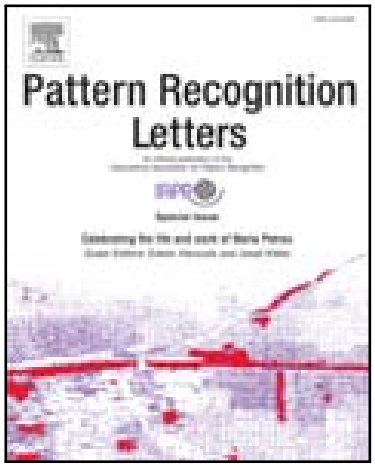
Journal Pre-proof



A permutational-based Differential Evolution algorithm for feature subset selection

Rafael Rivera-López, Efrén Mezura-Montes, Juana Canul-Reich, Marco Antonio Cruz-Chávez

PII: S0167-8655(20)30060-X
DOI: https://doi.org/10.1016/j.patrec.2020.02.021
Reference: PATREC 7799

To appear in: *Pattern Recognition Letters*

Received date: 31 October 2019
Revised date: 29 January 2020
Accepted date: 20 February 2020

Please cite this article as: Rafael Rivera-López, Efrén Mezura-Montes, Juana Canul-Reich, Marco Antonio Cruz-Chávez, A permutational-based Differential Evolution algorithm for feature subset selection, *Pattern Recognition Letters* (2020), doi: https://doi.org/10.1016/j.patrec.2020.02.021

Highlights

- The integer-valued vector representing a candidate solution encodes all dataset features.
- The feature subset size is determined by using an additional element in the vector.
- Feature relevance is represented through its relative position in the vector.
- A permutation-matrix-based mutation operator can be used to create new solutions.
- A simple repair-based recombination operator is used to ensure feasible solution creation.

Pattern Recognition Letters
journal homepage: www.elsevier.com

A permutational-based Differential Evolution algorithm for feature subset selection

Rafael Rivera-López[a,**], Efrén Mezura-Montes[b], Juana Canul-Reich[c], Marco Antonio Cruz-Chávez[d]

[a]*Departamento de Sistemas y Computación, Instituto Tecnológico de Veracruz, M. A. de Quevedo 2779, Col. Formando Hogar, 91800, Veracruz, VER, México*
[b]*Centro de Investigación en Inteligencia Artificial, Universidad Veracruzana, Sebastián Camacho 5, Centro, 91000, Xalapa-Enrquez, VER, México*
[c]*División Académica de Ciencias y Tecnologías de la Información, Universidad Juárez Autónoma de Tabasco, Km. 1 Carretera Cunduacán-Jalpa de Méndez, Col. La Esmeralda, 86690, Cunduacán, TAB, México*
[d]*Centro de Investigación en Ingeniería y Ciencias Aplicadas, Universidad Autónoma del Estado de Morelos, Av. Universidad 1001, Col. Chamilpa, 62209, Cuernavaca, MOR, México*

ABSTRACT

This paper describes a permutational-based Differential Evolution algorithm implemented in a wrapper scheme to find a feature subset to be applied in the construction of a near-optimal classifier. In this approach, the relevance of a feature chosen to build a better classifier is represented through its relative position in an integer-valued vector, and by using a permutational-based mutation operator, it is possible to create new feasible candidate solutions only. Furthermore, to provide a controlled diversity rate in the population, a straightforward repair-based recombination operator is utilized to evolve a population of candidate solutions. Unlike the other approaches in the existing literature using integer-valued vectors and requiring a predefined subset size, in this approach, this size is determined by an additional element included in the encoding scheme, allowing to find an adequate feature subset size to each specific dataset. Experimental results show that this approach is an effective way to create more accurate classifiers as they are compared with those obtained by other similar approaches.

1. Introduction

Differential Evolution (DE) is an evolutionary algorithm highlighting for its simplicity and effectiveness to solve numerical optimization problems [1]. Although DE is designed to deal with real-valued parameters, it has also been applied to work out with several combinatorial optimization problems [2–5]. In this case, two encoding strategies have been proposed in the existing literature: The first evolves a set of real-valued vectors using the standard DE variation operators and implements a mapping scheme to obtain the required discrete values representing a feasible solution [2, 3]. The other encodes the candidate solutions with arrays of countable objects (binary or integer values) and implements specialized variation operators generating feasible solutions by the application of repair mechanisms [4, 5].

Among the diverse fields of knowledge where DE has been applied stands out its use to solve machine learning tasks such as building classification and clustering models [6, 7], and selecting and extracting the most relevant features of a dataset [8, 9]. In particular, feature subset selection (FSS) is a high-interest area for the research community since it is known that the data quality impacts on the induced model performance. The presence of irrelevant and redundant features in a dataset leads to a slower learning process, and their use propitiates the creation of relationships degrading the model performance. Then, removing these features can reduce the dataset size, simplifying the predictive model, and enhancing its performance [10]. Since the FSS problem has been cataloged as an NP-hard problem [11], several evolutionary and swarm intelligence algorithms have been used to find a near-optimal feature subset to build efficient classifiers [12–17].

In this paper, a permutational-based Differential Evolution algorithm, named DE-FS^{PM}, for FSS in a wrapper scheme is described. DE-FS^{PM} evolves a population of integer-valued vectors using the error-rate of the induced classifier as fitness value. The integer-valued vector encodes the feature indexes of a dataset, where the feature subset utilized to build a classifier is selected based on their relative position in the vector, and the

**Corresponding author:
e-mail: rrivera@itver.edu.mx (Rafael Rivera-López)

subset size is determined by an additional element included in the vector. The evolutionary process applies a permutational-based mutation operator, and a repair-based recombination operator is implemented to ensure the construction of feasible solutions only. Although the permutational-based DE algorithm has been used to solve some combinatorial optimization problems [18, 19], to the best of our knowledge, it has not been applied to select an adequate feature subset to build a more precise classification model. Experimental results show that this approach is useful in selecting appropriate feature subsets as these are compared to those obtained by other similar algorithms.

The rest of this paper is organized as follows: In Section 2, a description of the DE algorithm and its use to solve combinatorial optimization problems is presented. A definition of the FSS problem is provided in Section 3, as well as a short revision of DE-based approaches for FSS described in the existing literature. Section 4 is devoted to detailing the elements of the scheme proposed in this work, and the experimental results are discussed in Section 5. Finally, Section 6 holds the conclusions and the future work of this proposal.

2. Differential Evolution algorithm

DE [1] is an evolutionary algorithm that, by exploiting the differences between vectors, performs an intelligent search to a near-optimal solution in complex solution spaces, guaranteeing a good trade-off between its exploration and exploitation skills. DE is commonly implemented as follows: First, an initial population $\mathbb{X}_0 = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^{\mathrm{NP}}\}$ of NP candidate solutions selected at random from the solution space is created. Then, the evolutionary process is conducted to build new populations as follows: For each $i \in \{1, \ldots, \mathrm{NP}\}$ in the g-th generation, a *target vector* $\mathbf{x}^i$ is taken from the $\mathbb{X}_{g-1}$ population, and it is utilized to build a *trial vector* $\mathbf{u}^i$ by applying the mutation and recombination operators. These vectors are evaluated by the selection operator to update a new population $\mathbb{X}_g$. Finally, when the stop condition is fulfilled, DE returns the best candidate solution in the current population.

Instead of implementing traditional recombination and mutation operators, the standard DE algorithm, named *DE/rand/1/bin*, applies a linear combination of three candidate solutions ($\mathbf{x}^{r_1}$, $\mathbf{x}^{r_2}$ and $\mathbf{x}^{r_3}$) to yield a *mutant vector* $\mathbf{v}^i$, as follows:

$$\mathbf{v}^i \leftarrow \mathbf{x}^{r_1} + \mathrm{F}\,(\mathbf{x}^{r_2} - \mathbf{x}^{r_3})\,, \tag{1}$$

where F is a user-specified value representing a scale factor applied to control the differential variation. $\mathbf{x}^{r_1}$, $\mathbf{x}^{r_2}$, and $\mathbf{x}^{r_3}$ are selected at random from the previous population, being different from each other and also different from the target vector.

The mutant vector is utilized to perturb the values of the target vector in the binomial recombination operator, as follows: For each $j \in \{1, \ldots, n\}$, either x^i_j or v^i_j is selected based on a comparison between a uniformly distributed random number $r \in [0, 1]$ and the crossover rate CR. This operator also uses a randomly chosen index $l \in \{1, \ldots, n\}$ to ensure that the trial vector $\mathbf{u}^i$ gets at least one value from $\mathbf{v}^i$, as follows:

$$u^i_j \leftarrow \begin{cases} v^i_j & \text{if } r \le \mathrm{CR} \vee j = l, \\ x^i_j & \text{otherwise.} \end{cases} \tag{2}$$

Finally, $\mathbf{u}^i$ is selected as a member of the new population if it has a better fitness value than that of $\mathbf{x}^i$.

DE has several advantages in comparison with other evolutionary algorithms, such as the simplicity of its implementation, its ability to produce better results than those obtained by the others, and its low space complexity [20]. The mutation operator favors a good trade-off between its exploitation and exploration skills, i.e., it is more explorative at the beginning, and more exploitative as the evolutionary process progresses [21]. Furthermore, the recombination and selection operators provide diversity control and continuous improvements in the DE algorithm, respectively [22]. On the other hand, although DE requires the definition of a smaller number of parameters compared to other algorithms, its performance is sensitive to the values selected for CR, F, and NP [23].

DE has been successful in solving numeric optimization problems, but it has also been used to solve different combinatorial optimization problems such as the Vehicle Routing Problem [2], the Traveling Salesman Problem [24], and several scheduling-based problems [3, 4], among others. Two encoding schemes have been utilized to represent the candidate solutions of these problems: real-valued vectors and arrays of countables (binary or integer values). In the first case, mapping schemes such as Random-Key decoders [25], and the Forward/Backward Transformation approach [26] are applied to transform one real-valued vector into an array of integers. On the other hand, when the candidate solutions are encoded with countable values representing permutations, specialized variation operators such as the permutation matrix [27], the adjacency matrix [27], the list of movements scheme [28], the geometric search operators [29], and the algebraic-based scheme [30] are needed to generate feasible solutions.

3. Feature subset selection

FSS tries to identify relevant and informative features in a training set and discards those that are irrelevant and redundant [31]. If A is the original feature set, $b \in \mathbb{N} : b \le |A|$ is a predefined subset size, $\wp(A)$ is the *power set* of A (the collection of all subsets of A), and $f : \wp(A) \to \mathbb{R}$ is the function determining the quality value of each feature subset of A, in this work, two types of FSS problems are defined as follows:

- **Weak FSS:** The feature subset $Y^* \in [A]^{=b}$ is an optimal solution of the weak FSS problem, if:

$$f(Y^*) \leftarrow \mathrm{opt}\left\{f(Y) : Y \in [A]^{=b}\right\},$$

  where $[A]^{=b}$ is the collection of subsets of A with size b.

- **Strong FSS:** The feature subset $Y \subseteq A$ is an optimal solution of the strong FSS problem, if:

$$f(Y^*) \leftarrow \mathrm{opt}\left\{f(Y) : Y \in \wp(A)\right\}.$$

There are three forms to combine one FSS method with a classification model induction procedure: filter, wrapper, and embedded schemes. In particular, in wrapping schemes, the

FSS is carried out by interacting with a classifier, by finding a feature subset achieving better performance for a particular learning model [32].

DE has been applied to solve the FSS problem by using integer-valued, binary-valued, and real-valued vectors to encode the candidate solutions. Integer-valued vectors are commonly used to encode the relative position of each feature in the original dataset, expressing a candidate subset as one sequence of feature indexes. To the best of our knowledge, DE-based approaches found in existing literature using integer-valued vectors only solve the weak FSS problem, as they look for a near-optimal subset with a user-specified size [33–37]. Although the initial population consists of valid candidate solutions, trial vectors generated by the variation operators can have repeated elements, then they must be repaired to avoid infeasible solutions. Two procedures to replace repeated elements have been defined in the existing literature. One of them uses a roulette-wheel-based structure representing a feature contribution rate to form useful subsets [33–35], and the other applies one ranked feature list previously built with some filter scheme [36]. Furthermore, integer representation has also been used in only one scheme preventing the creation of trial vectors with duplicate elements [37]. In this scheme, the initial vector value represents the first feature included in the subset, and the following values indicate the number of features that are omitted before adding the next one in the subset.

On the other hand, in both binary-valued [38, 39] and real-valued [8, 40] vectors, each value indicates whether a feature of the original dataset is or not selected as being one subset element.

4. Permutational-based DE to feature subset selection

In this paper, a permutational-based DE algorithm to solve the strong FSS problem in a wrapper scheme is described. This approach, named DE-FS$^{\text{PM}}$, differs from the others in the existing literature in four aspects:

1. The integer-valued vector encodes all features of a dataset, instead of encoding only those used in a candidate subset.
2. The size of the feature subset is determined in the evolutionary process by using an additional element included in the encoded vector.
3. The relevance of a feature used to build a better classifier is represented through its relative position in the vector, and a permutation-matrix-based mutation operator can be used to create new candidate solutions.
4. A simple repair-based recombination operator is implemented to ensure the construction of feasible candidate solutions only.

The DE-FS$^{\text{PM}}$ elements are described in the following paragraphs.

Encoding scheme: The integer-valued vector utilized to encode the dataset features is constructed as follows:

- If the dataset has m features, the size of the integer-valued vector is $n \leftarrow m + 1$.
- Vector values encode the features position in the dataset: The first feature in the dataset is represented with the value 1, the second feature is represented with the value 2, and so for the remaining attributes.
- Value 0 is utilized as one indicator to select the features used in the classifier construction process. In this case, indices placed to the left of value 0 in the vector are considered to be part of the candidate subset, and indices located to its right side are discarded.

By example, if $\mathbf{x}^i = \left(8, 3, 1, 6, 2, 0, 5, 4, 7\right)$ is a vector representing one candidate solution, the feature subset selected is $Y^i = \left\{1, 2, 3, 6, 8\right\}$, and features 4, 5, and 7 are not used to build a classifier. Since these vectors are perturbed using a permutational-based mutation operator, value 0 is relocated during the evolutionary process, allowing it to create feature subsets with different sizes.

Mutation operator: This operator implements the permutation matrix approach [27] in which a matrix represents the difference between two vectors encoding sequences of non-repeated integer values. If $\mathbf{P}$ is the permutation matrix mapping $\mathbf{x}^{r_2}$ from $\mathbf{x}^{r_3}$, as follows:

$$\mathbf{x}^{r_2} \leftarrow \mathbf{P}\mathbf{x}^{r_3} \tag{3}$$

and $\mathbf{P}_{\text{F}}$ is a *scaled matrix* with a fraction of permutations randomly selected from $\mathbf{P}$, this matrix is used to create a mutant vector from $\mathbf{x}^{r_1}$, as follows:

$$\mathbf{v}^i \leftarrow \mathbf{P}_{\text{F}}\mathbf{x}^{r_1}. \tag{4}$$

The F parameter is used as a threshold value to select a subset of permutations from $\mathbf{P}$. Since $\mathbf{P}_{\text{F}}$ represents the permutations applied to a feasible solution, it produces a new feasible solution, only. Figure 1 shows an example of this procedure, where $\mathbf{P}_{\text{F}}$ is computed using the algorithm defined by Price *et al.* [27], as follows: For each $i \in \{1, \ldots, n\}$, if $p[i,i] = 0 \wedge r_i > \text{F}$, first find the j-th row, $j \in \{1, \ldots, n\}$, where $p[j,i] \neq 0$, and then swap rows i and j in $\mathbf{P}$. r_i is a uniformly distributed random number between 1 and 0. In this example, F is 0.5, and the sequence of random numbers to build $\mathbf{P}_{\text{F}}$ is $\left(0.3, 0.8, 0.4, 0.6, 0.4, 0.3, 0.4, 0.3, 0.7\right)$.

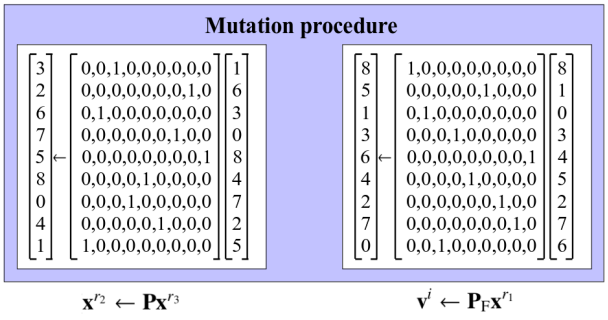


Fig. 1. Example of the mutation procedure in the DE-FS$^{\text{PM}}$ algorithm.

Although it is known that using a permutation matrix to evolve integer-valued vectors has several disadvantages, such as its tendency of stagnation, its usage with recombination operators can lead to an acceptable exploration of the solution space.

Recombination operator: This operator uses Eq. 2 to combine the values of the mutant vector $\mathbf{v}^i$ with those of the target vector $\mathbf{x}^i$. The trial vector $\mathbf{u}^i$ obtained by this operation can contain repeated values, and a repair procedure is needed. In this case, a simple strategy is applied: First, an auxiliary variable p is assigned the vector size ($p \leftarrow n$). Next, for each $j \in \{1, \ldots, p\}$, if u^i_j has a value equal to any value of the previous $\{u^i_1, \ldots, u^i_{j-1}\}$ elements, this value is removed, and the $\{u^i_j, \ldots, u^i_p\}$ elements are adjusted as follows:

1. For each $k \in \{j, \ldots, p-1\}$, $u^i_k \leftarrow u^i_{k+1}$,
2. $p \leftarrow p - 1$.

Finally, for each $j \in \{p+1, \ldots, n\}$, u^i_j is assigned one non-used value, taken from $\mathbf{x}^i$ and $\mathbf{v}^i$.

This procedure is based on the idea that the non-selected features by the recombination operator can be placed at the end of the vector, considering that they are less important than those selected. Since the feature subset is obtained using the feature indexes placed to the left of value 0 in the trial vector, they are less likely to be included in the selected feature subset. Figure 2 shows an example of this repair procedure. In this example, the trial vector result of the recombination operator has two repeated values: 1 and 3, and values 4 and 7 are not present in it. Then, the repeated values are removed of $\mathbf{u}^i$, and the non-used values are placed at the end of the trial vector.

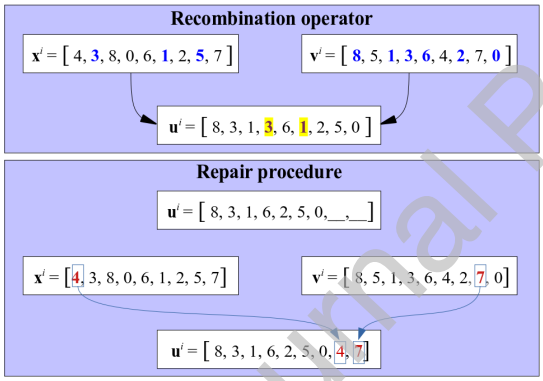


Fig. 2. Example of the repair procedure in the DE-FS^{PM} algorithm.

Algorithm 1 shows the general structure of the DE-FS^{PM} method. With those elements, this method can find feature subsets that are used to build a better classifier than those induced using all features in the dataset.

Algorithm 1 The DE-FS^{PM} algorithm.

function DE-FS^{PM}(CR, F, NP, NG)
Input: The crossover rate (CR), the scale factor (F), the population size (NP), and the number of generetions (NG).
Output: The best individual in current population ($\mathbf{x}^{best}$).

$\mathbb{X}_0 \leftarrow \varnothing$
for each $i \in \{1, \ldots, NP\}$ **do**
  $\mathbf{x}^i \leftarrow$ A permutation chosen at random from the solution space.
  $\mathbb{X}_0 \leftarrow \mathbb{X}_0 \cup \{\mathbf{x}^i\}$
end for
for each $g \in \{1, \ldots, NG\}$ **do**
  $\mathbb{X}_g \leftarrow \varnothing$
  for each $i \in \{1, \ldots, NP\}$ **do**
    $\mathbf{x}^i \leftarrow$ Target vector from $\mathbb{X}_{g-1}$
    $\mathbf{v}^i \leftarrow$ Mutated vector generated using Eqs. (3) and (4)
    $\mathbf{u}^i \leftarrow$ Trial vector constructed using Eq. (2) and the repair procedure.
    $\mathbb{X}_g \leftarrow \mathbb{X}_g \cup \begin{cases} \{\mathbf{u}^i\} & \text{if } f(\mathbf{u}^i) \text{ is better than } f(\mathbf{x}^i) \\ \{\mathbf{x}^i\} & \text{otherwise} \end{cases}$
  end for
end for
$\mathbf{x}^{best} \leftarrow$ The best individual in $\mathbb{X}_g$
return $\mathbf{x}^{best}$

5. Experimental study

Today, the FSS problem has gained relevance since the rapid growth of emerging areas such as big data, data science, and data analytics. Several new studies have been conducted to analyze, categorize, and compare FSS techniques. In those studies, diverse ways of conducting the experimental part have been applied. Some authors first use a proportion of the dataset to find one adequate feature subset and then evaluate its quality value with the remaining instances [8, 9, 37]. Others carry out the selection and evaluation of the feature subset simultaneously [17, 33, 34, 36]. Furthermore, other researchers first find a near-optimal feature subset using some validation scheme such as cross-validation (CV) or holdout (HO) and then take the same instances to evaluate the performance of the classifier induced with the selected features [16].

In this section, the experimental study adopted to analyze and compare the DE-FS^{PM} performance is presented. First, the experimental methodology applied in this proposal is detailed. Then, the definition of the DE-FS^{PM} parameters is given. Next, the experimental results and the statistical tests carried out to evaluate these results are outlined. Finally, a discussion about the performance of the DE-FS^{PM} method is provided.

Experimental methodology: This methodology includes the choice of the datasets used in the experiments, the definition of the validation scheme, and the strategy to compare the experimental results. Table 1 shows the datasets used in the experimental study, the first 24 datasets are chosen from the UCI machine learning repository [41], and the remaining 6 are microarray datasets obtained from the Kent Ridge Biomedical Dataset Repository [42].

The results of the DE-FS^{PM} method are compared with those obtained by the following FSS approaches:

1. **SFS** (Sequential Forward Selection) [43]: This selection strategy sequentially adds the feature improving one quality-measure when it is combined with the previously selected features, starting from one empty feature set.
2. **SBE** (Sequential Backward Elimination) [44]: This selection approach sequentially removes the feature that least improves the quality-measure value, starting from the full

Table 1. Description of datasets used in the experiments.

Dataset	Instances	Features	Classes	Dataset	Instances	Features	Classes
UCI datasets:							
arrhythmia	452	279	16	audiology	226	69	24
australian	690	14	2	coil200	9000	85	2
cylinder-b	540	39	2	crx	690	15	2
dermatology	366	34	6	german-c	1000	20	2
h-valley	1212	100	2	ionosphere	351	34	2
madelon	2600	500	2	m-libras	360	90	15
musk-1	476	168	2	musk-2	6598	166	2
optic	5620	64	10	parkinsons	195	22	2
semeion	1593	256	10	sonar	208	60	2
soybean	683	35	19	spectf	267	44	2
splice	3190	60	3	vehicle	846	18	4
vote	435	16	2	wdbc	569	30	2
Microarray datasets:							
all-aml	72	7129	3	colon	62	2000	2
leukemia	72	7129	2	lymphoma	62	4026	3
ovarian	253	15154	2	srbct	83	2308	4

feature set.

3. **SBPSO** (Set-based Particle Swarm Optimization) [16]: This method uses a particle swarm optimization (PSO) algorithm in conjunction with a k-nearest neighbor (k-NN) classifier to find a near-optimal feature subset. SBPSO defines the position and velocity of a particle (candidate solution) as mathematical sets.
4. **DE-FLS** (Differential Evolution with Fuzzy Local Search) [8]: This is a DE-based method using a k-NN classifier to determine the quality of candidate solutions encoded by real-valued vectors. At the end of each generation of the evolutionary process, DE-FLS applies a fuzzy-local-search procedure to remove redundant features and improve the best candidate solution.
5. **M-SVM** (Memetic algorithm based-SVM) [17]: This is a hybrid approach combining an Emperor Penguin Optimizer (EPO) as global search method with a Social Engineering Optimizer (SEO) as local search procedure. M-SVM uses a Support Vector Machine (SVM) as classifier whose parameters (C and γ) are self-adjusted since they are included in the real-valued vector representing a candidate solution. Before starting the search stage, M-SVM applies a filtering scheme to select the 500 most representative features.

In this work, SBPSO, DE-FLS, and M-SVM are selected for two reasons: (1) they report competitive results with several datasets, and (2) their authors provide one accurate and well-defined description of their experimental setup. In particular, SBPSO is compared to three other PSO-based FSS approaches, and its authors report that their results outperform those of these algorithms with statistical significance. Furthermore, the experimental results of DE-FLS are superior to those of two deterministic FSS strategies and several swarm-based FSS approaches. Finally, the authors of M-SVM report significant improvements in the accuracy and number of selected features with microarray datasets. M-SVM results are compared with those of two GA and PSO-based procedures implemented by the authors.

The experimental methodologies used in SBPSO, DE-FLS, and M-SVM are reproduced in this study to favor a fair comparison between the DE-FS^{PM} results with those obtained by these methods. First, datasets are preprocessed by applying: min-max normalization (SBPSO, M-SVM), missing values imputation (SBPSO), missing values elimination (DE-FLS), and conversion from nominal to numerical values (SBPSO). Then, for each dataset, the following steps are applied:

- 1st experiment (SBPSO): First, an FSS procedure using a stratified five-fold CV scheme with the dataset instances is applied. Then, the selected subset is evaluated using a stratified ten-fold CV scheme with the dataset instances. By each dataset, the most appropiate k-value of the classifier is computed.
- 2nd experiment (DE-FLS): First, each dataset is randomly partitioned into 70% as the training set and the remaining as the test set. Then, an FSS procedure using a stratified ten-fold CV scheme with the training set is applied. Finally, the best selected subset is evaluated using the test set. DE-FLS use 5 as the k-value of the classifier.
- 3rd experiment (M-SVM): The selection and evaluation of the feature subset is carried out in a single step using a stratified ten-fold CV scheme.

Several independent runs are conducted (thirty for the first two experiments and ten for the remaining one), and the average accuracy of the classifier is reported.

DE-FS^{PM} parameters setting: The DE-FS^{PM} method is implemented in the Java language using the Weka [45] and the JMetal libraries [46]. This method uses the rand/1/bin variant of the DE algorithm and, since the DE performance is sensitive to its parameter values, in this work, the values of F and CR are determined using the Irace package [47]. This package implements several automatic configuration strategies based on an iterated racing procedure chosing the more appropriate values to the algorithm parameters using a set of training problem instances. By using five representative datasets of the total used in the experimental study, the Irace package indicates that one mutation scale factor of 0.1514, and one crossover rate of 0.8552 are the best parameters' values.

Furthermore, following the suggestion of Storn and Price [1], the population size is adjusted to $5n$, with 200 and 450 individuals as lower and upper bound, respectively. These bounds are used to ensure that the population is not too small to not allow a reasonable exploration of the search space, and it is not too large to impact the algorithm runtime. The fitness function used in the DE-FS^{PM} method computes the error-rate of each classifier induced using the feature subset encoded in each candidate solution. Finally, the evolutionary process stops when it reaches 200 generations or when the best fitness value in the population had not improved in 100 iterations.

Results: In the next paragraphs, the results of the experimental study are detailed: We analyze the accuracies and the feature subset sizes obtained with DE-FS^{PM} and the compared algorithms. This study is carried out on a computer with four Intel Core i7-6500U 2.50 GHz CPUs and 8 GB RAM. These results are shown in several tables where the best one for each dataset is highlighted using bold numbers. Furthermore, numbers in round brackets are used to refer to (1) the ranking (Rank)

reached of each method, and (2) the reduction rate (% Red) of the feature subset obtained by each method, concerning the total dataset features. These tables also show the win-tie-loss values and the average ranking for each method.

Tables 2 and 3 show the results of the first experiment. In Table 2 is observed that DE-FS^{PM} produces better results than those generated by SBPSO since it yields higher average accuracies in 15 datasets. In Table 3 can be noted that SFS gets the smallest subsets; however, its average accuracies are not the best. Furthermore, subsets obtained by DE-FS^{PM} in 18 datasets are lower than those generated by SBE and SBPSO.

Table 2. Average accuracies obtained in the first experiment.

Dataset	Without FS		SFS		SBE		SBPSO		DE-FS^{PM}	
	Acc	Rank	Acc	Rank	Acc	Rank	Acc	Rank	Acc	Rank
arrhythmia	58.92	(5)	73.73	(2)	63.72	(4)	70.06	(3)	**76.19**	(1)
audiology	76.29	(5)	83.83	(3)	84.28	(2)	83.68	(4)	**85.43**	(1)
australian	85.55	(5)	86.64	(3)	86.34	(4)	**87.47**	(1)	87.02	(2)
cylinder-b	73.68	(5)	83.31	(2)	79.57	(4)	82.03	(3)	**85.09**	(1)
crx	86.41	(3)	86.03	(4)	87.38	(2)	71.24	(5)	**87.83**	(1)
dermatology	96.98	(5)	**98.37**	(1.5)	97.94	(4)	98.18	(3)	**98.37**	(1.5)
german-c	75.11	(5)	76.73	(3)	75.81	(4)	77.00	(2)	**77.22**	(1)
h-valley	65.05	(4)	70.06	(2)	68.35	(3)	63.51	(5)	**72.04**	(1)
ionosphere	86.82	(5)	92.67	(3)	90.89	(4)	94.43	(2)	**94.86**	(1)
m-libras	86.13	(5)	88.85	(3)	87.53	(4)	89.51	(2)	**89.57**	(1)
musk-1	85.63	(5)	90.51	(4)	92.19	(3)	**95.16**	(1)	94.64	(2)
parkinsons	95.92	(4)	95.08	(5)	98.11	(3)	**99.19**	(1)	98.61	(2)
sonar	86.53	(5)	89.47	(4)	91.08	(3)	92.62	(2)	**93.77**	(1)
soybean	91.63	(5)	95.75	(2)	94.57	(3)	94.50	(4)	**95.85**	(1)
spectf	77.69	(5)	80.78	(3)	80.17	(4)	84.00	(2)	**84.88**	(1)
vehicle	70.44	(5)	71.26	(4)	72.96	(3)	73.77	(2)	**74.44**	(1)
vote	93.58	(5)	95.63	(4)	95.98	(3)	96.17	(2)	**96.45**	(1)
wdbc	97.01	(5)	97.26	(4)	97.47	(3)	97.63	(2)	**97.64**	(1)
WTL:	0-0-18		0-1-17		0-0-18		3-0-15		14-1-3	
Avg. ranks:	4.78		3.14		3.33		2.56		1.19	

Table 3. Average feature subset sizes obtained in the first experiment.

Dataset	Features	SFS		SBE		SBPSO		DE-FS^{PM}	
		|FS|	% Red	|FS|	% Red	|FS|	% Red	|FS|	% Red
arrhythmia	279	14.0	(95.0)	126.0	(54.8)	65.6	(76.5)	**10.5**	(96.2)
audiology	69	**13.7**	(80.1)	27.8	(59.8)	45.4	(34.2)	23.7	(65.7)
australian	14	**3.1**	(77.6)	10.2	(26.9)	6.3	(55.0)	8.5	(39.3)
cylinder-b	39	**5.4**	(86.2)	26.2	(32.9)	17.4	(55.4)	5.8	(85.1)
crx	15	**2.0**	(86.5)	10.8	(28.2)	7.7	(48.7)	7.0	(53.5)
dermatology	34	**12.5**	(63.3)	21.3	(37.4)	24.6	(27.6)	21.2	(37.6)
german-c	20	**6.5**	(67.5)	17.4	(13.2)	9.0	(55.0)	9.9	(50.3)
h-valley	100	**9.1**	(90.9)	79.2	(20.8)	40.4	(59.6)	10.9	(89.1)
ionosphere	34	**4.3**	(87.4)	25.7	(24.3)	7.6	(77.6)	7.2	(78.8)
m-libras	90	**11.4**	(87.4)	69.5	(22.7)	38.6	(57.1)	19.6	(78.2)
musk-1	168	**14.1**	(91.6)	102.2	(39.2)	69.9	(58.4)	35.8	(78.7)
parkinsons	22	**6.6**	(70.0)	12.6	(42.7)	12.9	(41.4)	10.3	(53.3)
sonar	60	**10.0**	(83.3)	44.6	(25.7)	25.9	(56.8)	20.4	(66.0)
soybean	35	**13.8**	(60.7)	20.2	(42.3)	22.1	(36.9)	14.7	(57.9)
spectf	44	**2.0**	(95.5)	38.5	(12.5)	12.8	(70.9)	5.3	(87.9)
vehicle	18	**6.5**	(64.1)	14.4	(20.0)	11.8	(34.4)	9.8	(45.5)
vote	16	**1.0**	(93.8)	9.1	(42.9)	5.8	(63.7)	5.5	(65.8)
wdbc	30	**6.3**	(79.0)	22.9	(23.7)	20.7	(31.0)	16.8	(44.1)

Tables 4 and 5 show the results of the other experiments. Table 4 indicates that DE-FS^{PM} creates better 5-NN classifiers than DE-FLS in 8 datasets, and in Table 5 is observed that it builds better SVM classifiers than M-SVM in 5 datasets. However, it can be noted that its feature subsets are not smaller than those generated by the compared methods.

Table 4. Results obtained in the second experiment.

Dataset	Without FS		DE-FLS		DE-FS^{PM}		DE-FLS		DE-FS^{PM}	
	Acc	Rank	Acc	Rank	Acc	Rank	|FS|	% Red	|FS|	% Red
coil2000	93.7	(3)	**94.0**	(1.5)	**94.0**	(1.5)	**2.0**	(97.6)	5.9	(93.1)
madelon	71.8	(3)	**85.6**	(1)	78.2	(2)	**37.0**	(92.6)	213.2	(57.4)
musk-1	80.0	(3)	83.7	(2)	**93.8**	(1)	64.5	(61.1)	**42.7**	(74.3)
musk-2	80.0	(3)	81.8	(2)	**96.2**	(1)	**35.4**	(78.7)	108.2	(34.8)
optic	98.9	(2)	97.3	(3)	**99.1**	(1)	**21.3**	(66.7)	41.5	(35.2)
semeion	90.8	(2)	88.1	(3)	**90.9**	(1)	**109.4**	(57.3)	222.4	(13.1)
soybean	85.5	(3)	88.0	(2)	**88.4**	(1)	**13.7**	(60.9)	28.3	(19.1)
sonar	75.8	(3)	76.8	(2)	**80.8**	(1)	**21.0**	(65.0)	21.1	(64.8)
splice	66.8	(3)	78.9	(2)	**82.3**	(1)	**8.0**	(86.7)	25.0	(58.3)
WTL:	0-0-9		1-1-7		7-1-1					
Avg. ranks:	2.78		2.06		1.17					

Table 5. Results obtained in the third experiment.

Dataset	Without FS		M-SVM		DE-FS^{PM}		M-SVM		DE-FS^{PM}	
	Acc	Rank	Acc	Rank	Acc	Rank	|FS|	% Red	|FS|	% Red
all-aml	62.8	(3)	**97.7**	(1)	95.7	(2)	**8**	(99.9)	212.3	(97.0)
colon	64.5	(3)	96.4	(2)	**97.2**	(1)	**8**	(99.6)	27.1	(98.6)
leukemia	85.3	(3)	98.8	(2)	**98.9**	(1)	**7**	(99.9)	236.3	(96.7)
lymphoma	**100.0**	(1.5)	99.0	(3)	**100.0**	(1.5)	**6**	(99.9)	562.0	(86.0)
ovarian	86.9	(3)	**100.0**	(1.5)	**100.0**	(1.5)	**5**	(99.9)	90.0	(99.4)
srbct	97.6	(3)	98.9	(2)	**99.9**	(1)	**6**	(99.7)	821.4	(64.4)
WTL:	0-1-5		1-1-4		3-2-1					
Avg. ranks:	2.75		1.92		1.33					

Statistical tests: In this work, the Friedman [48] and the Bergmann-Hommel [49] tests are applied to evaluate the DE-FS^{PM} performance. These non-parametric tests are used since it is known that experimental studies involving evolutionary algorithms do not fulfill the necessary conditions (independence, normality, and homoscedasticity) to apply a parametric test [50]. Furthermore, they are most appropriate for multiple comparisons among several methods: The Friedman test detects significant differences over the whole multiple comparisons, and the Bergmann-Hommel post-hoc test finds the particular pairwise comparisons with statistical differences [51]. For each experiment, the Friedman test is run and, if the p-value obtained is less than a significance level of 5%, the null hypothesis is rejected, and the post-hoc test is applied to evaluate all possible pairwise comparisons. Table 6 shows that all experiments have statistical differences and in Table 7 are shown those pairwise comparisons where DE-FS^{PM} is evaluated. It is observed that DE-FE^{PM} has a better performance than (1) the classifiers induced without an FSS procedure, (2) two deterministic FSS schemes, and (3) SBPSO. Furthermore, it has one comparable performance than DE-FLS and M-SVM.

Discussion: Tables 2, 4, and 5 show that FSS allows building more precise classifiers than those induced using the full feature set. Furthermore, in most cases, DE-FS^{PM} builds better classifiers than those created with the compared methods in the three experiments. Nevertheless, although DE-FS^{PM} generates smaller subsets than those of SBPSO, it creates subsets larger than those produced by DE-LFS and M-SVM. This behavior is due to three differences in the implementation of these methods. First, they use a filtering scheme to reduce the number of features involved in the FSS procedure: DE-LFS apply it at the

end of each iteration of its evolutionary process, and M-SVM uses it before to start its search procedure. Next, DE-LFS uses a fitness function combining the error-rate and the feature subset size. Finally, these methods apply a local search to improve the performance of the selected subset. Since the evolutionary process in DE-FS^{PM} is guided using the error-rate only, it generates more precise classifiers than the others, without considering the size of the feature subset. This behavior is clear when DE-FS^{PM} is compared with SVPSO. In particular, the proposed method selects those features that are relevant to the problem, which are discarded in the other approaches with the intention of reducing the subset size.

The results of the statistical analysis point out that DE-FS^{PM} has a better performance than SBPSO and a comparable behavior with both DE-LFS and M-SVM. However, DE-FS^{PM} is a more simple FSS strategy because: (1) it does not use additional local search procedures, (2) it needs to set the smallest number of parameters, and (3) it is a pure wrapper approach. DE-FS^{PM} creates more accurate classifiers than those of the other methods due to the exploration and exploitation skills of the DE algorithm. Also, since the permutation matrix represents the difference between two permutations, the mutation operator maintains the DE essence by generating a new solution using the difference of two solutions selected at random from the current population.

Finally, since DE is an evolutionary algorithm, its computational complexity is related to the number of fitness function evaluations. Then, if NG is the number of generations, NP is the population size, and n is the size of a candidate solution, the asymptotic computational complexity of DE-FS^{PM} can be computed as $O(NP \times NG \times n^2)$. In practice, since each individual needs to generate a classifier using the training set, the algorithm runtime depends on the number of training instances. Still, this consideration is assumed for any wrapper-based algorithm, and it is a drawback of this type of FSS-schemes.

6. Conclusions

In this paper, a permutational-based DE algorithm to solve the strong FSS problem is described. The feature subset size is controlled by the evolutionary process using integer-valued vectors encoding the candidate solutions, and DE uses both a permutation-matrix mutation operator and a straightforward recombination operator. To the best of our knowledge, the FSS problem had not been represented as a permutation vector in the existing literature, and experimental results indicate that it is a promising approach to generate more accurate feature selection. Although the permutation-matrix approach with the DE algorithm has reported some disadvantages, the experimental results in this paper suggest that such approach has a highly competitive performance in the FSS problem when compared against classic and very recent nature-inspired approaches.

Future directions of this work are aimed at applying other strategies attempting to improve the capabilities of the mutation operator and to apply different permutation-based representations, such as those used by Random-Key decoders. In addition, it is important to compare the experimental results with those of different deterministic and evolutionary and swarm-based FSS approaches. Finally, the use of a multi-objective approach with the DE-FS^{PM} method can improve the generation of more compact feature subsets used to induce more precise classifiers.

Table 6. Results of Friedman test.

Experiment	Datasets	Methods	Friedman	p-value	Null hypothesis
First	18	5	48.589	7.113×10^{-10}	rejected
Second	9	3	11.722	2.848×10^{-3}	rejected
Third	6	3	6.083	4.776×10^{-2}	rejected

Table 7. Results of Bergmann-Hommel post-hoc test.

Experiment	DE-FS^{PM} versus	p-value	Result
First	Without FS	1.0542×10^{-10}	different
First	SBE	4.9440×10^{-5}	different
First	SFS	2.2485×10^{-4}	different
First	SBPSO	9.8097×10^{-3}	different
Second	Without FS	1.8947×10^{-3}	different
Second	DE-FLS	5.9346×10^{-2}	comparable
Third	Without FS	4.2413×10^{-2}	different
Third	M-SVM	3.1232×10^{-1}	comparable

References

[1] R. Storn, K. Price, Differential Evolution–A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, Journal of Global Optimization 11 (4) (1997) 341–359. doi:10.1023/A:1008202821328.

[2] H. Xu, J. Wen, Differential evolution algorithm for the optimization of the vehicle routing problem in logistics, in: Int. Conf. on Computational Intelligence and Security, IEEE, 2012, pp. 48–51. doi:10.1109/CIS.2012.19.

[3] J. Liang, P. Wang, L. Guo, B. Qu, C. Yue, K. Yu, Y. Wang, Multi-objective flow shop scheduling with limited buffers using hybrid self-adaptive differential evolution, Memetic Computing 11 (4) (2019) 407–422. doi:10.1007/s12293-019-00290-5.

[4] P. Yan, G. Wang, A. Che, Y. Li, Hybrid discrete differential evolution algorithm for biobjective cyclic hoist scheduling with reentrance, Computers & Operations Research 76 (2016) 155–166. doi:10.1016/j.cor.2016.06.011.

[5] H. Li, L. Zhang, Y. Jiao, Discrete differential evolution algorithm for integer linear bilevel programming problems, Journal of Systems Eng. and Electronics 27 (4) (2016) 912–919. doi:10.21629/JSEE.2016.04.20.

[6] C. Duchanoy, M. Moreno, L. Urbina, C. Cruz-Villar, H. Calvo, J. Rubio, A novel recurrent neural network soft sensor via a differential evolution training algorithm for the tire contact patch, Neurocomputing 235 (2017) 71–82. doi:10.1016/j.neucom.2016.12.060.

[7] I. Mishra, I. Mishra, J. Prakash, Differential Evolution with Local Search Algorithms for Data Clustering: A Comparative Study, in: Soft Computing: Theories and Applications, Springer, 2019, pp. 557–567. doi:10.1007/978-981-13-0589-4_52.

[8] E. Hancer, Differential evolution for feature selection: a fuzzy wrapper–filter approach, Soft Computing 23 (13) (2019) 5233–5248. doi:10.1007/s00500-018-3545-7.

[9] A. Bidgoli, S. Rahnamayan, H. Ebrahimpour, Opposition-based multi-objective binary differential evolution for multi-label feature selection, in: Int. Conf. on Evolutionary Multi-Criterion Optimization, Springer, 2019, pp. 553–564. doi:10.1007/978-3-030-12598-1_44.

[10] M. Dash, H. Liu, Feature selection for classification, Intelligent Data Analysis 1 (1) (1997) 131–156. doi:10.1016/S1088-467X(97)00008-5.

[11] S. Davies, S. Russell, NP-completeness of searches for smallest possible feature sets, Tech. Rep. FS-94-02, AAAI (1994).

[12] F. Viegas, L. Rocha, M. Gonçalves, F. Mourão, G. Sá, T. Salles, G. Andrade, I. Sandin, A genetic programming approach for feature selection in

highly dimensional skewed data, Neurocomputing 273 (2018) 554–569. doi:10.1016/j.neucom.2017.08.050.

[13] Y. Zhang, X. Song, D. Gong, A return-cost-based binary firefly algorithm for feature selection, Information Sciences 418 (2017) 561–574. doi:10.1016/j.ins.2017.08.047.

[14] R. Guha, M. Ghosh, S. Kapri, S. Shaw, S. Mutsuddi, V. Bhateja, R. Sarkar, Deluge based Genetic Algorithm for feature selection, Evolutionary Intelligence (2019) 1–11doi:10.1007/s12065-019-00218-5.

[15] Y. Zhang, S. Cheng, Y. Shi, D. Gong, X. Zhao, Cost-sensitive feature selection using two-archive multi-objective artificial bee colony algorithm, Expert Systems with Applications 137 (2019) 46–58. doi:10.1016/j.eswa.2019.06.044.

[16] A. P. Engelbrecht, J. Grobler, J. Langeveld, Set based particle swarm optimization for the feature selection problem, Engineering Applications of Artificial Intelligence 85 (2019) 324–336. doi:10.1016/j.engappai.2019.06.008.

[17] S. K. Baliarsingh, W. Ding, S. Vipsita, S. Bakshi, A memetic algorithm using emperor penguin and social engineering optimization for medical data classification, Applied Soft Computing 85 (2019) 105773. doi:10.1016/j.asoc.2019.105773.

[18] F. Guimarães, R. C. P. Pedrosa Silva, R. S. Prado, O. M. Neto, D. D. Davendra, Flow Shop Scheduling Using a General Approach for Differential Evolution, in: I. Zelinka, et al. (Eds.), Handbook of Optimization, Springer, Berlin, Heidelberg, 2013, pp. 597–614. doi:10.1007/978-3-642-30504-7_24.

[19] M. Baioletti, A. Milani, V. Santucci, Linear Ordering Optimization with a Combinatorial Differential Evolution, in: Int. Conf. on Systems, Man, and Cybernetics, IEEE, 2015, pp. 2135–2140. doi:10.1109/SMC.2015.373.

[20] S. Das, P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE Trans. on Evolutionary Computation 15 (1) (2011) 4–31. doi:10.1109/TEVC.2010.2059031.

[21] F. Neri, V. Tirronen, Recent advances in differential evolution: a survey and experimental analysis, Artificial Intelligence Review 33 (1-2) (2010) 61–106. doi:10.1007/s10462-009-9137-2.

[22] V. Feoktistov, Differential Evolution: In Search of Solutions, Springer, 2007. doi:10.1007/978-0-387-36896-2_1.

[23] K. Zielinski, P. Weitkemper, R. Laur, K. D. Kammeyer, Parameter Study for Differential Evolution Using a Power Allocation Problem Including Interference Cancellation, in: Int. Conf. on Evolutionary Computation, 2006, pp. 1857–1864. doi:10.1109/CEC.2006.1688533.

[24] J. G. Sauer, L. dos Santos Coelho, Discrete differential evolution with local search to solve the traveling salesman problem, in: Int. Conf. on Cybernetic Intelligent Systems, IEEE, 2008, pp. 1–6. doi:10.1109/UKRICIS.2008.4798955.

[25] P. Kromer, J. Janoušek, J. Platoš, Random Key Self–Organizing Migrating Algorithm for Permutation Problems, in: Congress on Evolutionary Computation, IEEE, 2019, pp. 2878–2885. doi:10.1109/CEC.2019.8790322.

[26] G. Onwubolu, D. Davendra, Scheduling flow shops using differential evolution algorithm, European Journal of Operational Research 171 (2) (2006) 674–692. doi:10.1016/j.ejor.2004.08.043.

[27] K. Price, R. M. Storn, J. A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer, 2006. doi:10.1007/3-540-31306-0.

[28] R. S. Prado, R. Silva, F. Guimarães, O. Neto, Using differential evolution for combinatorial optimization: A general approach, in: Int. Conf. on Systems, Man and Cybernetics, IEEE, 2010, pp. 11–18. doi:10.1109/ICSMC.2010.5642193.

[29] A. Moraglio, J. Togelius, S. Silva, Geometric differential evolution for combinatorial and programs spaces, Evolutionary Computation 21 (4) (2013) 591–624. doi:10.1162/EVCO_a_00099.

[30] V. Santucci, M. Baioletti, A. Milani, Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion, IEEE Trans. on Evolutionary Computation 20 (5) (2015) 682–694. doi:10.1109/TEVC.2015.2507785.

[31] G. Chandrashekar, F. Sahin, A survey on feature selection methods, Computers & Electrical Engineering 40 (1) (2014) 16–28. doi:10.1016/j.compeleceng.2013.11.024.

[32] R. Kohavi, G. John, Wrappers for feature subset selection, Artificial Intelligence 97 (1-2) (1997) 273–324. doi:10.1016/S0004-3702(97)00043-X.

[33] R. Khushaba, A. Al-Ani, A. Al-Jumaily, Feature Subset Selection Using Differential Evolution, in: M. Köppen, et al. (Eds.), Advances in Neuro-Information Processing, Springer, Berlin, Heidelberg, 2009, pp. 103–110. doi:10.1007/978-3-642-02490-0_13.

[34] R. Khushaba, A. Al-Ani, A. Al-Jumaily, Feature subset selection using differential evolution and a statistical repair mechanism, Expert Systems with Applications 38 (9) (2011) 11515–11526. doi:10.1016/j.eswa.2011.03.028.

[35] B. Niu, X. Yang, H. Wang, K. Huang, S. Weng, Feature Subset Selection Using a Self-adaptive Strategy Based Differential Evolution Method, in: Y. Tan, et al. (Eds.), Advances in Swarm Intelligence, Springer, Cham, 2018, pp. 223–232. doi:10.1007/978-3-319-93815-8_22.

[36] A. Ghosh, A. Datta, S. Ghosh, Self-adaptive differential evolution for feature selection in hyperspectral image data, Applied Soft Computing 13 (4) (2013) 1969–1977. doi:10.1016/j.asoc.2012.11.042.

[37] D. K. Tasoulis, V. P. Plagianakos, M. N. Vrahatis, Differential Evolution Algorithms for Finding Predictive Gene Subsets in Microarray Data, in: I. Maglogiannis, et al. (Eds.), Artificial Intelligence Applications and Innovations, Springer US, 2006, pp. 484–491. doi:10.1007/0-387-34224-9_56.

[38] X. S. Zhao, L. L. Bao, Q. Ning, J. C. Ji, X. W. Zhao, An Improved Binary Differential Evolution Algorithm for Feature Selection in Molecular Signatures, Molecular Informatics 37 (4) (2018) 1700081. doi:10.1002/minf.201700081.

[39] Y. Zhang, D. Gong, X. Gao, T. Tian, X. Sun, Binary differential evolution with self-learning for multi-objective feature selection, Information Sciences 507 (2020) 67–85. doi:10.1016/j.ins.2019.08.040.

[40] G. Krishna, V. Ravi, Feature Subset Selection Using Adaptive Differential Evolution: An Application to Banking, in: ACM India Joint Int. Conf. on Data Science and Management of Data, 2019, pp. 157–163. doi:10.1145/3297001.3297021.

[41] D. Dua, C. Graff, UCI machine learning repository (2017). URL http://archive.ics.uci.edu/ml

[42] Z. Zhu, Y. Ong, M. Dash, Markov blanket-embedded genetic algorithm for gene selection, Pattern Recognition 40 (11) (2007) 3236–3248. doi:10.1016/j.patcog.2007.02.007.

[43] A. Whitney, A Direct Method of Nonparametric Measurement Selection, IEEE Trans. on Computers 100 (9) (1971) 1100–1103. doi:10.1109/T-C.1971.223410.

[44] T. Marill, D. Green, On the effectiveness of receptors in recognition systems, IEEE Trans. on Information Theory 9 (1) (1963) 11–17. doi:10.1109/TIT.1963.1057810.

[45] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. Witten, The weka data mining software: an update, ACM SIGKDD explorations newsletter 11 (1) (2009) 10–18. doi:10.1145/1656274.1656278.

[46] J. Durillo, A. Nebro, jMetal: A Java framework for multi-objective optimization, Advances in Engineering Software 42 (10) (2011) 760–771. doi:10.1016/j.advengsoft.2011.05.014.

[47] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58. doi:10.1016/j.orp.2016.09.002.

[48] M. Friedman, The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance, Journal of the American Statistical Association 32 (200) (1937) 675–701. doi:10.1080/01621459.1937.10503522.

[49] G. Hommel, A stagewise rejective multiple test procedure based on a modified Bonferroni test, Biometrika 75 (2) (1988) 383–386. doi:10.1093/biomet/75.2.383.

[50] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms¢¢ behaviour: a case study on the CEC¢¢2005 special session on real parameter optimization, Journal of Heuristics 15 (6) (2009) 617. doi:10.1007/s10732-008-9080-4.

[51] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1 (1) (2011) 3–18. doi:10.1016/j.swevo.2011.02.002.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: