature ranking strategy has been used in many studies. It can be based on a filter-based measure (e.g., symmetric uncertainty (SU) [3], [16], [17], [20] and the attention-based feature ranking mechanism [21]) or a wrapper-based measure (e.g., classification accuracy [22], [23]). In this paper, a wrapper-based measure is used to rank features, since it straightforwardly measures a feature's predictive performance. Specifically, the 5-fold cross-validation [4] is used based on the training set to obtain the classification accuracy of every single feature, and then the features are ranked according to the accuracy rates in descending order.

C. Generating Sub-spaces

After ranking the features based on the accuracy measure, we can divide the original search space into several sub-spaces. Let the ranked feature set be $\mathbb{F}^s = \{f_1^s, f_2^s, ..., f_D^s\}$, which can be used to represent the original search space. Then, we uniformly divide the search space into M sub-spaces. Each sub-space can be represented by a feature subset denoted by

$$\mathbb{F}_k^s = \{f^s_{(k-1)*\frac{D}{M}+1}, f^s_{(k-1)*\frac{D}{M}+2}, ..., f^s_{k*\frac{D}{M}}\}, k = 1, ..., M \tag{4}$$

where k is the index of the sub-space and $\frac{D}{M}$ is the number of features in each sub-space. According to the definition of $\mathbb{F}_k^s$, each sub-space has the same dimensionality $\frac{D}{M}$, and all the sub-spaces compose the original search space, i.e., we have $\mathbb{F}^s = \mathbb{F}_1^s \cup \mathbb{F}_2^s \cup \cdots \cup \mathbb{F}_M^s$. Moreover, since the division of the search space is based on the ranked feature set $\mathbb{F}^s$, the sub-spaces $\mathbb{F}_k^s$, $k = 1, ..., M$ have different utilities. Specifically, a lower value of k shows a higher utility of the sub-space $\mathbb{F}_k^s$.

D. Constructing the Search Space with Forward Search

In this paper, a forward search scheme is used to dynamically construct the search space $\mathbb{F}_c$ during the evolutionary process. This scheme divides the evolutionary process of FSIPSO into M (the number of sub-spaces) phases. The $1, 2, ..., M$th sub-spaces are sequentially added into $\mathbb{F}_c$ to construct the search space during the evolutionary process, which means that a sub-space with a high utility has priority to be selected in $\mathbb{F}_c$. Specifically, the search space $\mathbb{F}_c$ at the mth ($m = 1, 2, ..., M$) evolutionary phase is obtained as

$$\mathbb{F}_c = \mathbb{F}_1^s \cup, ..., \cup \mathbb{F}_m^s. \tag{5}$$

FSIPSO is allowed to only evolve solutions in $\mathbb{F}_c$, i.e., for a particle's velocity $\mathbf{v}_i = (v_{i1}, v_{i2}, ..., v_{iD})$ and position $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{iD})$, only the elements v_{id} and x_{id}, $d \in \mathbb{F}_c$ are updated based on Eqs. (1) and (2).

E. Mutation Operations

To improve the global search performance, mutation operations are conducted in FSIPSO to further update the particles. In FSIPSO, each element x_{id} in a particle position $\mathbf{x}_i$ is a real value in $[0, 1]$. Let the elements in the particle position updated by the PSO mechanism using Eqs. (1) and (2) be x_{id}^{t+1}, $d \in \mathbb{F}_c$. The mutations operation is conducted to update each x_{id}^{t+1} to $\overline{x}_{id}^{t+1}$ ($d \in \mathbb{F}_c$) with the following equation:

$$\overline{x}_{id}^{t+1} = \begin{cases} 1 - x_{id}^{t+1} & \text{if} \quad rand() < p_m \\ x_{id}^{t+1} & \text{else} \end{cases}, \tag{6}$$

where $rand()$ denotes a random value in $[0, 1]$, p_m is the mutation rate determining the probability of an element to mutate. Specifically, p_m is defined as

$$p_m = 1/|\mathbb{F}_c|, \tag{7}$$

where $|\mathbb{F}_c|$ is the number of features in $\mathbb{F}_c$. Therefore, the expected number of mutated elements in a particle is 1.

F. Overall Algorithm

The overall procedure of FSIPSO is shown in Algorithm 1. First, the features are ranked according to the wrapper-based accuracy measure described in Section III-B, and the sub-spaces are generated with the procedure described in Section

III-C. Second, the search space is initialized as the first sub-space, i.e., $\mathbb{F}_c = \mathbb{F}_1^s$. We initialize the swarm $\mathbb{S}^t$ in $\mathbb{F}_c$ since FSIPSO only evolves new solutions in the search space $\mathbb{F}_c$. Specifically, for each particle position $\mathbf{x}_i^t = (x_{i1}^t, x_{i2}^t, ..., x_{iD}^t)$, the elements x_{id}^t, $d \in \mathbb{F}_c$ are initialized as a random value in $[0, 1]$ and other elements x_{ik}^t, $k \notin \mathbb{F}_c$ are assigned as a value of 0. Third, during the iterations, FSIPSO first determines if the algorithm should move to the next evolutionary phase or not. Specifically, FSIPSO moves to the next evolutionary phase every $\lfloor \frac{T}{M} \rfloor$ generations. This setting ensures that the search space with all features can be constructed before the algorithm stops. Once the condition of changing the evolutionary phase matches, the phase counter m is added by 1 and the search space $\mathbb{F}_c$ is updated by further selecting/adding the sub-space $\mathbb{F}_m^s$ into it with the forward search scheme. Since the search space expands, we additionally initialize each particle $\mathbf{x}_i^t = (x_{i1}^t, x_{i2}^t, ..., x_{iD}^t)$ in $\mathbb{S}^t$ in the added sub-space $\mathbb{F}_m^s$, i.e., each x_{id}^t, $d \in \mathbb{F}_m^s$ is initialized as a random value in $[0, 1]$. The particle positions are then updated based on the PSO solution updating mechanism (shown in Eqs. (1) and (2)) and the proposed mutation operations (shown in Eq. (6)) in the constructed search space $\mathbb{F}_c$. Based on the updated particle positions, the $pbest_i$ of each particle i and the $gbest$ are updated. Finally, after FSIPSO reaches the stopping criterion, the $gbest$ is returned as the found best solution (feature subset).

Note that, since the single-feature classification performance is used to rank features in FSIPSO, the correlations among features are not considered during the feature ranking step. The redundant features might have a high importance value according to the ranking measure. Thus, the ranked features roughly but not precisely reflect their true importance in descending order. However, since we use the combined fitness function considering both the classification performance and the number of selected features, the redundant features can be removed during the evolutionary process of FSIPSO, which addresses the feature correlation issue in FS.

Algorithm 1: Pseudo-Code of FSIPSO.

```
Input  : The training set $tra$, maximum number of
         generations $T$, swarm size $S$, number of the
         evolutionary phases $M$;
Output : The best solution (feature subset) $\mathbf{x}^*$;
1  Rank the features based on $tra$ using the wrapper-based
     accuracy measure described in Section III-B;
2  Generate the sub-spaces $\mathbb{F}_1^s, ..., \mathbb{F}_M^s$ based on the procedure
     described in Section III-C;
3  $\mathbb{F}_c \leftarrow \mathbb{F}_1^s$ ; /* Initialize the search space. */
4  $t \leftarrow 0$, $m \leftarrow 1$;
5  $\mathbb{S}^t \leftarrow \{\mathbf{x}_1^t, \mathbf{x}_2^t, ..., \mathbf{x}_S^t\}$, where each $\mathbf{x}_i^t = (0)_{\times D}$, $i = 1, ..., S$;
6  Initialize each particle's velocity $\mathbf{v}_i^t = (0)_{\times D}$, $i = 1, ..., S$;
7  Initialize $\mathbb{S}^t$ in the search space $\mathbb{F}_c$ and evaluate the fitness
     value for each particle in $\mathbb{S}^t$ based on Eq. (3);
8  Update $pbest_i$ for each particle $i$ and update $gbest$;
9  while $t < T$ do
10     if $t > 0$ && $(t \ \mathbf{rem} \lfloor \frac{T}{M} \rfloor) = 0$ then
11         $m \leftarrow m + 1$;
12         $\mathbb{F}_c \leftarrow \mathbb{F}_c \cup \mathbb{F}_m^s$ ;     /* Update the search
             space with forward search. */
13         Initialize $\mathbb{S}^t$ in the added sub-space $\mathbb{F}_m^s$;
14     end
15     for $i \leftarrow 1$ to $S$ do
16         $\mathbf{x}_i^{t+1} \leftarrow \mathbf{x}_i^t = (x_{i1}^t, x_{i2}^t, ..., x_{iD}^t)$;
17         $\mathbf{v}_i^{t+1} \leftarrow \mathbf{v}_i^t = (v_{i1}^t, v_{i2}^t, ..., v_{iD}^t)$;
18         foreach $d \in \mathbb{F}_c$ do
19             $v_{id}^{t+1}$, $x_{id}^{t+1} \leftarrow$ Update $v_{id}^t$ and $x_{id}^t$ in $\mathbf{v}_i^{t+1}$ and
               $\mathbf{x}_i^{t+1}$ using Eqs. (1) and (2);
20             $\overline{x}_{id}^{t+1} \leftarrow$ Update $x_{id}^{t+1}$ in $\mathbf{x}_i^{t+1}$ using the
               mutation operation shown in Eq. (6);
21         end
22         Evaluate the fitness value of $\mathbf{x}_i^{t+1}$ based on Eq. (3)
             and update $pbest_i$ with $\mathbf{x}_i^{t+1}$;
23     end
24     $\mathbb{S}^{t+1} \leftarrow \{\mathbf{x}_1^{t+1}, \mathbf{x}_2^{t+1}, ..., \mathbf{x}_S^{t+1}\}$;
25     Update $gbest$ with $\mathbb{S}^{t+1}$;
26     $t \leftarrow t + 1$;
27 end
28 return $\mathbf{x}^* \leftarrow gbest$;
```

IV. Experiment Design

This section designs the experiments for verifying the proposed method. The datasets, benchmark methods, and parameter settings used in the experiments are described below.

A. Datasets

Eight datasets of varying numbers of instances, features, and classes from the UCI machine learning repository (http://archive.ics.uci.edu/ml) [24] are employed in the experiments. In each selected dataset, at least 100 features are included. The details of the datasets are shown in Table I.

B. Benchmark Methods

Eight benchmark FS methods are adopted in the experiments. First, the two conventional wrapper methods based on hill-climbing search strategies, i.e., SFS [4] and SBS [4] are used. Second, the FS method (denoted by PSO(C)) based on the standard PSO [9] is used as the benchmark method, since our method is established based on the standard PSO. In PSO(C), the same combined fitness function as FSIPSO shown in Eq. (3) is used. Finally, five wrapper methods based on recently proposed continuous and binary PSO algorithms are used as the benchmark methods to comprehensively evaluate the performance of the proposed method. These methods include PSO(4-2) [8], CSO(C) [25], CSO(A) [13], Up binary PSO (UBPSO) [26] and SBPSO [19]. Specifically, PSO(4-2) is a continuous PSO variant that adopts new initialization and solution updating mechanisms designed for FS. CSO(A) is established based on CSO for large-scale FS problems. In PSO(4-2) and CSO(A), the accuracy is used as the fitness function as suggested in [8] and [13]. To comprehensively compare the FS performance between FSIPSO and CSO, CSO(C) is also adopted as a benchmark method. CSO(C) adopts CSO [25] as the optimizer and uses the same combined fitness function as FSIPSO. UBPSO and SBPSO are two binary PSO algorithms. UBPSO dynamically increases the value of the inertia weight during the evolutionary process. SBPSO adopts a probability-based solution updating mechanism instead of that used in traditional binary PSO. Similarly, UBPSO and SBPSO adopt the same fitness function as FSIPSO.

TABLE I
DATASETS

Dataset	#Instances	#Features	#Classes
Urban	675	147	9
Musk1	476	166	2
Arrhythmia	452	279	13
LSVT	126	310	2
Isolet5	1559	617	26
Mfeat	2000	649	10
InterAd	3279	1558	2
DrivFace	606	6400	3

C. Parameter Settings

In FSIPSO, PSO(C) and PSO(4-2), we set the maximum number of generations as $T = 100$, the swarm size as $S = 30$, the inter weight as $w = 0.7298$, and the two constants to update the velocity as $c_1 = c_2 = 1.49618$ as suggested in [8]. The parameter that determines whether a feature is selected or not in a real-coded particle is set as $\eta = 0.6$ as suggested in [7] and [8]. The number of evolutionary phases in FSIPSO is set as $M = 5$, which is based on several tuning experiments. In CSO(C) and CSO(A), the same swarm size $S = 30$ as FSIPSO is used. Since the CSO algorithm only updates $S/2$ particles at each generation, we set the maximum number of generations in CSO(C) and CSO(A) as $T = 200$, twice that of FSIPSO, to make a fair comparison. The parameters ϕ and η in CSO(C) and CSO(A) are set as 0.1 and 0.5 as suggested in [13]. In UBPSO and SBPSO, the same swarm size $S = 30$ and the maximum number of generations $T = 100$ as FSIPSO are used. In UBPSO, the upper and lower bounds of the inertia weight are set as $\overline{w} = 1$ and $\underline{w} = 0.4$, and the parameter ρ is set as 0.9 according to [26]. In SBPSO, we set the step parameter $L = 50$, and set the three parameters to control the evolving behavior as $i_m = 0.25$, $i_p = 0.25$, and $i_g = 0.5$ according to [19]. Finally, SFS and SBS use the default settings in Waikato Environment for Knowledge Analysis (Weka) [27].

In the experiments, the K nearest neighbor (KNN) [28] classifier is used for its good performance and simplicity. The number of neighbors is set as $K = 5$ as used in [8]. Each original dataset is randomly divided into a training set (70%) and a test set (30%). During the FS process with the wrapper-based FS methods, the training set is further divided into 5 folds to perform an inner 5-fold cross-validation process to evaluate the fitness values of solutions [4]. Then, after the FS process, the test set is used to evaluate the performance of the found feature subset of each method. The experiments on each dataset with the stochastic FS methods (the methods except SFS and SBS) run 30 times independently. The Wilcoxon rank-sum test [29] with a significance level of 0.05 is used to compare the results between FSIPSO and each benchmark method. SFS, SBS, and KNN are implemented based on Weka, and other FS methods are implemented based on Java. The experiments are run on PCs with an 8 GB memory and a 3.4 GHz CPU.

TABLE II
FS RESULTS OF FSIPSO, SFS, AND SBS

Dataset	Method	#Features (std.)	T_f	BestA (%)	MeanA (Std.)(%)	T_a
	FSIPSO	14.8 (2.8)		84.31	81.88 (1.48)	
Urban	SFS	9.0	$-$	80.88	80.88	$+$
	SBS	113.0	$+$	78.92	78.92	$+$
	FSIPSO	16.9 (2.4)		86.11	82.85 (2.51)	
Musk1	SFS	10.0	$-$	79.86	79.86	$+$
	SBS	122.0	$+$	80.56	80.56	$+$
	FSIPSO	22.4 (3.0)		72.99	70.17 (1.44)	
Arrhythmia	SFS	19.0	$-$	70.80	70.80	$-$
	SBS	110.0	$+$	61.31	61.31	$+$
	FSIPSO	15.4 (6.2)		94.87	87.69 (3.65)	
LSVT	SFS	4.0	$-$	76.92	76.92	$+$
	SBS	55.0	$+$	84.61	84.61	$+$
	FSIPSO	101.8 (11.3)		89.96	88.13 (0.82)	
Isolet5	SFS	35.0	$-$	85.68	85.68	$+$
	SBS	N/A	N/A	N/A	N/A	N/A
	FSIPSO	47.4 (5.3)		98.50	98.03 (0.30)	
Mfeat	SFS	11.0	$-$	97.67	97.67	$+$
	SBS	N/A	N/A	N/A	N/A	N/A
	FSIPSO	113.9 (9.9)		97.97	97.45 (0.41)	
InterAd	SFS	14.0	$-$	96.65	96.65	$+$
	SBS	N/A	N/A	N/A	N/A	N/A
	FSIPSO	483.5 (23.0)		97.27	96.48 (0.64)	
DrivFace	SFS	9.0	$-$	93.99	93.99	$+$
	SBS	N/A	N/A	N/A	N/A	N/A

N/A: Not Available.

V. RESULTS AND DISCUSSION

This section first compares the results between FSIPSO and SFS/SBS. Then, the FS results are compared between FSIPSO and the PSO-based benchmark methods. Finally, the computation time of the methods is compared.

A. Comparison with SFS and SBS

Table II shows the FS results of FSIPSO, SFS, and SBS. The results of SBS on Isolet5, Mfeat, InterAd, and DrivFace are not available (N/A), since the experiments on these datasets did not finish within four days. In the table, #Features (std.) and MeanA (Std.) denote the mean (standard deviation) of "the number of selected features" and the mean (standard deviation) of "accuracy" over the 30 experimental runs. BestA denotes the best accuracy of the 30 experimental runs. In columns T_f and T_a, the statistical significance test results from the Wilcoxon rank-sum test comparing FSIPSO with SFS/SBS on the #Features and MeanA measures are shown. Specifically, "+" or "-" denotes that FSIPSO obtains significantly better or worse results than the compared method, and "=" denotes that there is no significant difference.

1) FSIPSO versus SFS: Compared with SFS, FSIPSO obtains significantly higher mean accuracy rates on 7 of the 8 datasets, and only obtains a lower mean accuracy rate on Arrhythmia. According to the number of selected features, FSIPSO selects significantly more features than SFS on all the 8 datasets. On some relatively lower-dimensional datasets, e.g., Urban and Musk1, the numbers of features selected by FSIPSO and SFS are close. Whereas, on relatively higher-dimensional datasets, e.g., InterAd and DrivFace, SFS selects substantially fewer features than FSIPSO. Seeing that FSIPSO obtains better accuracy results than SFS, it is shown that SFS may fail to select some informative features on the datasets.

2) FSIPSO versus SBS: Compared with SBS, FSIPSO obtains higher mean accuracy rates while selecting fewer features on the four available datasets, i.e., Urban, Musk1, Arrhythmia, and LSVT. The statistical significance test results comparing the two methods on the number of selected features and accuracy are all significant. These results show that FSIPSO obtains substantially better FS results than SBS.

B. Comparison with PSO-Based Benchmark Methods

The FS results of FSIPSO and the PSO-based benchmark methods, including PSO(C), PSO(4-2), CSO(C), CSO(A), UBPSO and SBPSO, are shown in Table III, where the same notations as Table II are used. The comparisons between FSIPSO and these PSO methods are shown below.

1) FSIPSO versus PSO(C) and PSO(4-2): Compared with PSO(C), FSIPSO obtains significantly higher mean accuracy rates on 6 of the 8 datasets, and obtains similar mean accuracy rates on 2 datasets, Urban and Musk1. Moreover, FSIPSO selects substantially fewer features than PSO(C) on all the 8 datasets. Similarly, the comparison between FSIPSO and PSO(4-2) also reveals that FSIPSO obtains significantly better mean accuracy rates (6 out of 8 datasets) while selecting substantially fewer features in most cases. In two cases, FSIPSO obtains a similar or lower mean accuracy rate compared with PSO(4-2), i.e., on Musk1 and Mfeat. These results show that, in most cases, FSIPSO selects fewer features while obtaining better accuracy results than PSO(C) and PSO(4-2).

2) FSIPSO versus CSO(C) and CSO(A): Compared with CSO(C), FSIPSO obtains significantly higher mean accuracy rates on 5 of the 8 datasets, obtains similar mean accuracy rates on Urban and LSVT, and obtains a worse mean accuracy rate on Musk1. Compared with CSO(A), FSIPSO obtains significantly higher mean accuracy rates on 5 of the 8 datasets, obtains similar mean accuracy rates on Urban and Mfeat, and obtains a worse accuracy rate on Musk1. On all the datasets, FSIPSO selects substantially fewer features than CSO(C) and CSO(A). These results show that, in most cases, FSIPSO obtains better or similar accuracy results while selecting fewer features. FSIPSO outperforms CSO(C) and CSO(A).

3) FSIPSO versus UBPSO and SBPSO: Compared with UBPSO, FSIPSO obtains significantly higher mean accuracy rates on 4 of the 8 datasets, obtains similar mean accuracy rates on Urban, Mfeat, and InterAd, obtains a worse accuracy rate on Musk1. Compared with SBPSO, FSIPSO obtains significantly higher mean accuracy rates on 6 of the 8 datasets, obtains a similar mean accuracy rate on Urban, and obtains a worse accuracy rate on Musk1. Seeing that FSIPSO selects fewer features on all the datasets, we can conclude that FSIPSO obtains better FS results than the two binary PSO algorithms in most cases.

The above results show that FSIPSO substantially reduces the number of selected features while obtaining similar or better accuracy results than the benchmark PSO-based FS methods in most cases. Specifically, FSIPSO obtains 68.9%, 73.4%, 39.3%, 84.6%, 59.1%, 77.7%, 79.7% and 55.5% fewer features than the second-best PSO-baed benchmark method (on reducing features) on the 8 datasets. This denotes that FSIPSO obtains better feature reduction performance than the benchmark PSO algorithms with the forward search scheme. Different from the benchmark PSO algorithms, FSIPSO searches for new solutions in a dynamically increasing search space. The initial search space of FSIPSO is much smaller than the space with full features, and it has the highest potential utility since the features are ranked. Hence, FSIPSO can quickly evolve relatively good particles with a few features in the small initial search space. During the following evolutionary phases, the expansion of the search space allows particles in FSIPSO to select new features in the added sub-spaces. However, only when the selection of new features improves the fitness values, the *pbest* and *gbest* can be updated. Thus, similar to SFS, the forward search scheme makes FSIPSO favor feature subsets with small sizes. This explains why FSIPSO performs effectively in feature reduction.

TABLE III
FS Results of FSIPSO and PSO-Based Benchmark Methods

Dataset	Method	#Features (std.)	T_f	BestA	MeanA (Std.)	T_a
Urban	FSIPSO	14.8 (2.8)		84.31	81.88 (1.48)	
	PSO(C)	49.5 (5.9)	+	84.80	81.62 (1.22)	=
	PSO(4-2)	63.1 (28.7)	+	84.31	81.03 (1.53)	+
	CSO(C)	47.9 (5.0)	+	84.31	81.75 (1.35)	=
	CSO(A)	66.1 (7.2)	+	83.82	81.65 (1.15)	=
	UBPSO	51.0 (5.7)	+	84.80	81.49 (1.66)	=
	SBPSO	47.5 (4.2)	+	85.78	82.04 (1.70)	=
Musk1	FSIPSO	16.9 (2.4)		86.11	82.85 (2.51)	
	PSO(C)	63.3 (7.7)	+	89.58	84.21 (2.95)	=
	PSO(4-2)	67.6 (24.8)	+	91.67	84.86 (3.42)	−
	CSO(C)	65.1 (5.8)	+	91.67	87.66 (2.35)	−
	CSO(A)	78.2 (6.1)	+	92.36	88.26 (2.18)	−
	UBPSO	64.2 (5.9)	+	93.06	87.89 (2.43)	−
	SBPSO	67.4 (6.2)	+	92.36	87.57 (2.12)	−
Arrhythmia	FSIPSO	22.4 (3.0)		72.99	70.17 (1.44)	
	PSO(C)	93.5 (7.1)	+	65.69	61.34 (1.83)	+
	PSO(4-2)	36.9 (16.4)	+	72.99	65.62 (2.30)	+
	CSO(C)	95.3 (7.7)	+	66.42	62.80 (1.73)	+
	CSO(A)	125.9 (9.4)	+	67.15	62.68 (2.04)	+
	UBPSO	113.5 (8.9)	+	66.42	63.02 (1.63)	+
	SBPSO	105.9 (8.3)	+	64.96	62.72 (1.64)	+
LSVT	FSIPSO	15.4 (6.2)		94.87	87.69 (3.65)	
	PSO(C)	106.1 (7.9)	+	89.74	83.93 (3.01)	+
	PSO(4-2)	100.5 (57.9)	+	92.31	84.53 (4.01)	+
	CSO(C)	113.2 (8.6)	+	92.31	86.41 (3.10)	=
	CSO(A)	152.9 (7.1)	+	89.74	84.79 (3.86)	+
	UBPSO	142.1 (8.5)	+	89.74	83.42 (3.91)	+
	SBPSO	135.5 (9.2)	+	92.31	84.36 (3.18)	+
Isolet5	FSIPSO	101.8 (11.3)		89.96	88.13 (0.82)	
	PSO(C)	248.7 (14.8)	+	86.54	84.08 (1.31)	+
	PSO(4-2)	313.3 (69.4)	+	85.47	83.18 (1.30)	+
	CSO(C)	257.6 (12.7)	+	88.46	86.92 (0.95)	+
	CSO(A)	291.4 (14.2)	+	88.25	86.58 (0.82)	+
	UBPSO	274.7 (8.9)	+	87.61	85.66 (0.97)	+
	SBPSO	268.8 (12.0)	+	87.61	85.64 (1.08)	+
Mfeat	FSIPSO	47.4 (5.3)		98.50	98.03 (0.30)	
	PSO(C)	212.7 (7.7)	+	98.33	97.86 (0.32)	+
	PSO(4-2)	398.4 (69.0)	+	98.33	97.96 (0.22)	=
	CSO(C)	219.1 (11.0)	+	98.33	97.79 (0.27)	+
	CSO(A)	324.2 (10.3)	+	98.50	97.96 (0.24)	=
	UBPSO	254.8 (11.2)	+	98.33	97.90 (0.31)	=
	SBPSO	250.5 (11.5)	+	98.33	97.77 (0.30)	+
InterAd	FSIPSO	113.9 (9.9)		97.97	97.45 (0.41)	
	PSO(C)	561.0 (20.6)	+	97.46	96.87 (0.43)	+
	PSO(4-2)	792.1 (121.2)	+	97.26	96.76 (0.24)	+
	CSO(C)	620.2 (18.2)	+	97.76	97.24 (0.34)	+
	CSO(A)	773.1 (25.5)	+	97.87	97.20 (0.32)	+
	UBPSO	704.7 (16.4)	+	97.76	97.31 (0.32)	=
	SBPSO	696.6 (15.4)	+	97.76	97.13 (0.39)	+
DrivFace	FSIPSO	483.5 (23.0)		97.27	96.48 (0.64)	
	PSO(C)	2362.5 (57.6)	+	96.17	95.12 (0.40)	+
	PSO(4-2)	1085.7 (514.8)	+	97.27	95.77 (0.77)	+
	CSO(C)	2796.0 (54.4)	+	95.63	95.16 (0.37)	+
	CSO(A)	3200.1 (53.1)	+	96.17	95.17 (0.56)	+
	UBPSO	3017.4 (33.1)	+	96.17	95.28 (0.39)	+
	SBPSO	2971.2 (29.7)	+	96.17	95.21 (0.37)	+

C. Computation Time

Fig. 1 shows the average computation time taken by the PSO-based FS methods over the 30 experimental runs and the computation time taken by SFS and SBS. It is clearly shown that FSIPSO requires substantially less computation time than the benchmark methods except for SFS on the 8 datasets. SFS is time-efficient because the time for evaluating feature subsets can be substantially reduced as the adopted sequential forward search strategy can keep the feature subsets to be evaluated during the search process in a very small size. Similar to SFS, FSIPSO starts the evolutionary process from a very small sub-space and further selects informative features along with the expansion of the search space. Therefore, the feature subsets to be evaluated in FSIPSO are kept in small sizes during the whole evolutionary process, and the computation time of FSIPSO can be substantially reduced.

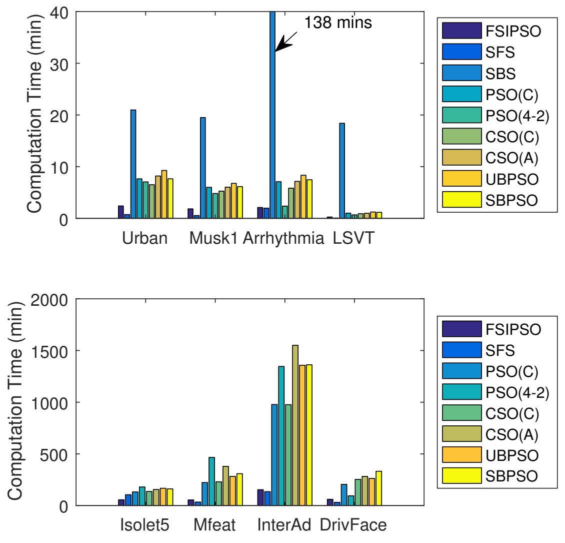

Fig. 1. Computation time of each FS method.

VI. Further Analysis

In Section V, it has shown that FSIPSO obtains better FS results than the PSO algorithms. We further compare the convergence curves of FSIPSO and the compared PSO algorithms in Fig. 2, where the x-axis denotes the number of generations and the y-axis denotes the average fitness value of $gbest$ at each generation over the 30 runs. Except for FSIPSO, only the convergence curves of PSO(C), CSO(C), UBPSO, and SBPSO are drawn, because these algorithms use the same fitness function as FSIPSO. It should be noted that the number of generations of CSO(C) is twice that of other algorithms. We choose the fitness values every two generations (i.e., the fitness values at generations $0, 2, 4, ..., 200$) to draw the convergence curves of CSO(C) to facilitate the comparison.

According to Fig. 2, it is clearly shown that FSIPSO obtains better convergence curves than the compared PSO algorithms. FSIPSO can start and end with a higher fitness value than all the compared PSO algorithms on each dataset. This shows that FSIPSO obtains good search performance for FS problems. Moreover, the effect of the forward search scheme for search space construction can also be found in the figure. For example, on Urban and Isolet5, the convergence curves of FSIPSO have an obvious growing trend at generation 20 when a new sub-space is selected in the search space.

Two reasons can explain why the proposed forward search scheme is effective for improving the search performance of FSIPSO. First, since a dynamically increasing search space is used, FSIPSO generally updates solutions in a space smaller than that with all features. This actually reduces the search space of FSIPSO compared with that of most other PSO-based FS methods, and thus accelerates the evolutionary process. Second, with the proposed forward search scheme, FSIPSO first focuses on selecting features in the sub-space $\mathbb{F}_1^s$ with the highest utility. Thus, the informative features in the first sub-space can be quickly found in the low dimensional space $\mathbb{F}_1^s$. Then, in the following evolutionary phases, FSIPSO can further select informative features in the newly added sub-spaces on the basis of the evolved particles in the previous phases. Although the search space continuously increases during the evolutionary process, at each phase, FSIPSO actually searches for new solutions around the currently found solutions (with potentially informative features). Therefore, compared with the benchmark PSO algorithms that initialize and evolve solutions in the space with all features during the whole evolutionary process, the evolutionary performance of FSIPSO is improved.

VII. Conclusions

In this paper, a forward search inspired PSO algorithm (i.e., FSIPSO) is proposed for FS in classification. In FSIPSO, a forward search scheme is used to dynamically construct the search space from a relatively smaller sub-space. Moreover, the mutation operations are adopted in FSIPSO to avoid the algorithm being trapped into local optima. We have compared the performance of FSIPSO with SFS, SBS, and several PSO-based FS methods on 8 UCI datasets of varying difficulty. The results show that FSIPSO can substantially reduce the number of selected features while obtaining high classification performance. In the meantime, FSIPSO requires much less computation time than the compared PSO algorithms. Further comparisons on the convergence curves also reveal that FSIPSO obtains good convergence performance.

Compared with standard PSO, one additional user-defined parameter, the number of evolutionary phases M, is required in FSIPSO. It would be interesting to further examine the effects of different values of M in FSIPSO and obtain a strategy for properly setting M given different optimization problems. In the future, we will further investigate FSIPSO with other classifiers, such as light gradient boosting machines (GBM), support vector machines (SVM), and random forests, in terms of the accuracy, the number of features, and the computational time. Moreover, we will compare FSIPSO with other state-of-the-art FS methods in more detail using a controlled set of experiments as that used in [30].

References

[1] I. Guyon and A. Elisseeff, “An introduction to variable and feature selection,” *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

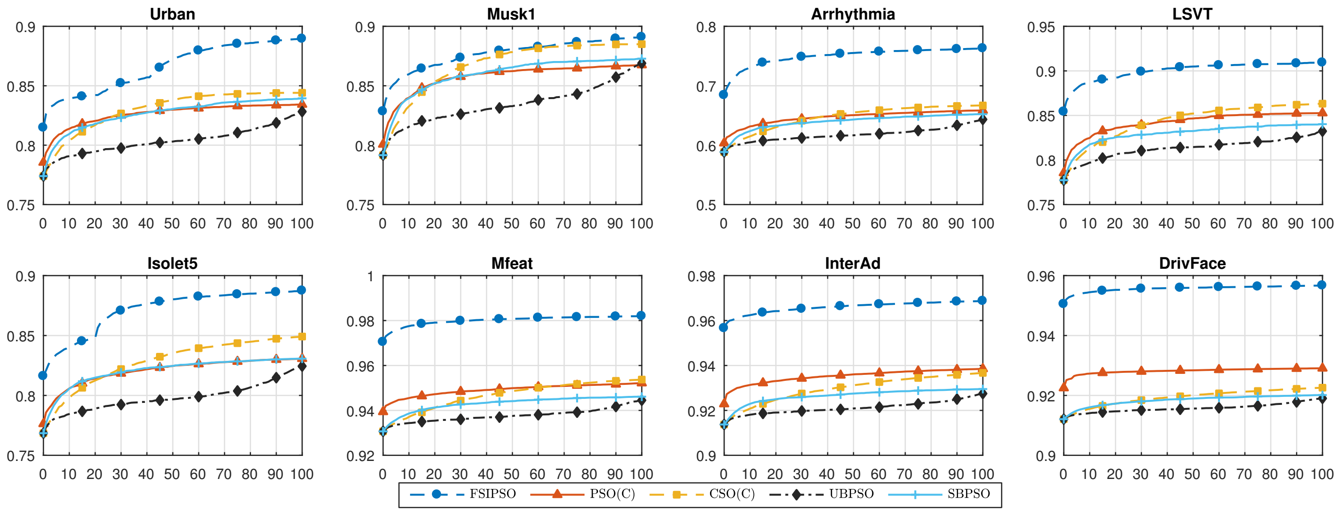

Fig. 2. Convergence curves of the PSO algorithms.

[2] B. Xue, M. Zhang, W. N. Browne, and X. Yao, "A survey on evolutionary computation approaches to feature selection," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 4, pp. 606–626, 2016.

[3] L. Yu and H. Liu, "Efficient feature selection via analysis of relevance and redundancy," *Journal of Machine Learning Research*, vol. 5, no. Oct., pp. 1205–1224, 2004.

[4] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1, pp. 273 – 324, 1997.

[5] I. Oh, J. Lee, and B. R. Moon, "Hybrid genetic algorithms for feature selection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, 2004.

[6] A.-D. Li, B. Xue, and M. Zhang, "Multi-objective feature selection using hybridization of a genetic algorithm and direct multisearch for key quality characteristic selection," *Information Sciences*, vol. 523, pp. 245 – 265, 2020.

[7] B. Xue, M. Zhang, and W. N. Browne, "Particle swarm optimization for feature selection in classification: A multi-objective approach," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1656–1671, 2013.

[8] ——, "Particle swarm optimisation for feature selection in classification: Novel initialisation and updating mechanisms," *Applied Soft Computing*, vol. 18, pp. 261 – 276, 2014.

[9] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[10] B. H. Nguyen, B. Xue, P. Andreae, and M. Zhang, "A new binary particle swarm optimization approach: Momentum and dynamic balance between exploration and exploitation," *IEEE Transactions on Cybernetics*, pp. 1–15, 2019.

[11] B. Tran, B. Xue, and M. Zhang, "A new representation in PSO for discretization-based feature selection," *IEEE Transactions on Cybernetics*, vol. 48, no. 6, pp. 1733–1746, 2018.

[12] S. F. Pratama, A. K. Muda, Y. Choo, and N. A. Muda, "PSO and computationally inexpensive sequential forward floating selection in acquiring significant features for handwritten authorship," in *2011 11th International Conference on Hybrid Intelligent Systems (HIS)*, Dec 2011, pp. 358–363.

[13] S. Gu, R. Cheng, and Y. Jin, "Feature selection for high-dimensional classification using a competitive swarm optimizer," *Soft Computing*, vol. 22, no. 3, pp. 811–822, Feb 2018.

[14] A.-D. Li, B. Xue, and M. Zhang, "Improved binary particle swarm optimization for feature selection with new initialization and search space reduction strategies," *Applied Soft Computing*, vol. 106, p. 107302, 2021.

[15] Y. Xue, W. Jia, and A. X. Liu, "A particle swarm optimization with filter-based population initialization for feature selection," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 1572–1579.

[16] B. Tran, B. Xue, and M. Zhang, "Variable-length particle swarm optimization for feature selection on high-dimensional classification," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 473–487, 2019.

[17] X. F. Song, Y. Zhang, Y. N. Guo, X. Y. Sun, and Y. L. Wang, "Variable-size cooperative coevolutionary particle swarm optimization for feature selection on high-dimensional data," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 5, pp. 882–895, 2020.

[18] Bing Xue, M. Zhang, and W. N. Browne, "New fitness functions in binary particle swarm optimisation for feature selection," in *2012 IEEE Congress on Evolutionary Computation*, 2012, pp. 1–8.

[19] B. H. Nguyen, B. Xue, and P. Andreae, "A novel binary particle swarm optimization algorithm and its applications on knapsack and feature selection problems," in *Intelligent and Evolutionary Systems*, G. Leu, H. K. Singh, and S. Elsayed, Eds. Cham: Springer International Publishing, 2017, pp. 319–332.

[20] B. Tran, B. Xue, and M. Zhang, "Adaptive multi-subswarm optimisation for feature selection on high-dimensional classification," ser. GECCO '19. New York, NY, USA: Association for Computing Machinery, 2019.

[21] H. Dong, Y. Pan, and J. Sun, "High dimensional feature selection method of dual gbest based on PSO," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, 2020, pp. 1–8.

[22] M. Gutlein, E. Frank, M. Hall, and A. Karwath, "Large-scale attribute selection using wrappers," in *2009 IEEE Symposium on Computational Intelligence and Data Mining*, 2009, pp. 332–339.

[23] B. Xue, M. Zhang, and W. N. Browne, "Single feature ranking and binary particle swarm optimisation based feature subset ranking for feature selection," in *Proceedings of the Thirty-fifth Australasian Computer Science Conference - Volume 122*, ser. ACSC '12. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2012, pp. 27–36.

[24] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[25] R. Cheng and Y. Jin, "A competitive swarm optimizer for large scale optimization," *IEEE Transactions on Cybernetics*, vol. 45, no. 2, pp. 191–204, 2015.

[26] J. Liu, Y. Mei, and X. Li, "An analysis of the inertia weight parameter for binary particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 5, pp. 666–681, 2016.

[27] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[28] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, Jan. 1991.

[29] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80 – 83, 1945.

[30] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data," *Knowledge and Information Systems*, vol. 34, no. 3, pp. 483–519, 2013.