Surrogate-model based Particle Swarm Optimisation with Local Search for Feature Selection in Classification

Hoai Bach Nguyen *, Bing Xue, and Peter Andreae

School of Engineering and Computer Science
Victoria University of Wellington
{Hoai.Bach.Nguyen, Bing.Xue, Peter.Andreae}@ecs.vuw.ac.nz

Abstract. Evolutionary computation (EC) techniques have been applied widely to many problems because of their powerful search ability. However, EC based algorithms are usually computationally intensive, especially with an expensive fitness function. In order to solve this issue, many surrogate models have been proposed to reduce the computation time by approximating the fitness function, but they are hardly applied to EC based feature selection. This paper develops a surrogate model for particle swarm optimisation based wrapper feature selection by selecting a small number of instances to create a surrogate training set. Furthermore, based on the surrogate model, we propose a sampling local search, which improves the current best solution by utilising information from the previous evolutionary iterations. Experiments on 10 datasets show that the surrogate training set can reduce the computation time without affecting the classification performance. Meanwhile the sampling local search results in a significantly smaller number of features, especially on large datasets. The combination of the two proposed ideas successfully reduces the number of features and achieves better performance than using all features, a recent sequential feature selection algorithm, original PSO, and PSO with one of them only on most datasets.

Keywords: Feature selection, particle swarm optimization, surrogate model, instance selection

1 Introduction

In classification, a set of training instances is used to build a classifier, which assigns a pre-defined class label to unseen instances. However in many classification problems, each instance is described by a large number of features, which causes difficulties to the training process due to the "curse of dimensionality". Therefore, feature selection is proposed to select a small number of features while maintain or even improve the classification performance. However, feature selection is not an easy task because of its huge search space, which increases exponentially with respect to the number of features. It is also challenging to capture complex interactions between features [1]. The two issues are usually handled by two main

* Corresponding Author

components, a search mechanism and an evaluation criterion, respectively. The search mechanism is used to generate candidate feature subsets, which are then evaluated by the evaluation criterion.

Based on the evaluation measure, feature selection methods can be divided into wrapper approaches and filter approaches [1]. In wrappers, a classification algorithm is used to measure the candidate feature subsets' performance, which often results in promising classification performance. However wrappers are usually computationally intensive and the selected feature subsets are only for a specific classification algorithm. Meanwhile, in filter approaches, feature subsets are evaluated based on the characteristics of data, which are captured by some filter measures such as information measure or correlation measure. Filters are usually faster and selects more general features than wrappers. However, in terms of classification accuracy, filter approaches often can not achieve as good results as wrappers.

In terms of the search mechanism, evolutionary computation (EC) techniques have been widely applied to feature selection because of their potential global search ability. Compared with other EC techniques like genetic algorithms (GAs), memetic algorithms, PSO evolves better solutions in more efficient computation time on many problems [2]. Therefore, this work uses PSO as the search mechanism to find optimal feature subsets.

A common problem of EC based algorithms is the expensive computation cost since the fitness evaluation is performed on many individuals. Surrogate models have been used to reduce the computation cost in many expensive problems [3]. The main idea is to partly replace the highly cost fitness function using its cheap estimation. Despite of being applied widely, surrogate models have seldom been applied to feature selection, which also has an expensive fitness evaluation. To the best of our knowledge, this work will be the first attempt to develop a surrogate model for PSO based wrapper feature selection.

Although PSO is a global search technique, it easily converges prematurely and stuck at local optima when applying to feature selection, whose search space is complex with many local optima. Furthermore, PSO only considers the current best solutions, while useful information from the previous generations might be discarded. Therefore, this work investigates on improving the evolved feature subsets by applying an efficient local search, which uses information of the best solutions from all iterations so far to prevent the premature convergence.

1.1 Goals

This paper aims to develop a new PSO based feature selection approach with the goal of maintaining or improving the classification accuracy of wrappers while significantly reducing the computational cost and the number of features. To achieve this goal, firstly a surrogate training set is built by applying an instance selection algorithm. The expectation is to reduce the computation cost and maintain the information of the whole training set by selecting a small number of informative instances. Based on the surrogate training set, to improve the quality of the current best solution, a local search is developed to utilise features selected by the best feature subsets in the previous iterations to sample new candidate feature subsets, which will compete with and might replace the current best solution. The proposed

approach will be evaluated and compared with a PSO based pure wrapper method and a sequential feature selection method. Specifically, we will investigate:

- whether the surrogate training set can reduce the computation cost while maintain or improve the performance over using the whole training set,
- whether the local search can assist PSO to evolve smaller feature subsets with similar or better classification accuracy, and
- whether combining the local search and the surrogate model can achieve better classification performance and scalability than using all instances and a state-of-the-art sequential feature selection algorithm.

2 Background

2.1 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) was originally proposed by Kennedy [4] to solve continuous problems. In PSO, an optimisation problem is solved by using a set of particles, called swarm, in which each particle represents a candidate solution. Particles move around the search space by updating their positions using velocities, which are based on their best position, called *pbest* and their neighbours's best position, named *gbest*. In order to solve binary problems, Sticky PSO [5] was proposed, which replaced the velocity vector by a probability vector. Each element of the probability vector represents the chance of flipping the corresponding position's bit. The probability element consists of three components: cognitive factor determined by *pbest*, social factor determined by *gbest*, and a stickiness property, which is a new binary momentum. The stickiness property is defined as a tendency to stay with the current position. It has the maximum value of 1 if its corresponding bit is just flipped to a new value, and decays in the following iterations if the bit's value is not changed. This idea is implemented by a variable called $currentLife$, which records the number of iterations after the bit was flipped. $maxLife$ is the upper limit of $currentLife$, which ensures that the stickiness is not negative. After a number of iterations without flipping, the bit's stickiness becomes 0. The position and flipping probability vector of a particle, denoted by x and p, are updated according to the following equations:

$$x_d^{t+1} = \begin{cases} 1, \text{ if } rand() < \frac{1}{1+e^{-v_d^{t+1}}} \\ 0, \text{ otherwise} \end{cases} \tag{1}$$

$$p_d = i_m * (1 - stickiness_d) + i_p * |pbest_d - x_d| + i_g * |gbest_d - x_d| \tag{2}$$

$$stickiness_d = 1 - \frac{currentLife_d}{maxLife} \tag{3}$$

where t is the t^{th} iteration, and d is the d^{th} dimension in the search space. i_m, i_p and i_g are used to control the proportions of the stickiness, cognitive and social components in the flipping probability, respectively. By redefining the velocity and momentum concepts, Sticky PSO can cope with binary search spaces to search more efficiently and effectively. Specifically, Sticky PSO can evolve better solutions than PBPSO [6], a state-of-the-art binary PSO, on two well-known types of binary problems: knapsack and feature selection. Therefore, Sticky PSO is selected as the search mechanism in this work.

2.2 Related Work On Feature Selection

Feature Selection Using Non-EC Techniques: Sequential forward selection (SFS) [7]/backward selection (SBS) [8] are two well-known traditional feature selection algorithms, which starts with an empty/full feature subset and incrementally adds/removes a feature which gives the best performance improvement. The process of adding/removing continues until the performance can not be improved. Although these techniques are more efficient than considering all possible feature subsets, they suffer the "nesting" problem. Specifically, once a feature is added or removed from the feature subset, it can not be removed or added later. To overcome this issue, "plus-l-take away -r" is proposed in [9], which iteratively does l forward steps followed by r backward steps. The pair (l,r) is determined dynamically in [10]. Later, Nakariyakul et al. [11] propose a method named IFFS which further improves the sequential searches using an additional step replacing weak features in the current feature subset by an unselected feature. The experimental results show that IFFS achieves better performance than other sequential search algorithms. Therefore, IFFS is chosen to compare against our proposed method.

PSO based Feature Selection: Many works attempt to improve PSO based feature selection algorithms by modifying initialisation, representation or fitness function. In [12], an opposition chaotic method is applied to improve the initialisation of PSO. Specifically, two candidate feature subsets are generated on two opposite sides of the search space and the better side is used to initialise a particle in the swarm. In addition, opposition chaotic also assists to define PSO parameters dynamically and update *gbest* to avoid being stagnation in local optima. A representation of PSO is proposed by Vieira et al. [13] to simultaneously optimise support vector machine's parameters and select a feature subset. However, the length of this representation is longer than the traditional one since it needs extra bits for the parameters. The results indicate that the proposed representation achieves better performance than other binary PSO and GAs based feature selection algorithms. Based on statistical feature clustering, Nguyen et al. [14] propose a representation which is shorter than the traditional one. Firstly, similar features are grouped into the same feature cluster. For each feature cluster, a maximum number of selected features from the cluster are predefined. Each position's bit belongs to a cluster and indicates which feature is selected from that cluster. The experimental results show that the proposed algorithm selects a smaller number of features while improves the classification performance over two other PSO based algorithms.

Premature convergence is a typical problem of PSO based feature selection algorithms. To address this problem, a *gbest* resetting mechanism is proposed in [15]. Specifically, if the *gbest* is not changed for a certain number of iterations, its position entries are reset to 0. This mechanism is utilised in [16] along with a local search to simultaneously reduce both classification error and the size of feature subsets. In addition, to speed up the fitness calculation, the proposed algorithm only considers the changed features. In [17], PSO is even used with GAs to solve feature selection problems, where in each iteration, the swarm is divided into two parts which are enhanced by PSO and GAs, respectively. The combination of two EC algorithms enhances the population variety, which results in informative feature subsets. Nguyen et al. [18] aim to improve *gbest* by applying a local search called

filter based backward elimination. Features selected by the current *gbest* are ranked by mutual information. The elimination process mimics the backward selection to remove the less informative and redundant features. The proposed algorithm reduces the number of features and achieves significantly better performance than other recent PSO based algorithms. A comprehensive survey on EC based feature selection is provided by Xue et al. [1].

There have been many works attempting to improve the efficiency of PSO based feature selection algorithms by modifying the representation and fitness function. Recently, the surrogate model is applied to many EC techniques to reduce the computation time. However, it has never been applied to PSO based feature selection algorithms. This will be the first time that the surrogate model is investigated in PSO based feature selection. In addition, the current PSO based feature selection algorithms do not use the feature subsets selected by *gbest* in previous generations, which might contain information about good features.

3 Proposed Feature Selection Approach

Wrappers usually achieve high classification performance feature subsets with an intensive computation cost. To alleviate the problem, we propose a surrogate model and a local search to improve the PSO based wrapper feature selection algorithm.

3.1 Surrogate Model For Feature Selection

Surrogate Training Set: A large training set is one of the reasons for a long classification process i.e. long time for wrapper feature selection. There might be some noisy instances which may deteriorate the classification performance. In order to alleviate the expensive computation cost and at least maintain the accuracy, an instance selection algorithm can be used to select a small number of representative instances to form a new instance set, called a surrogate training set, which is expected to contain all essential information from the original training set. The fitness function calculated on the *surrogate training set* is called a *surrogate fitness function*, which can be considered an estimation of the original fitness function.

K nearest neighbours (KNN) is a simple and powerful instance based classification algorithm [19], so it is chosen to evaluate feature subsets. Wilson et al. [20] propose several instance selection algorithms for KNN. Among the proposed algorithms, it has been shown in [21] that DROP3 achieves good performance with respect to KNN. In general, DROP3 reduces the number of instances by removing central instances and retaining border instances. The main reason is that the internal instances do not affect on the decision boundaries as much as the border points. Therefore, removing the central instances affects the classification performance less than removing the border instances.

DROP3 starts with filtering out all noisy instances, which are assigned to wrong class labels by their K nearest neighbours. This step also removes instances in the middle of two or more class boundaries, which creates a smoother decision boundary. For each remaining instance, there is an "enemy" instance, which is the closest instance with a different class label. The distance from an instance to its "enemy" instance is called an "enemy" distance. Therefore, the larger "enemy" distance an instance has, the farther the instance is to its class label boundary. All

remaining instances will be sorted according to their "enemy" distances so that the internal instances, which are far from their class label boundary, are removed first. For each instance, its removing process relates to its associated instances, which have the instance as one of their K nearest neighbours. If removing the instance does not reduce the number of correctly classified associated instances, then the instance will be removed. The set of selected instances is used as the *surrogate training set* for KNN in this work.

Surrogate Training Process: Surrogate models for fitness evaluations can be roughly divided into three categories: individual based, generation based and population based [22]. In individual based approaches, some individuals from the population are calculated by using the original fitness function and other individuals are evaluated by the surrogate fitness function. Population based approaches have more than one sub-populations, which might use different surrogate fitness functions. The communication and exchanging individuals from different sub-populations are allowed. For generation based, the surrogate fitness function is used in some of the generations before the original fitness function is used in the rest of the generations. In PSO, the swarm starts by exploring the search space to locate promising areas, which are then exploited in the later iterations. So it can be said that in early iterations the swarm tries to estimate the possible regions of global optima. In the sense of estimation, it would be safe to use the surrogate model at the beginning of a PSO algorithm to locate promising areas before using the original fitness function to find out the exact optima. Therefore, the idea of generation based approach is suitable here. Specifically, the particles are evaluated using the surrogate training set in the first I_s iterations, while in rest of the iterations, the whole training set is used. Given I is the maximum number of iterations, the task is to figure out the value of I_s iterations so that the classification performance is still maintained or even improved over using the original fitness function. The rate between I_s and I is called the surrogate rate α_s, i.e. $\alpha_s = \frac{I_s}{I}$.

3.2 Local Search: Sampling On *gbest*

In a PSO algorithm, the main idea is to use the current *gbest* and *pbest* to guide the particles to follow promising trajectories. However, the *gbest* from the previous generations might contain some useful information, which can assist the swarm to achieve better solutions. For instance, in feature selection, features which appear in *gbest* for many iterations tend to be good features. Moreover, since feature selection has a large and complex search space, some good features might not be selected together in the *gbest* solutions. These features might be complementary features, which provide even more information about the class label when appearing in one feature subset. Therefore the main idea of the local search is to keep features selected on all *gbest* and use them to improve the current *gbest*.

Suppose that S_{best} is the set of features which are selected in all *gbest* from previous iterations. Each feature from S_{best} has a score (explained later and shown in Eq. 4). The local search constructs $\frac{P}{2}$ candidate solutions by using S_{best}, where P is the population size. $|gbest|$, the number of features in the current *gbest*, is the maximum number of features in each candidate feature subsets. Specifically, based on the calculated scores, a tournament selection is used to select $|gbest|$

features from S_{best}, which form a sampled feature subset. The higher a feature's score is, the more chance that feature is selected. In addition, in $|gbest|$ selected features, there might be some duplicated features, which means that the size of the sampled feature subset can be less than $|gbest|$. All sampled feature subsets are then compared with each other based on their surrogate fitness values to find the best candidate feature subset. In this way, the local search utilises the surrogate training set to approximate a good solution in a short time. After that, the best candidate subset and the current *gbest* compete on the current fitness function, $fitness_{cur}$, which is the surrogate fitness function in the first I_s iterations or the original fitness function in the last $(I - I_s)$ iterations. The winner becomes *gbest*.

The task now is to define the scores of features in S_{best}. The main idea is to give more scores to features, which are selected more frequently and recently by *gbest*. The first component of the score is $freq$, which measures the number of iterations, in which a feature is selected in *gbest*. The higher $freq$ a feature has, the better the feature's quality. Before contributing to the score, $freq$ is normalised to $freq_n$ so that the total $freq_n$ of all features from S_{best} is 1. The second component of the score is related to the current *gbest*, called *gc*. If a feature in S_{best} is also selected in the current *gbest*, its *gc* is set to $\frac{1}{|gbest|}$. Otherwise its *gc* is 0. The score of the f^{th} feature in S_{best} is the sum of its $freq_n$ and *gc*, which can be seen in Eq. (4).

$$score_f = freq_{nf} + gc_f \tag{4}$$

where

$$freq_{nf} = \frac{freq_f}{\sum_{f=1}^{|S_{best}|} freq_f}$$

$$gc_f = \begin{cases} \frac{1}{|gbest|} \text{ if } f \in \ gbest \\ 0 \text{ otherwise} \end{cases}$$

As illustrated in Eq. (4), given the same $freq_n$, the *gc* component gives more scores to features which are recently selected. In addition, since the best sampled subset might replace the current *gbest*, it is preferred to keep the sampled subset close to the current *gbest* so that the swarm is not distracted. The *gc* component lets the features from the current *gbest* to have more chance to be chosen, which allows to build new feature subsets not far from the current *gbest*. The local search is applied after the *gbest* of the current iteration is determined and before that *gbest* is informed to all particles.

3.3 Overall Algorithm

In this work, Sticky PSO, a binary PSO algorithm, is applied to achieve feature selection, in which each position entry corresponds to one original feature. The value 1 of a position entry indicates that the corresponding feature is selected. Otherwise the corresponding feature is not selected. In this work, a feature subset is evaluated by the fitness function given in Eq. (5)

$$fitness = \gamma * ClassificationError + (1 - \gamma)\frac{\#\text{selected features}}{\#\text{all features}} \tag{5}$$

where PSO needs to minimise the classification error determined by a classification algorithm and the number of selected features. The proportions of the two objectives are controlled by γ. If the classification algorithm is trained on the surrogate

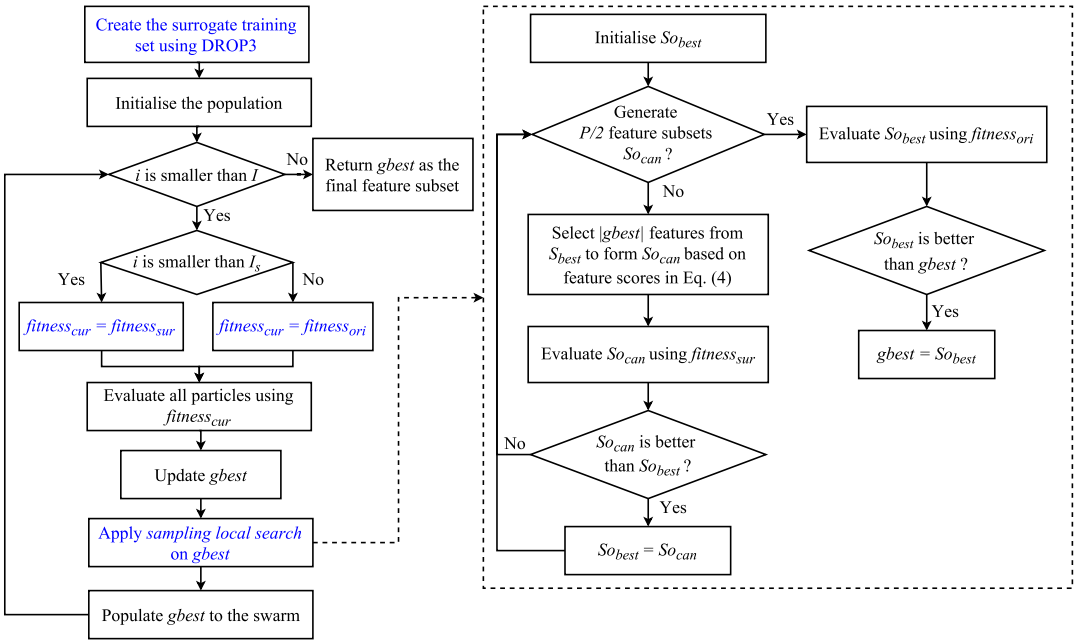

Fig. 1: Overall algorithm

training set, the corresponding fitness function is a surrogate fitness function, denoted by $fitness_{sur}$. If the whole training set is used to build the classifier then the fitness function is the original one, called $fitness_{ori}$. In terms of PSO's representation, each feature corresponds to a position entry, whose value is either 1 or 0, indicating the feature is selected or not selected, respectively. The overall proposed algorithm, named SurSammPSO, is described in Fig. 1, in which the contribution of this work is marked in blue. So_{can} is a sampled feature subset, So_{best} is the best So_{can} generated by the sampling process, and P is the swarm size.

4 Experiment Design

4.1 Datasets

Ten datasets (Table 1) with various numbers of features, classes and instances chosen from the UCI machine learning repository [23] are used in the experiments. Each dataset is randomly divided into two parts for training and test purpose, which preserve the original class distribution and contain 70% and 30% instances, respectively. DROP3 [20] is applied on the training set to create the surrogate training set. During the training process, a 10-fold cross validation is used on the training set or surrogate training set to calculate the classification accuracy. On each dataset, feature selection algorithms are executed 40 independent times.

4.2 Parameter Settings

A KNN classification algorithm is used to classify instances, where K=3, which is recommended in [20]. The weight γ in Eq. (5) is set to 0.9 so that the search process focuses more on the classification performance than the number of features.

Table 1: Datasets.

Dataset	#features	#classes	#instances
Vehicle	18	4	496
German	24	2	1000
Ionosphere	34	2	351
Lung	56	2	32
Sonar	60	2	208
Movementlibras	90	15	360
Plant	64	100	1600
Hillvalley	101	2	1213
LSVT	310	2	126
Multiple Features	649	10	2000

For sticky PSO, i_m, i_p and i_g are set to 0.1154, 0.4423 and 0.4423, respectively, which ensures that *pbest* and *gbest* contribute more to a particle's movement than *momentum*. $maxLife$ is set to 40. The population size is equal to the number of features and limited by 100. The maximum number of iterations is 100.

In order to find out the best α_s, different values of α_s ranging from 0.0 to 1.0 with a step of 0.25 are tested on different datasets. The value of 0.0 or 1.0 mean that the surrogate training set is never or always used, respectively. According to the results, $\alpha_s = 0.75$ gives the similar or better classification performance with the shortest running time. So the value of 0.75 is set to α_s in this work, which is given 100 iterations, the surrogate training set is used in the first 75 iterations.

The PSO based feature selection algorithm, which uses purely the original fitness function, is called OriPSO. SurPSO applies the surrogate training set in the first 75 iterations while OriSamPSO uses the sampling local search in OriPSO. The combination of the local search and the surrogate training model results in SurSamPSO. The classification accuracies of different algorithms are compared by a statistical significance test called Wilcoxon test with significance level set to 0.05.

5 Experiment Results

This section presents the results of SurPSO, OriSamPSO and SurSamPSO. In all tables, NF, *Ave* and *Std* stand for the average number of features, the average and standard deviation of the classification accuracy over 40 independent runs. T shows the significant test results. *All* means using all features.

5.1 Effect of the Surrogate Training Set

To analyse the effect of the surrogate training set, SurPSO will be compared with OriPSO, which only uses the original fitness function to evaluate feature subsets. The comparison can be seen from Table 2, in which "+"/"-"/"=" indicate that OriPSO or using all features are significantly better/worse/similar to SurPSO.

In comparison with using all features, SurPSO selects from 5% to 20% of the original features while still achieves significantly better test performance on 8 out of the 10 datasets. For example, on the LSVT dataset, SurPSO reduces more than 90% of the features and achieves around 1% better accuracy than using all features.

As can be seen from the table, in terms of training accuracy, SurPSO achieves similar or better performance than OriPSO on 7 out of the 10 datasets. SurPSO's test accuracies are better than that of OriPSO on 2 datasets. For example, on the Hillvalley dataset, SurPSO selects 2.5 features fewer than OriPSO, but its classification performance is about 1.1% better. On the rest of the datasets, SurPSO

Table 2: Results of OriPSO and SurPSO

Dataset	NF	Training Results		Test Results		Time (ms)
		AveTrain ± Std	T	AveTest ± Std	T	
Vehicle						
All	18.0	89.44 ± 0.00	-	83.07 ± 0.00	-	
OriPSO	4.8	91.41 ± 0.69	=	84.66 ± 1.29	-	113187.62
SurPSO	5.1	91.43 ± 0.69		85.23 ± 0.82		34294.93
German						
All	24.0	83.14 ± 0.00	-	65.33 ± 0.00	-	
OriPSO	5.6	78.70 ± 0.52	-	67.63 ± 1.90	=	229110.30
SurPSO	6.4	84.33 ± 0.52		68.35 ± 0.75		60600.90
Ionosphere						
All	34.0	90.24 ± 0.00	-	86.67 ± 0.00	-	
OriPSO	4.8	95.27 ± 0.81	+	88.48 ± 2.12	=	33564.40
SurPSO	4.0	94.64 ± 0.81		88.38 ± 2.61		9903.90
Lung						
All	56.0	86.36 ± 0.00	-	80.00 ± 0.00	=	
OriPSO	5.1	99.89 ± 1.73	+	79.50 ± 4.44	=	707.38
SurPSO	4.4	99.20 ± 1.73		79.50 ± 5.45		451.75
Sonar						
All	60.0	88.97 ± 0.00	-	84.13 ± 0.00	+	
OriPSO	14.8	95.55 ± 1.78	+	81.94 ± 3.32	=	23605.65
SurPSO	12.4	93.55 ± 1.78		82.58 ± 3.14		7291.62
Movementlibras						
All	90.0	98.52 ± 0.00	-	95.06 ± 0.00	-	
OriPSO	9.2	98.54 ± 0.16	=	95.33 ± 0.41	=	105780.20
SurPSO	9.4	98.63 ± 0.16		95.29 ± 0.32		41165.30
Plant						
All	64.0	99.55 ± 0.00	+	99.10 ± 0.00	+	
OriPSO	3.0	99.24 ± 0.04	=	98.67 ± 0.03	=	1756442.18
SurPSO	3.1	99.26 ± 0.04		98.68 ± 0.05		566479.72
Hillvalley						
All	100.0	79.83 ± 0.00	-	59.07 ± 0.00	-	
OriPSO	24.8	81.54 ± 0.83	=	58.85 ± 1.88	-	1557361.75
SurPSO	22.3	81.16 ± 0.83		59.92 ± 1.46		443232.00
LSVT						
All	310.0	79.55 ± 0.00	-	55.26 ± 0.00	-	
OriPSO	27.9	85.45 ± 4.72	=	65.53 ± 1.53	=	18240.05
SurPSO	27.4	85.11 ± 4.72		65.07 ± 4.89		4708.20
Multiple Features						
All	649.0	99.49 ± 0.00	-	98.57 ± 0.00	-	
OriPSO	118.2	99.66 ± 0.05	=	99.04 ± 0.10	=	7203332.90
SurPSO	143.6	99.65 ± 0.05		99.05 ± 0.12		2068439.52

achieves similar test performance in comparison with OriPSO. In terms of the number of selected features, except for the last dataset with a large number of features, the size of feature subsets selected by the two algorithms are roughly equal. Despite of maintaining or improving the classification performance, SurPSO's evolutionary processes are much shorter than that of OriPSO. Specifically, on most of datasets SurPSO spends 70% less time than OriPSO to evolve a feature subset. Although OriPSO is already quite fast on the Lung dataset (only 707 ms), SurPSO is still able to reduce about 40% of the OriPSO's computation time. So the surrogate training set significantly reduces the computation cost without deteriorating the test performance. On some datasets, SurPSO even improves the performance due to the DROP3's intention to remove noisy instances.

5.2 Effect of the Sampling Local Search

To analyse the effect of the proposed local search, OriSamPSO, which applies the local search to the original fitness function is compared with using all features and OriPSO. The comparisons are illustrated in Table 3.

Table 3: Results of OriPSO and OriSamPSO

Dataset	NF	Training Results		Test Results		Time (ms)
		AveTrain ± Std	T	AveTest ± Std	T	
Vehicle						
All	18.0	89.44 ± 0.00	-	83.07 ± 0.00	-	
OriPSO	4.8	91.41 ± 0.73	=	84.66 ± 1.29	=	113187.62
OriSamPSO	4.7	91.40 ± 0.73		84.83 ± 1.10		117337.07
German						
All	24.0	83.14 ± 0.00	+	65.33 ± 0.00	-	
OriPSO	5.6	78.70 ± 4.90	=	67.63 ± 1.90	=	229110.30
OriSamPSO	5.4	79.26 ± 4.90		67.33 ± 1.98		226021.25
Ionosphere						
All	34.0	90.24 ± 0.00	-	86.67 ± 0.00	-	
OriPSO	4.8	95.27 ± 0.42	=	88.48 ± 2.12	+	33564.40
OriSamPSO	4.3	95.19 ± 0.42		87.31 ± 1.68		39497.60
Lung						
All	56.0	86.36 ± 0.00	-	80.00 ± 0.00	=	
OriPSO	5.1	99.89 ± 0.00	=	79.50 ± 4.44	=	707.38
OriSamPSO	4.5	100.00 ± 0.00		80.75 ± 2.63		833.70
Sonar						
All	60.0	88.97 ± 0.00	-	84.13 ± 0.00	+	
OriPSO	14.8	95.55 ± 1.54	=	81.94 ± 3.32	=	23605.65
OriSamPSO	14.6	95.60 ± 1.54		81.83 ± 3.27		23992.17
Movementlibras						
All	90.0	98.52 ± 0.00	=	95.06 ± 0.00	-	
OriPSO	9.2	98.54 ± 0.20	=	95.33 ± 0.41	=	105780.20
OriSamPSO	9.0	98.55 ± 0.20		95.21 ± 0.36		107488.62
Plant						
All	64.0	99.55 ± 0.00	+	99.10 ± 0.00	+	
OriPSO	3.0	99.24 ± 0.02	-	98.67 ± 0.03	=	1756442.18
OriSamPSO	3.0	99.25 ± 0.02		98.67 ± 0.04		1703456.68
Hillvalley						
All	100.0	79.83 ± 0.00	-	59.07 ± 0.00	=	
OriPSO	24.8	81.54 ± 1.01	=	58.85 ± 1.88	=	1557361.75
OriSamPSO	23.3	81.81 ± 1.01		59.48 ± 1.46		1553409.23
LSVT						
All	310.0	79.55 ± 0.00	-	55.26 ± 0.00	-	
OriPSO	27.9	85.45 ± 2.75	-	65.53 ± 1.53	=	18240.05
OriSamPSO	38.1	88.38 ± 2.75		66.51 ± 4.40		19069.62
Multiple Features						
All	649.0	99.49 ± 0.00	-	98.57 ± 0.00	-	
OriPSO	118.2	99.66 ± 0.04	-	99.04 ± 0.10	=	7203332.90
OriSamPSO	59.8	99.67 ± 0.04		99.03 ± 0.12		6260769.17

As can be seen from the table, applying the sampling local search not only reduces the number of features on all datasets but also maintains or even improves the test accuracy on 8 out of the 10 datasets. Especially, on the largest dataset, Multiple Features, OriSamPSO selects less than 10% of the original features and still achieves better performance than using all features.

According to the significant test results, the local search maintains or improves training performance on all datasets. Especially, on the large datasets, the training accuracy is significantly improved, for example on the LSVT dataset, OriSamPSO achieves almost 3% better accuracy than OriPSO. Although test accuracies are not significantly better, the local search assists PSO to evolve smaller number of features on 8 out of the 10 datasets. This pattern is obvious in the largest dataset, Multiple Features, where OriSamPSO selects only 59.8 features, which is two times smaller than the feature subset evolved by OriPSO. Despite of spending time on the local search, OriSamPSO still has comparative computation time in comparison with OriPSO on all datasets. OriSamPSO is even more efficient on 4 out of 10 datasets, for instance on Multiple Features, OriSamPSO is 15% more

Table 4: Results of OriPSO and SurSamPSO

Dataset	NF	Training Results		Test Results		Time (ms)
		AveTrain ± Std	T	AveTest ± Std	T	
Vehicle						
All	18.0	89.44 ± 0.00	-	83.07 ± 0.00	-	
OriPSO	4.8	91.41 ± 0.46	-	84.66 ± 1.29	-	113187.62
SurSamPSO	5.0	91.79 ± 0.46		85.44 ± 0.72		36255.28
German						
All	24.0	83.14 ± 0.00	-	65.33 ± 0.00	-	
OriPSO	5.6	78.70 ± 0.38	-	67.63 ± 1.90	=	229110.30
SurSamPSO	6.5	84.44 ± 0.38		68.22 ± 0.68		69631.98
Ionosphere						
All	34.0	90.24 ± 0.00	-	86.67 ± 0.00	-	
OriPSO	4.8	95.27 ± 0.80	+	88.48 ± 2.12	=	33564.40
SurSamPSO	3.7	94.51 ± 0.80		88.93 ± 2.45		9867.20
Lung						
All	56.0	86.36 ± 0.00	-	80.00 ± 0.00	-	
OriPSO	5.1	99.89 ± 1.62	=	79.50 ± 4.44	-	707.38
SurSamPSO	4.2	99.32 ± 1.62		81.50 ± 4.21		559.83
Sonar						
All	60.0	88.97 ± 0.00	-	84.13 ± 0.00	+	
OriPSO	14.8	95.55 ± 1.45	+	81.94 ± 3.32	=	23605.65
SurSamPSO	11.6	94.46 ± 1.45		81.67 ± 2.58		8337.40
Movementlibras						
All	90.0	98.52 ± 0.00	-	95.06 ± 0.00	-	
OriPSO	9.2	98.54 ± 0.20	=	95.33 ± 0.41	=	105780.20
SurSamPSO	9.0	98.58 ± 0.20		95.29 ± 0.31		45424.00
Plant						
All	64.0	99.55 ± 0.00	+	99.10 ± 0.00	+	
OriPSO	3.0	99.24 ± 0.03	=	98.67 ± 0.03	=	1756442.18
SurSamPSO	3.0	99.25 ± 0.03		98.68 ± 0.04		647379.50
Hillvalley						
All	100.0	79.83 ± 0.00	-	59.07 ± 0.00	=	
OriPSO	24.8	81.54 ± 0.96	=	58.85 ± 1.88	=	1557361.75
SurSamPSO	19.7	81.24 ± 0.96		59.27 ± 1.99		458048.42
LSVT						
All	310.0	79.55 ± 0.00	-	55.26 ± 0.00	-	
OriPSO	27.9	85.45 ± 3.72	-	65.53 ± 1.53	-	18240.05
SurSamPSO	22.2	89.09 ± 3.72		70.39 ± 5.76		4883.50
Multiple Features						
All	649.0	99.49 ± 0.00	-	98.57 ± 0.00	-	
OriPSO	118.2	99.66 ± 0.04	=	99.04 ± 0.10	=	7203332.90
SurSamPSO	61.8	99.66 ± 0.04		99.07 ± 0.14		1674807.10

efficient than OriPSO. The main reason is that the local search is cheap since it mainly works on the surrogate training set. Meanwhile, it can reduce the number of features significantly, which leads to shorter calculation time for the original fitness function.

The experimental results show that the local search using the surrogate training set reduces the number of selected features significantly while maintains the classification performance. The main reason is that the sampling process generates feature subsets containing at most the same number of features as the current *gbest*. Therefore, if a sampling feature subset replaces the current *gbest*, it achieves similar or better fitness with a smaller number of features than the current *gbest*.

5.3 Combining Sampling Local Search and Surrogate Training Set

In the above sections, it can be seen that both surrogate model and local search reduces either the computation time or the number of features significantly while maintains or improves the classification performance. This section analyses the effect of combining them together in one algorithm called SurSamPSO. It is expected

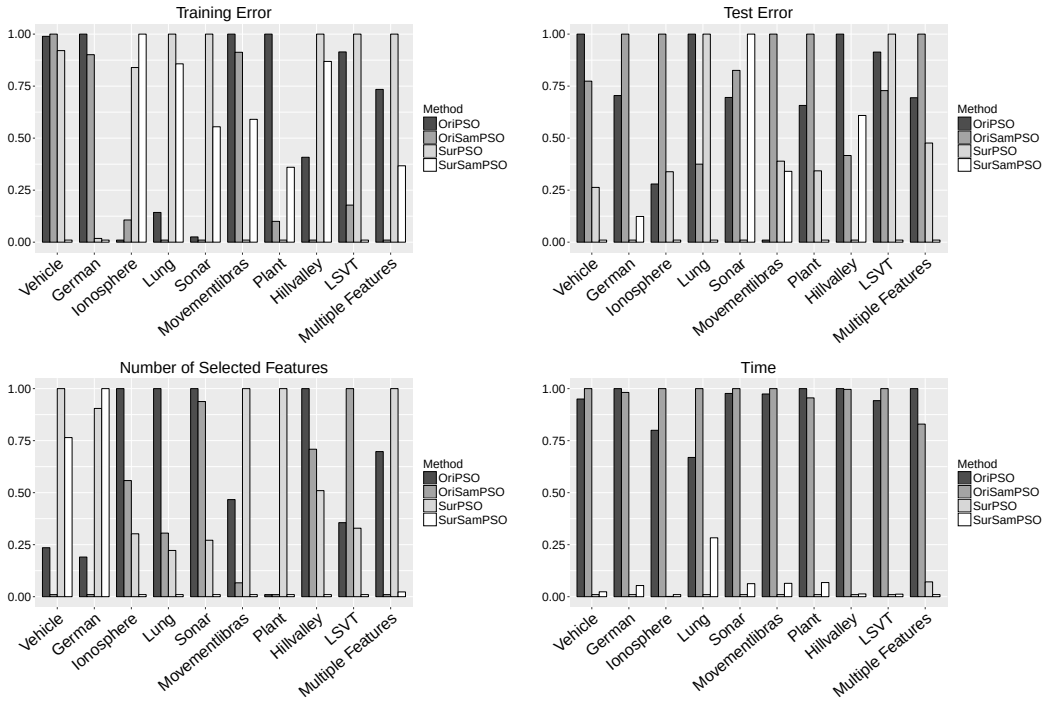

Fig. 2: Comparisons between all algorithms

that SurSamPSO can take the advantages of these two components. SurSamPSO is compared against OriPSO and using all features, which is shown in Table 4.

According to the experimental results, SurSamPSO successfully reduces the number of features on all datasets while improves the classification accuracy over using all features on 7 out of the 10 datasets. Although using all features achieves better performance on two datasets, Sonar and Plant, SurSamPSO selects less than 20% and 5% of the original features, respectively.

As can be seen from the table, although being trained on the surrogate training set for 75 iterations, SurSamPSO still evolves comparative feature subsets in terms of the training accuracy, which is calculated on the whole training set. Particularly, SurSamPSO is significantly better on 3 datasets while maintaining the training performance on 5 datasets. The largest difference between training accuracies of the two algorithms is on LSVT where SurSamPSO is 3.5% more accurate. In terms of the test performance, SurSamPSO is always similar or better than OriPSO, especially on LSVT, SurSamPSO improves about 5% over OriPSO. The local search still maintains its effect on SurSamPSO, in which the number of features is reduced significantly on 8 out of the 10 datasets. This pattern is more obvious on the large dataset. For example on Multiple Features, SurSamPSO selects 50% less features than OriPSO. In comparison with OriPSO, SurSamPSO is able to reduce up to 70% computation time.

5.4 Overall Comparisons

The comparisons between OriPSO, OriSamPSO, SurPSO and SurSamPSO are summarised in Fig. 2. The comparisons are based on 4 criteria: training error, test

Table 5: Comparison between SurSamPSO and IFFS

Dataset	Method	NF	AveTest	Std	T	Time (ms)
Vehicle	IFFS	4.0	81.69			15923.00
	SurSamPSO	5.0	85.44	0.72	+	36255.28
German	IFFS	2.0	66.00			11312.00
	SurSamPSO	6.5	68.22	0.68	+	69631.98
Ionosphere	IFFS	4.0	86.67			8442.00
	SurSamPSO	3.7	88.93	2.45	+	9867.20
Lung	IFFS	2.0	90.00			30.00
	SurSamPSO	4.2	81.50	4.21	-	559.83
Sonar	IFFS	3.0	77.78			2143.00
	SurSamPSO	11.6	81.67	2.58	+	8337.40
Movementlibras	IFFS	6.0	94.32			114080.00
	SurSamPSO	9.0	95.29	0.31	+	45424.00
Plant	IFFS	3.0	98.67			258225.00
	SurSamPSO	3.0	98.68	0.04	+	647379.50
Hillvalley	IFFS	5.0	59.07			525402.00
	SurSamPSO	19.7	59.27	1.99	=	458048.42
LSVT	IFFS	3.0	63.16			3507.00
	SurSamPSO	22.2	70.39	5.76	+	4883.50
Multiple Features	IFFS	12.0	98.73			48614670.00
	SurSamPSO	61.8	99.07	0.14	+	1674807.10

error, number of selected features and computation time, which need to be minimised. For making an easy comparison, all values are normalised in the range [0,1], where 0 is the best value and 1 is the worst value. It can be seen that SurSamPSO achieves the best test accuracy on 7 out of the 10 datasets, which is followed by SurPSO with two times of being the best. Meanwhile OriSamPSO is not the best one on any dataset despite of using both original fitness function and the sampling local search. The reason might be the inconsistency between the fitness function of the local search and the main fitness function of PSO. In addition, it can be seen that both algorithms using the surrogate training set evolve more promising performance than the others with the whole training set. In other words, selecting instances helps to select more general features by removing noisy instances in the training set. Besides the test accuracy, SurSamPSO also shows its strength on the number of selected features criterion. Specifically, SurSamPSO selects the least number of features on 7 out of the 10 datasets. OriSamPSO achieves the smallest number of features on 4 datasets. These results illustrate that the sampling local search is good at reducing the number of features. Filtering out instances also helps to reduce the number of features since the small instance set might require fewer features to classify the instances correctly. The last but not least criterion is the computation time, where SurPSO is the fastest algorithm on 9 out of the 10 datasets. SurPSO loses its first position to SurSamPSO on Multiple Features. As can be seen from Fig. 2, SurSamPSO follows SurPSO quite closely. It is even faster than SurPSO on Multiple Features because it selects much fewer features.

Overall the combination of the surrogate model and the sampling local search results in a more efficient, effective and scalable feature selection algorithm.

5.5 Comparison with IFFS

The comparison between SurSamPSO and IFFS is shown in Table 5. In the table, "+"/"-"/"=" mean that SurSamPSO is significantly better/worse/similar to IFFS. As can be seen from the table, SurSamPSO achieves better performance on 8 out of the 10 datasets. The reason is that IFFS usually selects a small number of

features since it stops when adding more features does not improve the fitness value. Although IFFS attempts to capture more feature interactions than other traditional sequential methods, it still considers only one feature at a step, which cause difficulty to find out blocks of complimentary features. Furthermore, IFFS is not scalable with respect to the number of features. This can be seen on the Multiple Features dataset, where SurSamPSO is about 30 times faster while still achieves better classification performance.

6 Conclusions and Future Work

The goal of this paper was to develop a PSO based feature selection algorithm, which could select a small number of features in an efficient way while maintaining or improving the classification performance. This goal has been achieved firstly by applying an instance selection algorithm to create a surrogate training set, which is smaller than the original training set. In the first 75% of the selection process, feature subsets are evaluated on the surrogate set to reduce the computation time. In addition to the surrogate training set, a sampling local search is designed to utilise all features selected by *gbest* so far to improve the current *gbest*. The results show that the surrogate training set can significantly reduce the computation time while maintain or even improve the classification performance. The sampling local search assists PSO to evolve a much smaller number of features with similar or better classification performance. The combination of the two components leads to a more effective and efficient PSO based algorithm, especially on datasets with a large number of features. In comparison with a recent sequential feature selection method, the proposed algorithm is better at balancing between the classification performance and the number of selected features. Especially on the largest dataset, the proposed algorithm is even more efficient than the improved sequential algorithm.

From the results, it is evident that the local search focuses more on reducing the number of features while mainly maintaining the classification accuracy. In the future, we will further improve the balance between the two objectives so that the local search can significantly enhance the classification performance. We will also investigate more on the relationship between fitness landscapes of the surrogate and original fitness functions to not only shorten the computation time but also strengthen the classification ability.

References

1. Xue, B., Zhang, M., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. IEEE Transactions on Evolutionary Computation **20**(4) (2016) 606–626
2. Hu, M., Wu, T., Weir, J.D.: An adaptive particle swarm optimization with multiple adaptive methods. IEEE Transactions on Evolutionary Computation **17**(5) (2013) 705–720
3. Tabatabaei, M., Hakanen, J., Hartikainen, M., Miettinen, K., Sindhya, K.: A survey on handling computationally expensive multiobjective optimization problems using surrogates: non-nature inspired methods. Structural and Multidisciplinary Optimization **52**(1) (2015) 1–25

4. Kennedy, J.: Particle swarm optimization. In: Encyclopedia of machine learning. Springer (2011) 760–766
5. Nguyen, B.H., Xue, B., Andreae, P.: A novel binary particle swarm optimization algorithm and its applications on knapsack and feature selection problems. In: Intelligent and Evolutionary Systems: The 20th Asia Pacific Symposium, IES 2016, Canberra, Australia, November 2016, Proceedings, Springer (2017) 319–332
6. Xue, B., Nguyen, S., Zhang, M. In: A New Binary Particle Swarm Optimisation Algorithm for Feature Selection. Springer Berlin Heidelberg (2014) 501–513
7. Whitney, A.W.: A direct method of nonparametric measurement selection. IEEE Transactions on Computers **100**(9) (1971) 1100–1103
8. Marill, T., Green, D.M.: On the effectiveness of receptors in recognition systems. IEEE Transactions on Information Theory **9**(1) (1963) 11–17
9. Stearns, S.D.: On selecting features for pattern classifiers. In: Proceedings of the 3rd International Conference on Pattern Recognition (ICPR), Coronado, CA (1976) 71–75
10. Pudil, P., Novovičová, J., Kittler, J.: Floating search methods in feature selection. Pattern recognition letters **15**(11) (1994) 1119–1125
11. Nakariyakul, S., Casasent, D.P.: An improvement on floating search algorithms for feature subset selection . Pattern Recognition **42**(9) (2009) 19321940
12. Bharti, K.K., Singh, P.K.: Opposition chaotic fitness mutation based adaptive inertia weight bpso for feature selection in text clustering. Applied Soft Computing **43** (2016) 20–34
13. Vieira, S.M., Mendonça, L.F., Farinha, G.J., Sousa, J.M.: Modified binary PSO for feature selection using svm applied to mortality prediction of septic patients. Applied Soft Computing **13**(8) (2013) 3494–3504
14. Nguyen, H.B., Xue, B., Liu, I., Zhang, M.: PSO and statistical clustering for feature selection: A new representation. In: Simulated Evolution and Learning. Springer (2014) 569–581
15. Chuang, L.Y., Chang, H.W., Tu, C.J., Yang, C.H.: Improved binary PSO for feature selection using gene expression data. Computational Biology and Chemistry **32**(1) (2008) 29–38
16. Tran, B., Xue, B., Zhang, M.: Improved PSO for feature selection on high-dimensional datasets. In: Simulated Evolution and Learning. Springer (2014) 503–515
17. Ghamisi, P., Benediktsson, J.A.: Feature selection based on hybridization of genetic algorithm and particle swarm optimization. IEEE on Geoscience and Remote Sensing Letters **12**(2) (2015) 309–313
18. Nguyen, H., Xue, B., Liu, I., Zhang, M.: Filter based backward elimination in wrapper based PSO for feature selection in classification. In: IEEE Congress on Evolutionary Computation (CEC 2014). 3111–3118
19. Liu, H., Zhang, S., Zhao, J., Zhao, X., Mo, Y.: A new classification algorithm using mutual nearest neighbors. In: Ninth International Conference on Grid and Cloud Computing (2010). 52–57
20. Wilson, D.R., Martinez, T.R.: Reduction techniques for instance-based learning algorithms. Machine Learning **38**(3) (2000) 257–286
21. Olvera-López, J.A., Carrasco-Ochoa, J.A., Martínez-Trinidad, J.F., Kittler, J.: A review of instance selection methods. Artificial Intelligence Review **34**(2) (2010) 133–143
22. Jin, Y.: Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm and Evolutionary Computation **1**(2) (2011) 61 – 70
23. Lichman, M.: UCI machine learning repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Sciences (2013)