Cluster Computing (2023) 26:1949–1983
https://doi.org/10.1007/s10586-022-03725-w

Scalable feature subset selection for big data using parallel hybrid evolutionary algorithm based wrapper under apache spark environment

Yelleti Vivek[1,2] · Vadlamani Ravi[1] · P. Radha Krishna[2]

Received: 21 November 2021 / Revised: 17 June 2022 / Accepted: 20 August 2022 / Published online: 10 September 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract
Extant sequential wrapper-based feature subset selection (FSS) algorithms are not scalable and yield poor performance when applied to big datasets. Hence, to circumvent these challenges, we propose parallel and distributed hybrid evolutionary algorithms (EAs) based wrappers under Apache Spark. We propose two hybrid EAs based on the Binary Differential Evolution (BDE), and Binary Threshold Accepting (BTA), namely, (i) Parallel Binary Differential Evolution and Threshold Accepting (PB-DETA), where BDE and BTA work in tandem in every iteration, and (ii) its ablation variant, Parallel Binary Threshold Accepting and Differential Evolution (PB-TADE). Here, BTA is invoked to enhance the search capability and avoid premature convergence of BDE. For comparison purposes, we also parallelized two state-of-the-art algorithms: adaptive DE (ADE) and permutation based DE (DE-FS^{PM}), and named them PB-ADE and P-DE-FS^{PM} respectively. Throughout, logistic regression (LR) is employed to compute the fitness function, namely, area under the receiver operator characteristic curve (AUC). The effectiveness of the proposed algorithms is tested over the five big datasets of varying dimensions. It is noteworthy that the PB-TADE turned out to be statistically significant than the rest. All the algorithms have shown the repeatability property. The proposed parallel model attained a speedup of 2.2–2.9. We also reported feature subset with high AUC and least cardinality.

Keywords Feature subset selection · Differential evolution · Threshold accepting · MapReduce · Apache spark · Multithreading

1 Introduction

Selecting relevant and important feature subset is a paramount pre-processing step in the Cross-industry standard process for data mining (CRISP-DM) methodology [1] of data mining projects. This process of selecting the important group of features is popularly known as Feature subset selection (FSS) [2, 3]. The main objective of FSS is to select the most relevant and highly discriminative feature subsets. The spectacular benefits of FSS are as follows: it improves the comprehensibility of the models, reduces the model complexity, improves the training time, avoids overfitting, and sometimes improves the model's performance. Further, the resulting model becomes parsimonious. The ubiquitous presence of big datasets in every domain made FSS a mandatory pre-processing step.

FSS can be performed primarily in three distinct ways: filter, wrapper, and embedded approaches. The main difference lies in the fitness value computation and selecting the salient features individually or group-wise. The filter approaches measure the fitness value based on statistical measure such as Information gain, Mutual information, Gain ratio, etc. These methods are fast but result in less

✉ Vadlamani Ravi
vravi@idrbt.ac.in

Yelleti Vivek
yvivek@idrbt.ac.in

P. Radha Krishna
prkrishna@nitw.ac.in

[1] Center of Excellence in Analytics (renamed as Center for AI and ML), Institute for Development and Research in Banking Technology, Castle Hills Road #1, Masab Tank, Hyderabad 500057, India

[2] Department of Computer Science and Engineering, National Institute of Technology, Warangal 506004, India

accuracy and cannot account for the interaction effects amongst the features. Wrapper approaches comprise a metaheuristic optimization algorithm that searches for the best feature subsets as indicated by the highest fitness value determined by a classifier (for a classification problem) or a regression model (for a regression problem). These approaches are computationally intensive but highly accurate while accounting for the interaction effects of the features. In embedded approaches, the feature subset selection is embedded as a part of the model-building phase. These approaches combine the advantages of being less computationally expensive than wrapper and give better accuracy than filter approaches. Even though wrapper approaches impose high complexity, the selected feature subset is highly generalized to the underlying classifier.

In the current study, FSS is formulated as a combinatorial problem because, for a given n features, the total possible number of feature subsets is $2^n - 1$. Accordingly, the total number of feature subsets that can be formed constitutes the search space. The objective is to search for an efficient feature subset that comprises less redundant features. The best feature subset is found by checking all the possible feature subsets. However, this brute force method becomes unwieldy when the feature space dimension n become large, as in big datasets. Metaheuristics (evolutionary algorithms (EAs) subsumed) have demonstrated their superiority over conventional optimization methods in solving various combinatorial and continuous optimization problems. Metaheuristics are of two different types: (i) point-search-based methods such as Threshold Accepting (TA), Simulated Annealing (SA) etc., (ii) EAs, which are population-based EAs such as genetic algorithm (GA), differential evolution (DE), particle swarm optimization (PSO) etc. Recently, many new population based metaheuristics were also proposed such as monarch butterfly optimization (MBO) [4], slime mould algorithm (SMA) [5], moth search algorithm (MSA) [6], hunger games search (HGS) [7], colony predation algorithm (CPA) [8], and Harris hawks optimization (HHO) [9]. All of these algorithms were applied to solve many machine learning and engineering problems [2, 3, 10–14]. Between these two types, the point-search methods consume a lot of time to converge because their exploration capability is weaker than that of the population-based EAs [15]. The strategy of performing population based search entails the EAs inherent parallelism. Hence, the population-based EAs are considered to be robust in maintaining both the exploration and exploitation capabilities. Often the population based EAs outperformed point-search methods in various engineering problems [15, 16]. The property of inherent parallelism of the EAs is exploited in this current study to bring in process parallelization via Apache Spark. The present research study posed the FSS in a single objective environment where the objective function maximizes the AUC, thereby selecting the feature subsets of length less than or equal to k (where $k < n$) while achieving the best possible AUC.

As the data is generated in large volumes at a phenomenal rate, scalability becomes a major concern while developing solutions to analyze such big data. Therefore, designing scalable solutions gained prominence. Several new programming models were proposed to compute such large-scale computations on the commodity hardware. Distributed frameworks like MapReduce [17] or Spark [18] provide scalability with an improved performance by drawing capabilities from the underlying commodity hardware. These frameworks provide inherent support mainly: data distribution, load balancing, fault tolerance, and parallel processing. MapReduce [17] is a distributed programming paradigm used in handling big data. It mainly consists of two steps: map and reduce. MapReduce solutions are proven to be scalable. There are different big data frameworks available to design MapReduce solutions. Among them, Spark is faster due to its in-memory computation feature. Spark is an open-source, fast computing distributed engine used to handle large amounts of data. Spark uses in-memory computing by using Resilient Distributed Datasets (RDDs) that boost the performance, thereby avoiding the disk-access. Spark RDD is inherently distributed, follows lazy evaluation, and is immutable. Apache Spark also provides versatility by combining other big data tools such as Hadoop.

Extant EA-based wrapper algorithms are sequential and limited to small datasets. Even though they can be applied to larger datasets, they perform poorly. In the current world, the data growth is phenomenal, thereby demanding the development of scalable wrappers for FSS. There is a growing need to develop such parallel wrappers for FSS in the context of big data. These motivated us to propose a couple of scalable wrappers for FSS in a single objective environment. In the current study, the objective function is to select the feature subset of length less than or equal to k (where $k << n$) while achieving the best possible AUC. To the best of our knowledge, no work has been reported so far, where feature subset selection is either performed by a scalable wrapper involving parallel and distributed EC techniques or their hybrids under the Apache Spark environment.

In the current study, we considered DE from the population-based EA, because it is proven to be the robust search algorithm [19] in solving the optimization problems. Although DE is a robust algorithm, it still suffers from premature convergence [12] and faces difficulty to converge in not linearly separable functions [19]. Hence, these drawbacks motivated us to develop the hybrid EAs which

are based on memetic approach. In this approach, the control shifts between the constituent algorithms at a logical point of time of the execution in order to enhance the search ability and increase the probability of avoiding premature convergence. TA is chosen from the point-based metaheuristic because it is a deterministic variant of simulated annealing and exploits the search space well and because of this nature, it is more suitable for combinatorial optimization problems. For the purpose of better hybridization, we developed a population based TA. We have designed the hybrid EAs comprising both the DE and TA. It is important to note that the current hybrid sequential DETA is distinct from the hybrid version presented in Chauhan and Ravi [20]. The DE-TA [20] is implemented as a two-stage hybrid model, was a loosely–coupled hybrid system, wherein DE is executed in stage-I till it is converged later, the TA was invoked in stage-II. However, the current hybrid (variants of DE and TA) is a tightly-coupled hybrid system, where DE and TA work in tandem and this scheme is executed every global iteration.

The major reasons behind proposing the parallel hybrid EA designed under the Apache spark environment are as follows: (i) Hybrid EA, being inherently parallel by design, are eminently amenable for process parallelization and hence suitable for analysing high-dimensional big data problems. (ii) Further, sequential implementations are not scalable to handle large, high dimensional datasets. Hence, the parallel algorithms are designed under Apache spark to make the algorithm suitable to analyze big datasets. (iii) The hybrid EA is designed to enhance the search ability so that it could reach the global optimal solution or near-global optimal solution. (iv) Also, the hybrid EA increases the probability of avoiding premature convergence because it has got high exploration capability by virtue of fusing TA with DE in every iteration.

This paper's significant contributions are as follows: (i) Parallel DE is designed under Apache Spark to develop wrappers for FSS. (ii) Binary versions of the TADE and DETA are proposed, developed and parallelized under Apache Spark. These are named as PB-TADE and PB-DETA, respectively. DETA was initially introduced as a part of the ablation study and it turned out to be significantly different from TADE. (iii) Then, these are invoked to develop wrappers for FSS, where logistic regression is chosen as the classifier to evaluate the fitness function, namely the AUC. (iv) To compare our methods with the state-of-the-art methods, we found that the adaptive differential evolution ADE [21] and permutation based differential evolution DE-FS^{PM} [22] are competitive. But they are sequential in nature. Hence, we parallelized these two algorithms and refer to them as PB-ADE and P-DE-FS^{PM} throughout the paper. (v) To achieve scalability and algorithm parallelization, we proposed a novel MapReduce-multithread based framework.

The remainder of the paper is structured as follows: Sect. 2 presents the literature review. Section 3 presents an overview of the evolutionary algorithms employed. Section 4 presents the proposed scalable, parallel and distributed wrapper. Section 5 describes the datasets, Sect. 6 discusses the results obtained. Finally, Sect. 7 concludes the paper.

2 Literature review

Differential evolution, one of the widely used algorithms for feature subset selection, is proposed by Storn and Price [23]. Table 1 presents the works on filter and wrapper sequential versions of DE, where feature selection is posed as a combinatorial optimization problem. Zhang et al. [24] proposed a modified DE with self-learning (MOFS-BDE). In [24], the authors had introduced three different operators, namely: (i) modified binary mutation operator based on the probability difference, (ii) one-bit purifying search operator (OPS) to improve the self-learning capability of the elite individuals, and (iii) non-dominated sorting in the selection phase to reduce the computational complexity involved in the selection operator. Vivekanandan and Iyengar [25] designed a two-phase solution, where the critical features are selected by the DE and fed into the integrated model of the feed-forward neural network and fuzzy analytic hierarchy process (AHP) [26]. Nayak et al. [27] proposed FAEMODE, a filter approach using elitism-based multi-objective DE. It can handle both the linear and non-linear dependency among the features via both the correlation coefficient (PCC) and mutual information (MI). Mlakar et al. [28] designed a multi-objective DE (DEMO) based wrapper for facial recognition systems. In their approach, the important features are initially extracted based on the histogram of the oriented gradient descriptor (HOG) and fed to the DEMO to find the Pareto optimal solutions. Khushaba et al. [29] proposed DE with a statistical repair mechanism, DEFS, for selecting the optimal feature subsets in datasets with varying dimensionality. In their proposal, the probability of the feature distribution is fed to DEFS by the roulette wheel. Hancer et al. [30] proposed filter-based DE, MIFS, where the fisher score determines the mutual relevance between the features and class labels. The features are assigned a rank based on their fisher score. Hancer [31] proposed a new multi-objective differential evolution (MODE-CFS) with two-stage mutation, (i) centroid-based mutation to perform clustering, and (ii) feature-based mutation to perform feature selection. Non-dominated sorting is applied to determine the Pareto optimal solution set. Ghosh et al. [32] proposed a self-

Table 1 Sequential versions of DE and its wrapper variants

Authors	# objectives	Algorithm	Wrapper (classifier)/filter	Parallel/sequential
Zhang et.al. [24]	Multi Objective	Self-Learning DE	Wrapper (KNN)	Sequential
Vivekanandan and Sriraman [25]	Single Objective	Modified DE	Filter	Sequential
Nayak et.al. [27]	Multi Objective	FAEMODE	Filter	Sequential
Mlakar et.al. [28]	Multi Objective	DE + HOG	Wrapper (SVM)	Sequential
Khushaba et.al [29]	Single Objective	DEFS	Filter	Sequential
Hancer [31]	Multi Objective	MODE-CFS	Filter	Sequential
Hancer et.al. [30]	Multi Objective	DE + MIFS	Filter	Sequential
Ghosh et.al. [32]	Multi Objective	SADE	Wrapper (Fuzzy-KNN)	Sequential
Bhadra and Bandyopadhyay [33]	Multi Objective	MoDE	Filter MI	Sequential
Bhaig [34]	Multi Objective	Modified DE	Wrapper (SVM)	Sequential
Almasoudy et.al. [35]	Multi Objective	Modified DE	Wrapper (ELM)	Sequential
Zorarpaci et.al. [36]	Single Objective	DE + ACO	Weka J48 classifier	Sequential
Srikrishna et.al[37]	Single Objective	Quantum DE	Wrapper (LR)	Sequential
Lopez et.al. [22]	Single Objective	DE-FS^{PM}	Wrapper (SVM)	Sequential
Al-ani [92]	Single Objective	DE + Wheel based strategy	Filter	Sequential
Zhao et.al. [38]	Single Objective	Modified DE	Wrapper (SVM)	Sequential
Hancer [39]	Multi Objective	DE	Filter(Fuzzy + Kernel)	Sequential
Li et.al. [40]	Single Objective	DE	Wrapper (SVM)	Sequential
Wang et.al. [41]	Single Objective	DE	Wrapper (KNN)	Sequential
Krishna and Ravi [21]	Single Objective	Adaptive DE	Wrapper (LR)	Sequential
Current Study	Single Objective	PB-TADE, PB-DETA, PB-DE, PB-ADE,P-DE-FS^{PM}	Wrapper (LR)	Parallel

adaptive DE (SADE) based wrapper for the feature selection in the hyperspectral image data. The selected feature subsets are fed into fuzzy-KNN to obtain accuracy. Bhadra and Bandyopadhyay [33] improved the modified DE. They proposed an unsupervised feature selection approach called MoDE with the objective functions as (i) average dissimilarity of the selected feature subset, (ii) average similarity of the non-selected feature subset, and (iii) the average standard deviation of the selected feature subset. All the objectives mentioned above use normalized mutual information. Baig et.al [34]. proposed modified DE-based wrapper for the motor imagery EEG having a high dimensional dataset. SVM is used here as a classifier. Almasoudy et al. [35] designed a wrapper feature selection based on modified DE. The authors considered Extreme Learning Machine (ELM) as a classifier and tested its effectiveness over the intrusion detection dataset NSL-KDD. Zorarpaci and Ozel [36] proposed a hybrid FS approach based on DE and Ant-Bee Colony (ABC), where the J48 classifier from Weka computes the fitness score. This hybrid model achieved a significant F-score with less cardinal feature subset than the stand-alone DE and stand-alone ABC.

Then, a quantum-inspired wrapper based on DE (QDE) with logistic regression as the classifier was proposed by Srikrishna et al. [37]. The authors have introduced the quantum crossover and quantum mutation operators. They also used LR as the classifier, and the results obtained by the QDE are better than the ADE algorithms. They also reported that QDE achieved better repeatability than the BDE. Lopez et al. [22] proposed a wrapper based on permutation DE, where the permutation-based mutation replaced the mutation operator. The diversity of the generated children solutions is controlled by using a modified recombination operator. Zhao et al. [38] developed a two-stage wrapper feature selection algorithm, where in the first stage, the fisher score and information gain are applied to filter the redundant features. Then in the second stage, the top-k features are passed to the modified DE to perform the feature selection on four different breast cancer datasets. Hancer [39], for the first time, used fuzzy and kernel measures as filters to calculate the mutual relevance and redundancy with DE to handle continuous datasets. Li et al. [40] designed DE-SVM-FS and compared it with the default SVM approach, and they demonstrated that the DE and SVM-based FS achieved better accuracy than the

stand-alone SVM. Wang et al. [41] proposed DE-KNN, where the KNN is the classifier. DE-KNN performed both the feature selection as well as the instance selection.

Along with DE, several other EAs [10, 42–45] considered the FSS problem as a combinatorial optimization problem. Khammassi and Krichen [46] proposed two schemes, namely, (i) The NSGA-BLR approach to handle binary-class datasets and (ii) The NSGA-MLR to handle multi-class network intrusion datasets. Chaudari and Sahu [47] proposed a binary version of the popular crow search algorithm (CSA) with time-varying flight in wrapper version BCSA-VF. Binary Dragon Fly algorithm (BDA) is proposed by Too and Mirjalili [48] by taking the Covid-19 dataset as a case study. Recently many new metaheuristics were proposed to solve FSS. Hu et al. [5], introduced the dispersed foraging strategy inspired by ant-bee colony (ABC), into the slime mould algorithm (SMA), and named it as dispersed foraging slime mould algorithm (DFSMA), to avoid former's premature convergence and to maintain population diversity. The dispersion foraging strategy works as follows: an adaptive dispersion rate (DR) is maintained which is responsible to maintain the diversity in the population. DR decreases as the number of iterations increases which regulates the movement towards the optimal solutions. Hu et al. [49], had improved the grey wolf optimizer (GWO) by introducing the following operators: (i) covariance matrix adaptation evolution strategy (CMAES) to improve the exploration ability, (ii) levy flight mechanism to improve search accuracy, and (iii) orthogonal learning (OL) strategy to predict the optimal search direction and named GWOCMALOL. The authors conducted Wilcoxon signed rank test and proved that proposed GWOCMALOL obtained superior results in terms of convergence speed and accuracy. Hu et al. [50], introduced chaotic local search (CLS) mechanism into the GWO and named SCGWO to avoid local optimal, improve global exploration capability, individual moment is randomness. SCGWO has apparent advantages in processing unimodal, multimodal and composition functions. Too et al. [51], proposed MEHHO which follows an energetic learning strategy which considers the global best experience to update the locations of the other search agents, memory saving and updating mechanism where the best solution is stored and hawk is allowed to imitate its best solution if the fitness value becomes worse. The authors applied their proposed algorithm to the classification of electromyography signals. Zhang et al. [52], embedded the salp swarm algorithm (SSA) and proposed improved harris hawks optimization (IHHO) to improve the searchability of the HHO. The update stage in the IHHO is done in the following way: (i) SSA generates the population and it is called SSA-based population. (ii) generate hybrid individuals using SSA and HHO. (iii) by using greedy selection and HHO's mechanism update the search agent. The authors demonstrated that the proposed IHHO achieved a faster convergence speed and maintained a better balance between exploration and exploitation. Several variants of DE are employed, for example, in the estimation of tool-wear during the milling process [11], optimal resource scheduling [12], energy-efficient model [13], and anomaly detection [14].

Now we shift our attention to the works more relevant to the current study. Several parallel and distributed versions of the evolutionary algorithms [53–56] are proposed to handle high-dimensional datasets and big data. Various parallel and distributed implementations of the DE are presented in Table 2. Zhou [57] discussed various strategies for implementing parallel DE MapReduce versions and their pros and cons in the Hadoop distributed framework. Teijeiro et al. [58] designed parallel DE under the Spark environment, and the experiments were conducted in the AWS cloud environment to solve benchmark optimization problems. Recently, Cho et al. [59] designed a parallel version of DE to solve large-scale clustering problems. Another parallel version of DE Classifier (SCDE) was proposed by Al-Sawwa and Ludwig [60] to handle the imbalanced data. SCDE found the optimal centroid and assigns the class to the data point based on the Euclidian distance.

Chen et al. [61] proposed a parallel version of the modified DE using single-program multiple-data (SPMD), with improved genetic operators. They employed both fine-grained and coarse-grained approaches for cluster optimization. Adhianto et al. [62] proposed a fine-grained parallel version of DE using OpenMP to solve the optical networking problem, where the shared memory multiprocessing is supported. Deng et al. [63] proposed a parallel DE for solving the benchmark functions and reported the speedup. He et al. [64] proposed the parallel framework for five variants of DE under the Spark cloud computing platform. They analyzed the speedup by solving the benchmark functions. Wong et al. [65] developed the Computed Unified Device Architecture (CUDA) framework for self-adaptive DE solving benchmark functions. Cao et al. [66] proposed a message passing interface (MPI) based co-evolutionary version of DE, where the population is divided and co-evolved together to solve large-scale optimization problems. Ge et al. [67] proposed an adaptive merge and split strategy for DE, namely, DE-AMS using MPI, to improve resource utilization, which is a vital aspect to minimize while handling large-scale optimization problems. Falco et al. [68] designed MPI-based DE under a CUDA grid environment and tested it on different resource allocation strategies. Veronse and Krohling [69] developed the first implementation of the CUDA version of DE. The proposed algorithm was tested on well-known benchmark

Table 2 Parallel and distributed versions of DE and its variants

Authors	Algorithm	Environment	Problem solved
Zhou [57]	DE	Spark	Pros and cons of various approaches are discussed
Teijeiroet.al. [58]	DE	Spark + AWS	Tested on benchmark functions
Chou et.al [11]	DE	Spark	Clustering
Al-Sawwa and Ludwig [60]	DE	Spark	Designed a DE based classifier
Chen et.al [61]	Modified DE	SPMD	Cluster Optimization
Adhianto et.al [62]	DE	OpenMP	Optical Network problem
Liu et.al. [93]	DE	Distributed Cloud	Power electronic circuit optimization
Deng et.al. [63]	DE	Spark	Tested on benchmark functions and reported speedup
Wong et.al. [65]	Self-Adaptive DE	CUDA	Tested on benchmark functions and reported speedup
He et.al. [64]	Five variants of DE	Spark + Cloud	Developed a ring topology model and evaluated benchmark functions to report speedup
Cao et.al. [66]	DPCCMOEA	MPI	Developed co-evolutionary based DE to solve large scale optimization
Ge et.al. [67]	DDE-AMS	MPI	Developed adaptive population model to solve large scale optimization
Falco et.al [68]	DE	MPI	Resource allocation
Veronse and Krohling [69]	DE	CUDA	To solve large scale optimization in GPU environment
Glotik et.al. [70]	PSADE	MATLAB	Hydro Scheduling algorithm
Thomert et.al. [73]	NSDE-II	OpenMP	Cloud work placements
Daoudi et.al [71]	DE	Hadoop	Clustering
Kromer et.al. [72]	DE	Unified Parallel C	To solve large scale optimization problems
Current study	PB-TADE, PB-DETA, PB-DE, PB-ADE,P-DE-FS^{PM}	Spark	A parallel EA based wrapper algorithm solving FSS

functions, and compared the computing time with the standalone implementation. Glotik et al. [70]. parallelized the DE using parallel MATLAB to solve the hydro-scheduling problem. Daoudi et al. [71]. developed the MapReduce version of DE under the Hadoop environment to solve the clustering problem. Kromer et al. [72] developed a parallel version of DE using Unified Parallel C to handle large-scale clustering problems. Thomert et al. [73] developed a parallel version of DE using OpenMP to achieve the optimized workflow placement into the realm of practical utility.

In summary, the drawbacks observed in the extant approaches are as follows:

1. The existing EA based wrapper techniques, listed in Table 1, are sequential and are applied to small datasets. When applied to big datasets, sequential approaches will take large computation time, and result in poor performance. Hence, there is a need to develop parallel EA techniques and make them amenable to big datasets.
2. Moreover, the current parallel and distributed EA algorithms (refer to Table 2) are not yet applied to FSS problem.

These drawbacks motivated us to design and propose scalable, parallel EA-based wrapper techniques. As previously mentioned, the population-based EAs alone suffer from premature convergence. Hence to decrease the probability of premature convergence and to improve the search ability of the DE, a point-based metaheuristic viz., TA is invoked. Accordinlgy, we designed two parallel hybrid variants and named them PB-DETA and PB-TADE.

3 Overview of the evolutionary algorithms employed

Evolutionary algorithms (EAs) [19, 74, 75] effectively obtain global or near-global optimal solutions. The designing of the heuristics in EAs is inspired by natural selection of Darwinian evolution [19], and social behaviour. They start by initializing the population of solutions randomly. This population is evolved in order to find better solutions. For evolution, EAs utilize specialized heuristics to generate new solutions and compute the corresponding fitness score given by the fitness function or (objective function). EAs are perfect for both continuous and binary search spaces. This section discusses a detailed explanation of all the methods employed in the current work.

3.1 Solution encoding scheme

Kohavi and John [76] are the pioneers in proposing the wrappers for FSS by posing the FSS as a combinatorial optimization problem. Here, wrappers take the help of a classifier or a regression model since it may involve the fitness score evaluation of a given feature subset. Wrappers using metaheuristics (EAs subsumed) require the fundamental step of solution encoding. EAs randomly initiate the population consisting of a set of solution vectors. Each solution vector in the population represents a feature subset. Such a vector comprises bits, where 1 indicates the presence, and 0 indicates the absence of a feature. The dimension of this array is equal to the number of features in the dataset.

3.2 Binary differential evolution

Binary Differential Evolution (BDE), a stochastic population-based global optimization algorithm, includes the three heuristics, namely: mutation, crossover, and selection in that order. BDE starts by initializing the random population, consisting of n candidate solution vectors (X_i), where n is the population size. This candidate solution vector follows the binary encoding scheme. Each candidate solution vector is subjected to all the three heuristics in each iteration (or generation) of the algorithm.

In each generation t, in a dimensional search space (d), then the candidate solution vector (X_i^t) subjected to the mutation operation and produces the mutant vector (M_i^t). The mutation operation is applied as presented in Eq. (1):

$$M_i^t = X_{i1}^t + MF * \left(X_{i2}^t - X_{i3}^t\right) \tag{1}$$

where $X_{i1}^t, X_{i2}^t and X_{i3}^t$ are three randomly chosen distinct vectors from the current generation t. MF is the mutation factor, is a user-defined parameter, and lies in the range [0,1]. After this, the mutant vector may not be a binary anymore. Hence, sigmoid based discretization process (see Eq. 2.) is applied to every m_{ij}^t , j^{th} member of the M_i^t thereby converting a continuous vector into a binary vector.

$$m_{ij}^t = \begin{cases} 1, if rand(0,1) < sigmoid(m_j^t) \\ 0, else \end{cases} \tag{2}$$

Thus discretized mutant vector is subjected to crossover operation where it was subjected to the mating with the corresponding candidate solution vector to generate the trail vector. The crossover operation is applied to trail vector U_i^t, as presented in Eq. (3):

$$u_{ij}^t = \begin{cases} m_{ij}^t, if rand(0,1) < CR or j \neq randi \\ x_{ij}^t, if rand(0,1) \geq CR and j \neq randi \end{cases} \tag{3}$$

where j = 1,2,…d, u_{ij}^t is the j^{th} bit of U_i^t, rand(0,1) is the random number generated in the interval [0,1] from a uniform distribution. *randi* is a randomly chosen index to make sure that the generated trail vector is different from the mutant vector. CR represents the crossover rate, is a user-defined parameter, and lies in the range [0,1].

Finally, the fitness score is computed for the trail vectors. Then the selection operation is applied by comparing the corresponding target vectors and trail vector to produce an offspring. Better solutions survive and form the parent population for the subsequent iteration. The selection operation follows the rule as presented in Eq. (4):

$$X_i^{t+1} = \begin{cases} X_i, iff(X_i) > f(U_i) \\ U_i, otherwise \end{cases} \tag{4}$$

As mentioned earlier, this is continued till the completion of maximum iterations or other convergence criteria, if any, is met.

3.3 Binary threshold accepting

Dueck and Scheuer [77] proposed Binary Threshold Accepting (BTA) algorithm. Later, Ravi and Zimmermann [78] optimized a fuzzy rule-based classification model using modified TA. They developed the solution in three phases: feature selection was used as a preprocessing step, a modified fuzzy rule-based classification system was invoked over the selected feature subset, and finally, modified TA (MTA) was invoked to minimize the rule base size while guaranteeing high accuracy. Ravi et al. [79] proposed a modified TA (MTA) to minimize the number of rules in a fuzzy rule-based classification system. Then, Ravi and Zimmerman [80] proposed a continuous version of TA as an alternative to the backpropagation algorithm to overcome its limitations while training a neural network model. The trained neural network was utilized for feature selection, and the selected features were fed to the fuzzy

classifier optimized by the MTA proposed in [79]. Later, Ravi and Pramodh [81] proposed a principal component neural network architecture trained by TA for bankruptcy prediction of banks. Chauhan and Ravi [20] proposed a hybrid approach, involving DE and TA in that order to solve a set of benchmark unconstrained benchmark problems. This hybrid is a tightly-coupled system and is implemented as a two-stage model, where DE is executed first and then TA is executed.

Threshold Accepting is a deterministic variant of simulated annealing. BTA, a binary version of the TA (MTA in [79]), is presented in Algorithm 1. BTA is applied to a single solution. BTA performs a neighbourhood search by flip-flopping the bits in the current solution vector one at a time, starting them in the left-most position. Each flip flop yields one neighbourhood solution. If the first neighbourhood solution is not accepted, then the bit is reversed to its original value. Likewise, 2nd bit is flip-flopped so on and so forth until a neighbourhood solution is accepted. However, it is not necessary to exhaustively search all the neighbourhood solutions. BTA calculates the difference in the fitness score of the present and previous solutions. BTA accepts the neighbourhood solution if its fitness value is not much worse than that of the current solution.

Algorithm 1: Threshold accepting (TA) algorithm

```
1: Choose an initial configuration
2: Choose an initial THRESHOLD T > 0 and small number > 0

   perturbation of the old solution.
4: compute AE:=
      fitness (new solution) - fitness (old solution)
5: IF AE < T THEN
6:    old solution := new solution

8:    too many iterations THEN

10:   IF some time no change in quality anymore THEN stop
11:   GOTO opt
```

3.4 Adaptive differential evolution

Adaptive Differential Evolution (ADE) algorithm [82] is employed to perform FSS by Krishna and Ravi [21]. The main objective of the adaptive algorithms is to obviate the manual fine tuning of the hyperparameters, and help in faster and better convergence without compromising performance. This algorithm is different from the original DE in the following way:

(i) Instead of manually fine-tuning the hyperparameters MF and CR, ADE adjusts them adaptively by using the roulette wheel selection based on the success rate of the parameters.
(ii) Mutation and Crossover rate operations are performed in the same way as the Differential Evolution (refer to Sect. 3.1).

Like DE, ADE also starts by initializing the random population of size n and also follows the binary encoding scheme. Similarly, each candidate solution vector is subjected to all the three heuristics in each iteration (or generation) of the algorithm. Based on the success rate of the combinations, the MF and CR are adaptively adjusted by Roulette wheel selection. Inherently, there are 12 sets of parameters having different probabilities q_h, h = 1,2,3…H (here H = 12). Using h^{th} setting, the heuristics namely

Mutation as per the Eq. (1), and Crossover as per the Eq. (3) apply on the parent solution x_i resulting in generating an offspring solution y_i. If $f(y_i) > f(x_i)$, then the probability of h^{th} setting is adjusted by using Eq. (5).

$$q_h = \frac{n_h + n_o}{\sum_{j=1}^{H}(n_j + n_o)} \tag{5}$$

where n_h is the current combination success rate, $n_o > 1$ is a constant term to prevent the sudden change in the success rates by just one random success achieved by a setting. In each generation, the parent population is subjected to mutation (Eq. 1) and Crossover (Eq. 3) operations. But unlike previous algorithms, MF and CR are adaptively determined by the Roulette wheel selection based on the success rate. Then the selection operator applies, which results in better solutions. The better obtained solutions become the parent population for the subsequent iterations. This process continues till the completion of maximum iterations or other convergence criteria, if any, is met.

3.5 Permutation based differential evolution

Lopez et al. [22] proposed permutation-based Differential Evolution and named it DE-FS^{PM}. This is different from the traditional DE in the following way:

(i) Instead of a binary vector, here the population consists of the integer valued vector to encode all features of a dataset. Each feature is assigned with a number starting from 1 (one) i.e. and range between {1,2,..*nfeat*} and 0(zero) acts as a delimiter to select the features. For example, for the candidate solution X_{i1}^t= [7,3,5,8,6,0,1,4,2} where the left part of the 0(zero) is the selected feature subset, and the right part is the non-selected feature subset.

(ii) The permutation-matrix based mutation operator can be used to create mutant solutions. Here also the three solutions are randomly chosen namely, $X_{i1}^t, X_{i2}^t, X_{i3}^t$. But the difference is that here the permutation based mapping operations is done to generate the mutant vector. Initially, the permutation based mapping is done from $X_{i1}^t to X_{i1}^t from PM$. The size of the matrix, *PM* is *nfeat*nfeat,* where *nfeat* is the number of features. *PM* is a binary vector of size *nfeat *nfeat*. This is permuted to form a new permutation matrix, namely *PM'*. Then, *PM'* maps the X_{i1}^t and forms the mutant vector M_{i1}^t.

(iii) It also follows the same crossover operation as presented in Eq. (3). But the only difference is that it is applied on the integer valued vector instead of binary encoded vector. After applying this operation, there is a chance few of the features are placed in multiple locations and a few of the features may be discarded. By example, X_{i1}^t = [7,3,5,8,6,0,1,4,2], and X_{i2}^t= [3,5,2,6,1,0,7,8,4], might result in the trail vector, [3,**5,5,**6,1,0,7,8,4], which selects the feature 5 twice whereas feature 2 is discarded completely. Hence, a simple repair mechanism is implemented between the $X_{i1}^t and X_{i1}^t$ to ensure the feasible candidate solutions only, to ensure all the features are selected only once in the trail vector.

DE-FS^{PM}, starts by initializing the integer valued vector of population. Then, this vector is subjected to the permutation based mutation, which is followed by the Crossover operation Eq. (3) with the simple repair mechanism resulting in the generation of offspring population. Then the selection operation is applied as presented in Eq. (4), resulting in the new population. This process continuous till the completion of maximum iterations or other convergence criteria, if any, is met.

4 Proposed scalable, parallel and distributed wrapper

To perform FSS, we chose DE and TA, and built hybrids around them. DE explores and exploits the search space globally and is stochastic in nature. However, DE sometimes gets bogged down in local minima before convergence [12], thereby slowing down the convergence rate. If the search space becomes large, this phenomenon gets accentuated. Hence, it needs support from a local search-based optimization algorithm. Here, we chose to employ Binary Threshold Accepting (BTA) for that purpose. The deterministic way of accepting the candidate solutions in BTA helps in exploration, exploitation, and fast convergence.

Even though the EAs are intrinsically parallel, explicit parallel versions of EAs have to be designed so that they meet the following requirements: optimal utilization of the distributed resources, scalability, and low communication overhead. In general, the explicit parallelism from the population perspective of EAs is achieved by two main models:

Master–slave (MS) strategy [58], which is also known as the global model, has only a single global population. Here, the master takes the responsibility of applying metaheuristics (EAs subsumed) while the slave manages the fitness function evaluation.

The island strategy [58] is where the population is divided into islands, upon which the heuristics (operators) are applied independently.

The types mentioned above differ in the underlying topology and the migration rules to determine the communication between the nodes. A hybrid strategy can also be designed by combing the two strategies. In the current study, we proposed and designed a MapReduce multithreaded framework, which mimics the combination of the above-mentioned strategies.

The distributed framework like Apache Spark is not affected by the underlying topology because, in these frameworks, independent of the underlying topology, migrant solutions are broadcast to all the partitions.

The comparative analysis is carried out across PB-DE, PB-DETA, PB-TADE, PB-ADE, and PB-DE-FS^{PM} to establish the importance of hybrid global and local search optimization heuristics. We developed a parallel BTA, too, independent of the DE. If BTA alone is employed for FSS, then it consumed enormous computational time without yielding useful results. Hence, the comparative study excludes BTA. All the approaches follow the same solution encoding scheme and the population RDD encoding scheme as mentioned in Sect. 4.1. In what follows, algorithm-agnostic details of the parallelization strategy proposed here are presented in Sect. 4.2, while the algorithm-specific details of the parallelization strategy proposed here are presented in Sect. 4.3. Thus, Sects. 4.1 through 4.3, succinctly capture the proposed parallelization mechanism in an unambiguous manner.

Algorithm 2: Biased sampling driven population initialization.

```
Output: P: population RDD
1: While (i<n){
2:    While (j<nfeat){
3:          If rand() > 0.99 THEN
4:                Feature is selected
5:                }
```

4.1 Population RDD encoding scheme

Let the population, denoted by P, be initialized randomly using biased sampling from Uniform distribution between 0 and 1 as presented in Algorithm 2. Our objective is to select the less cardinal feature subsets yielding the highest AUC value. Hence, the parameter in the biased sampling is taken as 0.99. If the pseudo-random number is greater than 0.99, then the bit is assigned the value 1, indicating the feature's presence, and 0, otherwise, meaning the absence of the feature. Thus, according to Algorithm 2, the initialized population is taken as population RDD and the dataset as different RDD. The population RDD is presented in Fig. 1. The key is solution-id, and the first index of the value is the solution vector of binary type. It conveys which feature subset is selected. The second index stores the names or ids of the features, which helps us form the column-reduced dataset and the construction of pipeline RDD for the respective solutions. The last index stores the AUC corresponding to the solution vector.

4.2 Algorithm-agnostic details of the paralellization

This subsection overviews the proposed parallel model. This framework consists of three stages: (1) population initialization, (2) training phase (3) test phase. These stages are common for all the algorithms. In this section, we discuss the changes occurring to population RDD during these phases in a generic way to understand the workflow of the framework.

4.2.1 Changes occurring to population RDD during population initialization

This is stage 1, where the chosen algorithm initializes the population according to Algorithm 2 which follows the biased sampling driven population initialization. Thus initialized population is stored in a different RDD which follows the schema as depicted in Fig. 1. Complete details are presented in Sect. 4.1. All the necessary parameters associated with the algorithm are broadcasted to all the worker nodes. Once a variable is broadcasted, it is cached by the executor and utilized for other RDD operations, viz., transformations and actions. They are meant for read-only variables. By broadcasting, these variables are shared across the cluster, thereby reducing the communication overhead.

4.2.2 Changes occurring to population RDD during training phase

Once the population is initialized, its fitness score is computed. As mentioned earlier, the Population RDD is divided into different partitions, and each partition represents an island. These islands asynchronously undergo heuristics (which are algorithm specific) as a separate thread in Spark. As explained earlier, population RDD follows the schema as depicted in Fig. 1. The population information corresponding to the current generation is stored in the *binary vector* column of each solution. Using this binary vector information, the corresponding selected features are stored in the *selected feature* column of the population RDD. EAs spend most of the time computing the fitness value. Hence, adopting an asynchronous way of fitness score (namely, AUC) calculation is the major task. This is achieved by initiating the thread pool mapper, where the number of threads is equal to the number of worker nodes. One cannot create the number of threads arbitrarily because arbitrary creation of the threads may increase the communication overhead. Hence, after rigorous trials, the ideal size of the thread pool is found to be the number of worker nodes. The size of the thread pool also affects the speedup. A low-size thread pool leads to poor performance, whereas a higher thread pool leads to huge communication overhead.

Key	Value : Solution Vector
$Key_{\{1\}}$	$< \text{Binary Vector}_{\{1\}}, \text{selected Features}_{\{1\}}, \text{AUC Scores}_{\{1\}} >$
$Key_{\{2\}}$	$< \text{Binary Vector}_{\{2\}}, \text{selected Features}_{\{2\}}, \text{AUC Scores}_{\{2\}} >$
...	...
$Key_{\{n\}}$	$< \text{Binary Vector}_{\{n\}}, \text{selected Features}_{\{n\}}, \text{AUC Scores}_{\{n\}} >$

Fig. 1 Schema of the population RDD

Each thread in the thread pool is responsible for computing the AUC of the population island. Once this map is called, the reduced dataset is formed based on the feature subset information using the *selected feature* index of each solution. Then, the machine learning (ML) pipeline is constructed where the reduced dataset and LR model are bound together, thereby giving pipelined RDD. Such a generated pipelined RDD is a subclass of the RDD, an immutable and partitioned collection of elements where all the operations are executed in parallel. LR model is trained by creating the vector assembler that has to be created for the corresponding selected features. Thus, a vector assembler is created over the obtained reduced data frame. This kind of vector assembler is created for every solution in each iteration. As we have adopted the pipelined RDD, the above operations are executed in parallel and distributed across the nodes, thereby achieving fine-grained parallelism. Later, the AUC is evaluated with the training dataset in the same thread. All this process is repeated for each thread in the fitness value computation step. By adopting this strategy, the computation of AUC is performed in an asynchronous fashion. The same strategy is employed in all the proposed approaches to parallelize the fitness function evaluation. Thus, the AUC's are updated in the population RDD accordingly. This serves as the parent population. All the subsequent iterations follow the same thread pool mapper strategy to evaluate AUC.

As mentioned earlier, the binary encoded solution is stored in the *binary vector field*. Hence, by using the *binary vector field* information, the population is subjected to exploration and exploitation heuristics (which are algorithm specific) thereby forming the offspring population which also follows the same structure as the one depicted in Fig. 1. The corresponding selected feature is also updated using the binary vector information of thus formed offspring. Then the thread pool mapper is called, where the LR model evaluates the AUC on the reduced datasets. With the computed AUC of each solution, the corresponding AUC column is also updated. The selection operator is applied on both the parent and offspring populations, and the worst parent with less AUC is replaced with offspring having higher AUC. These better solutions are formed as one population, which serves as the parent population for the next iteration. It is important to note that all the algorithms follow a similar fitness score evaluation scheme. This process of computing AUC and selecting the better solutions, thereby achieving the evolution of the population, is repeated for *maxIter* number of iterations.

4.2.3 Changes occurring to population RDD during test phase

Thus, fully evolved population at convergence, which is nothing but the set of the required feature subsets obtained after convergence is evaluated on the test dataset in the test phase. The test AUC is computed and the corresponding solutions and test AUC are reported.

It is important to note that this stage is common for all the algorithms.

4.3 Algorithm-specific details of the parallelization

In this subsection, the discussion pivots on the proposed parallel algorithms. It is important to note that all the algorithms are subjected to the same framework as discussed in Sect. 4.2. Further, all the algorithms are identical in the following way: in the fitness score evaluation and test phase. Hence, the discussion on these two is obviated. The discussion is limpid more on the algorithm specific differences in detail i.e., population initialization and training phase.

4.3.1 Parallel binary differential evolutionary (PB-DE)

The proposed PB-DE based wrapper algorithm is presented in Algorithm 3, and the flowchart of the execution flow is depicted in Fig. 5 in the Appendix.

Population initialization

The PB-DE algorithm starts by initializing the population as mentioned in Sect. 4.2. The following information is broadcasted to all the nodes: mutation factor (*MF*), crossover rate (*CR*), number of features (*nfeat*), and population size (*n*). Then the AUC is evaluated by the LR on the initialized population and is done in an asynchronous fashion by creating the thread pool as mentioned in Sect. 4.2. All the necessary information is updated in the population RDD.

Training phase

Using the binary vector field information, the population is subjected to DE heuristics, viz., mutation and crossover thus forming the offspring population which follow the RDD structure depicted in Fig. 1. By utilizing the binary vector information of the offspring population information, the corresponding selected feature subset is updated. Then the thread pool mapper (refer to Sect. 4.2) is called, where the LR model evaluates the AUC on the reduced datasets. With the obtained AUC of each solution, their corresponding AUC column is also updated. The binary vector field information is extracted from parent and offspring population RDD which serves as the parent and offspring population. Then the selection operator is applied on both the parent and offspring populations, and the worst parent with less AUC is replaced with the better offspring with higher AUC. These better solutions are formed as one population, which serves as the parent population for the next iteration. This process of computing AUC and selecting the better solutions, thereby achieving the evolution of the population, is repeated for *maxIter* number of iterations.

Algorithm 3: Proposed PB-DE based wrapper

```
Input: P: population RDD, X: Input Dataset RDD, maxIter: maximum
iterations, MF: Muation Factor, CR: Cross over rate, n: population
size, nfeat: number of features

Output: P: Evolved population after maxIter

1: i← 0;
2: Initialize the population by biased sampling driven approach and
create population RDD; //Population initialization

   Island.
4: Create a thread pool of size number of nodes in the cluster;
     // Fitness score evaluation on Initial population

LR model and evaluate the AUC for a solution in the population;
6: Broadcast the variables MR, CR, n, nfeat;

7: While(i< maxIter){ // Training phase
8:    Each sub population subjected to DE heuristics via BDE Mapper;
9:    Create a thread pool of size number of nodes in the cluster;
10:   Evaluate the fitness function, each thread is responsible to
      train LR and evaluate the AUC for every solution in the
      population.
11:   Replace worst parent solutions with best children solutions;
12:   Update the population RDD;
13:   Increment i ← i+1;
14:   }
15: Evaluate on Test dataset // Test phase
16: return Population, test AUC.
```

Algorithm 4: Proposed parallel PB-DETA based wrapper

```
Input: P: population RDD, X: Input Dataset RDD, maxIter1: maximum DE
iterations, maxIter2: maximum TA iterations, MF: Mutation Factor, CR:
Cross over rate, n: population size, nfeat: number of features, T:
Threshold rate, eps: steps by which epsilon is decreasing after each
iteration

Output: P: Evolved population after maxIter1

1: i ← 0;
2: j ← 0;
3: T ← 0.05;
4: eps ← 0.95;
5: Initialise the population by using biased sampling driven
   approach and create population RDD; //Population initialization
6: Divide the population into partitions, each partition serves
   as an island;
    // Fitness score evaluation on Initial population
7: Create a thread pool of size number of nodes in the cluster;
8: Evaluate the fitness function, each thread is responsible to
   evaluate the AUC for a solution in the population;
9: Broadcast the variables MR, CR, n, nfeat;

10: While(i< maxIter1){ // Training phase
11:   Each subpopulation subjected to DE heuristics via BDE Mapper;
12:   Create a thread pool of size number of nodes in the cluster;

14:   Replace worst parent solutions with best children solutions;
15:   Update the population RDD;
16:   Increment i ← i+1;
17:   j ← 0;
18:   While(j<maxIter2){

20:         Create a thread pool of size number of nodes in the cluster;
21:         Evaluate the fitness function, each thread is responsible
            to evaluate the AUC for a solution in the population;

solutions;
23:         Update the population RDD;
24:         Increment j ← j+1;
25:         eps ← eps*(1-W);
26:         }
27:   }
28: Evaluate on Test dataset  // Test phase
29: return Population, test AUC.
```

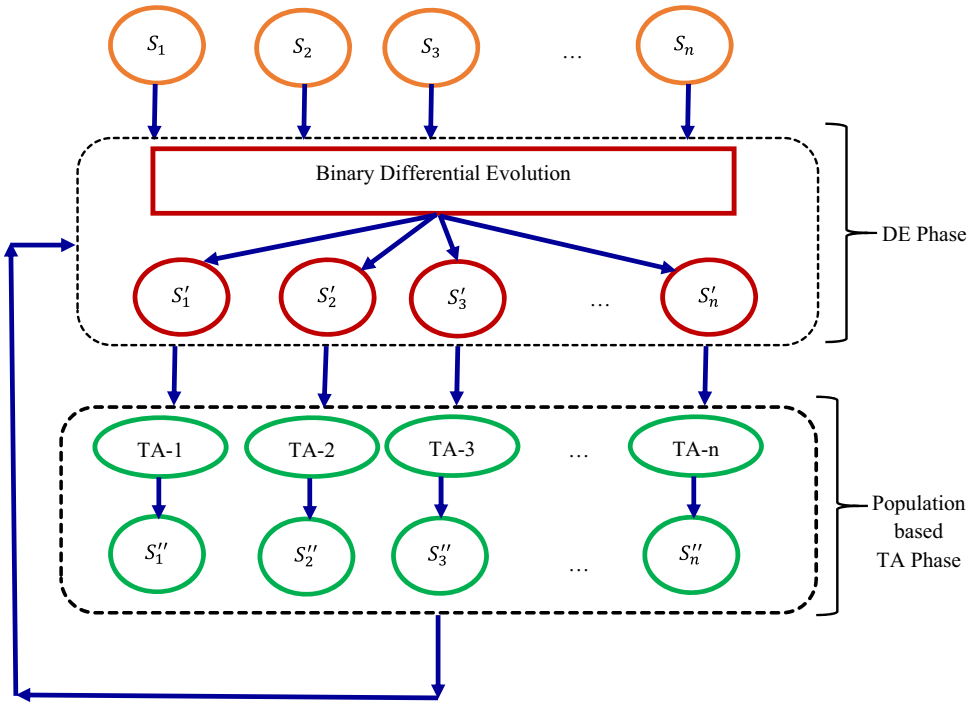

Fig. 2 Schematic representation of the DETA based wrapper

4.3.2 Parallel binary DETA (PB-DETA)

The proposed parallel approach PB-DETA is presented in Algorithm 4, the schematic representation is depicted in Fig. 2, and the flow chart of the execution flow is depicted in Fig. 6 in the Appendix. This hybrid variant of DE and TA is a tightly-coupled hybrid system, where DE and TA work in tandem and this scheme is executed every global iteration.

Population initialization

The population is initialized as per Algorithm 1 and then AUC is computed on this initialized population. All the initialized parameters such as *MF*, *CR*, *nfeat*, and *n* are broadcasted.

Training phase

As mentioned earlier, the BDE and BTA are executed in tandem i.e., one after another in one complete generation. Similarly, here also parent population is subjected to BDE heuristics, viz., crossover and mutation, thereby forming the offspring population. As mentioned in Sect. 4.2; now the thread pool mapper is called on the offspring population to evaluate the AUC, and the corresponding fields are updated. Then, the better offspring solutions replace the parent solutions. Thus evolved population is subjected to BTA heuristics as outlined in Algorithm 2.

Then, the thread pool mapper is called on, thus forming the offspring population to evaluate AUC, and also the corresponding fields are also updated. Here, the offspring solution, which is not much worse than the threshold limit value, replaces the solution in the parent population. BTA heuristics are invoked for *maxIter2* times. Thus emerged population after *maxIter2* times serves as the parent population. After this, the whole process of BDE and BTA in tandem is repeated until *maxIter1* number of iterations is completed.

Algorithm 5: Proposed parallel *PB-TADE* based wrapper.

```
Input P: Population RDD, X: Input Dataset RDD, maxIter1: maximum iterations
BDE invokes, maxIter2: maximum iterations BTA invokes, MF: Mutation Factor,
CR: Crossover Rate, n: population size, nfeat: number of features, T:
Threshold rate, eps: steps by which epsilon is decreasing after each
iteration

Output P: Evolved Population after maxIter1

1: i ← 0;
2: j ← 0;
3: T ← 0.05;
4: eps ← 0.95;
5: Initialise the population by using biased sampling driven approach
   and create population RDD; //Population initialization

island;
// Fitness score evaluation on Initial population
7: Create a thread pool of size number of nodes in the cluster;

   the AUC for a solution in the population;
9: Broadcast the variables MR, CR, n, nfeat;
10: While(i< maxIter1){ // Training phase
11:    j ← 0;
12:    While(j<maxIter2){

14:          Create a thread pool of size number of nodes in the cluster;
15:          Evaluate the fitness function, each thread is responsible
             to evaluate the AUC for a solution in the population;
16:          Replace parent solutions with not much worse children
             solutions;
17:          Update the population RDD;
18:          Increment j ← j+1;
19:          eps ← eps*(1 − W);
20:          }
21:    Each subpopulation subjected to DE heuristics via BDE Mapper;
22:    Create a thread pool of size number of nodes in the cluster;

       the AUC for a solution in the population;
24:    Replace worst parent solutions with best children solutions;
25:    Update the population RDD;
26:    Increment i ← i+1;
27:    }
28: Evaluate on Test dataset // Test phase
29: return Population, test AUC.
```

4.3.3 Parallel binary TADE (PB-TADE)

The proposed parallel approach PB-TADE is presented in Algorithm 5, the schematic representation is depicted in Fig. 3, and the flow chart is depicted in Fig. 7 in the Appendix. This hybrid variant of DE and TA is also a tightly-coupled hybrid system, where TA and DE work in tandem which means TA executes first, and the evolved population is given as input to the DE and this scheme is executed for every global iteration.

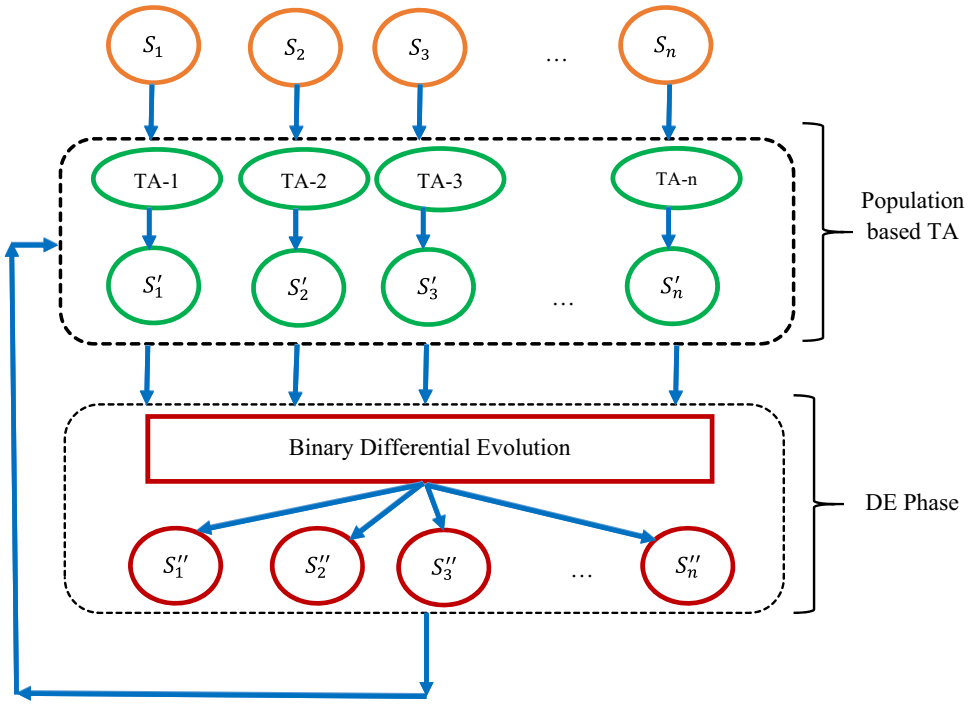

Fig. 3 Schematic representation of the PB-TADE based wrapper

Population initialization

The population is initialized as per Algorithm 1 and then AUC is computed on thus initialized population. All the initialized parameters such as *MF*, *CR*, *nfeat*, and *n* are broadcasted.

Training phase

This algorithm is a hybrid of BTA and BDE. Here, BTA is executed first for *maxIter2* times, followed by BDE. It means that first, the local exploitation happens in each solution individually. Later, it is followed by global exploration and exploitation over the entire population.

By utilizing the *binary vector* field information, the parent population, is subjected to BTA heuristic as presented in Algorithm 2, thus forming the offspring population. Then, AUC is evaluated by calling the thread pool mapper on the offspring population. The offspring solutions which are not much worse than the corresponding parent solution in terms of fitness value are selected. This process is continuted for *maxIter2* times which results in an evolved population that serves as the initial parent population for BDE. Thus evolved population obtained from BTA was subjected to the BDE heuristics, viz., crossover and mutation. After that, the thread pool mapper is called, and the AUC of the offspring population is evaluated. Then, the selection operation is applied, replacing the worse parent solutions with their better offspring solutions. This serves as the parent population for the next iteration. After this, the whole process of BTA and BDE in tandem is repeated until *maxIter1* iterations are completed.

4.3.4 Parallel binary ADE (PB-ADE)

The proposed parallel approach PB-ADE is presented in Algorithm 6, and the flow chart is depicted in Fig. 8 in the Appendix. As this is also implemented in the same environment, P-DE-FS^{PM}, also follows the same structure, the difference is the adoption of the MF and CR parameters, as mentioned in detail in Sect. 3.4.

Population initialization

The population is initialized as per Algorithm 1 and then AUC is computed on this initialized population. All the initialized parameters such as *MF*, *CR*, *nfeat*, and *n* are broadcasted.

Training phase

Using the binary vector field information, the parent population, is subjected to ADE heuristics as presented in Algorithm 6, thus forming the offspring population. Then, AUC is evaluated by calling the thread pool mapper on the offspring population. Then, the selection operation is applied, replacing the worse parent solutions with their better offspring solutions. This serves as the parent population for the next iteration. Now the rate of the success is calculated for the current MF and CR combination (refer to Eq. 5) and then stored accordingly. Then the roulette wheel is called in order to choose MF and CR. This whole process of applying heuristics and calling roulette wheel is repeated until *maxIter* iterations are completed.

Algorithm 6: Proposed PB-ADE based wrapper.

```
Input: P: population RDD, X: Input Dataset RDD, maxIter: maximum iterations,
MF: Mutation Factor, CR: Cross over rate, n: population size, nfeat: number
of features; qH: success rate probability vector.

Output: P: Evolved population after maxIter

1: i← 0;
2: Initialize the population by biased sampling driven approach and create
   population RDD; //Population initialization
3: Divide the population into partitions, each partition serves as an
   Island.
4: Create a thread pool of size number of nodes in the cluster;
     // Fitness score evaluation on Initial population
5: Evaluate the fitness function, each thread is responsible to train LR
   model and evaluate the AUC for a solution in the population;
6: Broadcast the variables n, nfeat;
7: While(i< maxIter){ // Training phase
8:    Each sub population subjected to ADE heuristics via BDE Mapper;
9:    Create a thread pool of size number of nodes in the cluster;
10:   Evaluate the fitness function, each thread is responsible to train LR
      and evaluate the AUC for every solution in the population.
11:   Replace worst parent solutions with best children solutions;
12:   Update the population RDD;
13:   Update the qH (refer Eq.(5)); update the CR and MR parameters by
      roulette wheel
14:   Increment i ← i+1;
15:   }
16: Evaluate on Test dataset // Test phase
17: return Population, test AUC.
```

4.3.5 Parallel permutation based DE (P-DE-FS.^{PM})

The proposed parallel approach P-DE-FS^{PM} is presented in Algorithm 7, and the flow chart is depicted in Fig. 9 in the Appendix. P-DE-FS^{PM}, has some special heuristics which are permutation inspired and require the integer value of the solution. Hence, along with the binary encoded form of the solution, the integer valued vector is also stored to undergo the underlying permutation based heuristic as depicted in Fig. 9.

Algorithm 7: Proposed P-DE-FS^{PM} based wrapper.

```
MF: Muation Factor, CR: Cross over rate, n: population size, nfeat: number
of features

Output: P: Evolved population after maxIter

1: i← 0;
2: Initialize the integer encoded vector of the population & create
population RDD; //Population initialization
3: Divide the population into partitions, each partition serves as an
   Island.
4: Create a thread pool of size number of nodes in the cluster;
     // Fitness score evaluation on Initial population
5: Evaluate the fitness function, each thread is responsible to train LR
model and evaluate the AUC for a solution in the population;
6: Broadcast the variables MR, CR, n, nfeat;
7: While(i< maxIter){ // Training phase

9:    Create a thread pool of size number of nodes in the cluster;
10:   Evaluate the fitness function, each thread is responsible to train LR
      and evaluate the AUC for every solution in the population.
11:   Replace worst parent solutions with best children solutions;
12:   Update the population RDD;
13:   Increment i ← i+1;
14:   }
15: Evaluate on Test dataset // Test phase
```

Population initialization

PB-DE-FS^{PM} algorithm requires the initialization of the integer valued vector (refer to Sect. 3.5). The delimiter of the selected and non-selected feature is '0', where the right side portion is the selected feature subset. Based on the selected feature subset, the binary vector is generated and thus forms the population following the structure as depicted in Fig. 4 as per the chosen population size (n). The major difference between the previous and current population RDD structure is the additional storage of the integer valued vector. Similarly, all the initialized parameters such as *MF*, *CR*, *nfeat*, and *n* are broadcasted. The same strategy which is used in the fitness function in earlier algorithms is also used here. Then AUC is computed on the initialized population.

Training phase

Using the binary vector field information, the parent population is subjected to DE-FS^{PM} heuristics namely the permutation based mutation, crossover and the repair mechanism as presented in Algorithm 7, thus forming the offspring population. Now after applying the heuristics by using the integer value information the corresponding binary form of the vector is generated and updated in the corresponding column. As mentioned in Sect. 4.2, the AUC is evaluated by calling the thread pool mapper on the offspring population. Then, the selection operation is applied thereby replacing the worse parent solutions with their better offspring solutions. This serves as the parent

Fig. 4 Schema of the population RDD for the P-DE-FS^{PM}

Key	Value : Solution Vector
$Key_{\{1\}}$	$< \text{Binary Vector}_{\{1\}}, \text{Integervalued Vector}_{\{1\}}, \text{selected Features}_{\{1\}}, \text{AUC Scores}_{\{1\}} >$
$Key_{\{2\}}$	$< \text{Binary Vector}_{\{2\}}, \text{Integervalued Vector}_{\{2\}}, \text{selected Features}_{\{2\}}, \text{AUC Scores}_{\{2\}} >$
...	...
$Key_{\{n\}}$	$< \text{Binary Vector}_{\{n\}}, \text{Integervalued Vector}_{\{3\}}, \text{selected Features}_{\{n\}}, \text{AUC Scores}_{\{n\}} >$

population for the next iteration. The whole process is repeated until *maxIter* iterations are completed.

4.4 Classification algorithm

Logistic Regression (LR) is employed as the classifier for the proposed FSS. LR is chosen because it is fast to train and is nonparametric. It does not make any assumptions about the errors or variables and has no hyperparameters to fine-tune.

4.5 Fitness function

Area under the ROC Curve (AUC) is the fitness function for our proposed wrappers. It is proven to be a robust measure than accuracy for unbalanced datasets. It is defined as the average of specificity and sensitivity. The cut-off for the probability to label samples into one of the two classes is taken as 0.5. Accordingly,

$$AUC = \frac{Sensitivity + Specificity}{2} \tag{6}$$

where sensitivity is the ratio of the positive samples that are correctly predicted to be positive to all the predicted positive samples. This is also called True Positive Rate (TPR).

$$Sensitivity = \frac{TP}{TP + FN} \tag{7}$$

where TP is true positive and FN is false negative and specificity is the ratio of the negative samples that are correctly predicted to be negative to all the predicted negative samples. This is also called True Negative Rate (TNR).

$$Specificity = \frac{TN}{TN + FP} \tag{8}$$

Table 3 Time complexity of the algorithms

Algorithm	Time complexity
DE	O(P*d*$globalG_{max}$)
DETA	O($globalG_{max}$ * $localG_{max}$ * P * d)
TADE	O($globalG_{max}$ * $localG_{max}$ * P * d)
ADE	O(P*d*$globalG_{max}$)
DE-FS^{PM}	O(P*d*d*$globalG_{max}$)

where TN is true negative and FP is false positive.

4.6 Time complexity

This subsection estimates the computational complexity of all the algorithms listed in Table 3**.** Zielinski [83] is the pioneer in investigating the time complexity of differential evolution. Let the population size be P, handling d-dimensional space and assume that DE converged after $globalG_{max}$, maximum global generations. Then, the computational complexity of DE is O(P*d*$globalG_{max}$). TA is a point-based EA, hence it is applied only for a single solution. Then the complexity of TA handling d-dimensional space by executing for $globalG_{max}$ iterations, is O(d*$globalG_{max}$). As we designed the population based TA, then the complexity becomes O(P*d* $globalG_{max}$). The complexity of the TADE, where the global maximum iteration is $globalG_{max}$ and local iterations where the TA is executed is $localG_{max}$ then the complexity of the TADE becomes, O($globalG_{max}$ * (P * d + $localG_{max}$ * P * d)) = O($globalG_{max}$ * $localG_{max}$ * P * d). Similarly, the DETA complexity is also O($globalG_{max}$ * $localG_{max}$ * P * d). Hence, one can observe that the complexity of both the DETA and TADE is identical and as these algorithms are tightly coupled hybrids. The hybrid algorithms will have the more time-complexity than the native algorithms. The time complexity of the ADE is also the same as that of the DE because the underlying heuristics complexity is the same. Lopez et.al [22] mentioned the algorithm complexity of the DE-FS^{PM} is as O(P*d*d*$globalG_{max}$).

5 Dataset description

The meta-information of the benchmark datasets is presented in Table 4. All other datasets except for the Microsoft Malware dataset, contain categorical features. Thus, categorical features are handled by using the one-hot encoding mechanism. All the datasets pertain to binary classification problems. The Microsoft Malware dataset is accessed from the Kaggle repository [84], the IEEE malware dataset from the IEEE data port [85], whereas OVM_Omentum and uterus are genomic datasets from

Table 4 Description of the benchmark datasets

Name of the dataset	# objects	# features	# classes	Size of the dataset
Epsilon	5,00,000	2000	2	10.8 GB
Microsoft Malware	32,59,724	76	2	1.8 GB
IEEE Malware	15,00,000	1000	2	3.2 GB
OVM_Omentum	1584	10,935	2	108.3 MB
OVM_Uterus	1584	10,935	2	108.3 MB

Table 5 Hyperparameters for all the approaches

Dataset	PB-DE		PB-DETA		PB-TADE		P-DE-FS^{PM}	
	MF	CR	MF	CR	MF	CR	MF	CR
Epsilon	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
Microsoft Malware	0.8	0.9	0.8	0.9	0.8	0.9	0.8	0.9
IEEE Malware	0.8	0.9	0.8	0.9	0.8	0.9	0.8	0.9
OVM_Omentum	0.75	0.9	0.75	0.9	0.75	0.9	0.75	0.9
OVM_Uterus	0.85	0.9	0.85	0.9	0.85	0.9	0.85	0.9

open source OpenML datasets [86], and the Epsilon dataset from LIBSVM binary dataset repository [87].

6 Results and discussions

All the datasets are divided into training and test sets in the ratio of 80%:20%. Stratified random sampling is performed to maintain a similar proportion of the classes in the training and test datasets. It is well-known that the performance of the EC techniques is susceptible to changes in hyperparameters. Hence, after rigorous fine-tuning with several combinations, the hyperparameters are listed in Table 5. For each algorithm and the dataset, the population size is fixed at 10, and the maximum number of generations is taken as 20. In the case of PB-DE, the DE is executed for 20 generations. However, in the case of the two hybrids (PB-DETA and PB-TADE), DE and TA are individually executed for 10 generations each, thereby making it 20 generations in all. The algorithms PB-ADE and P-DE-FS^{PM}, are also executed for 20 generations. The ADE is an adaptive mode. All the algorithms are run for 20 runs to nullify the impact of the random seed, which is customary for all EC techniques. The top solutions that achieved the highest AUC in each run are considered to report the average highest AUC and the corresponding average cardinality over 20 runs (see Table 6). All the top results obtained in this study are highlighted in bold.

6.1 Environmental setup

All the experiments are conducted in a Spark-HDFS cluster with Spark version 2.4.4 [88] and Hadoop version 2.7, having one master node and four worker nodes with 32 GB RAM with Intel i5 8th generation.

6.2 Mean AUC comparative analysis

The mean AUC obtained is compared to establish the importance of proposed hybrid approaches over the PB-DE and the other baselines PB-ADE and P-DE-FS^{PM}.

The results in Table 6 show that PB-TADE can achieve the best AUC because PB-DE got stuck in the local minima. Both PB-DETA and PB-TADE avoided this as they have employed with TA either before or after DE. The advantage of finding the local search exploitation helps not to get entrapped in the local minima but also find the better maxima. The feature subsets selected by PB-DE achieved less accuracy when compared to both PB-DETA and PB-TADE. Also, the average cardinality obtained by PB-DE is relatively high compared to that of both the PB-DETA and PB-TADE. Both these cases are not ideal for obtaining an optimal solution. Invocation of TA is also necessary while designing the hybrid model. Hence, we worked on both possibilities as part of the ablation study. We designed parallel BTA, too, independent of the DE. If BTA alone is employed for FSS, it consumes enormous computational

Table 6 Average Cardinality and mean AUC obtained

Dataset	PB-DE		PB-DETA		**PB-TADE**		PB-ADE		P-DE-FS^{PM}	
	Avg Cardinality	Mean AUC	Avg. Cardinality	Mean AUC	Avg Cardinality	Mean AUC	Avg Cardinality	Mean AUC	Avg Cardinality	Mean AUC
Epsilon	617.3	0.7932	486.1	0.8029	457.7	**0.8089**	555.65	0.797	558	0.7971
Microsoft Malware	29.6	0.6872	21.7	0.7002	18.60	**0.7054**	16.95	0.682	15.8	0.6924
IEEE Malware	643.45	0.7929	477.9	0.8035	463.9	**0.8009**	463.5	0.790	499.55	0.7937
OVA_Omentum	47.28	0.8607	35.54	0.8722	26.15	**0.8817**	49.3	0.846	32.9	0.870
OVA_Uterus	37.3	0.8607	28.60	0.8712	27.12	**0.8802**	46.2	0.845	49.7	0.871

time with inferior results. Hence, the comparative study excludes BTA.

In the ablation experiment involving PB-TADE and PB-DETA, the former achieved little lesser average cardinality of the feature subsets than the latter. In addition to that, the mean AUC is a little less but quite comparable in the case of IEEE Malware and Microsoft Malware. The former outperformed the latter because (i) BTA is essentially very good at local search by virtue of it being a point-based algorithm and a deterministic variant of simulated annealing. (ii) even though we proposed a population-based BTA, in these hybrids, the hallmark of population-based evolutionary algorithm, namely passing on the knowledge learned by the individual solutions to one another from generation to generation, is conspicuously missing by design. Therefore, they are the most population size number of BTA instances running independently. (iii) Thus, in PB-TADE, after BTA does the exploitation well, the baton is passed on to the BDE, which is demonstrably superior in both exploration and exploitation. This process continues in every iteration. (iv) However, in PB-DETA, the BDE does the job of exploration and exploitation well before the BTA is invoked, which only minimizes the fitness values obtained by DE. Further, we should note that both BTA and BDE are run for 10 iterations each, which means that they are run with relaxed convergence criteria without adversely impacting the fitness value or the AUC. This is a significant departure from the traditional implementations of both BDE and BTA for solving combinatorial optimization problems, where they are typically run for many iterations for convergence. This strategy is designed to reduce computational time, primarily because we deal with big data sets in a distributed manner in this paper.

Further proposed hybrids PB-TADE and PB-DETA outperformed the PB-DE, PB-ADE, and PB-DE-FS^{PM} in terms of mean AUC. Among the latter three, PB-DE-FS^{PM} stands out to be the best algorithm because, in Microsoft Malware and Omentum datasets, its average cardinality is less than the PB-DETA with a little cost of AUC. Along with this P-DE-FS^{PM} achieved significantly comparable results with the PB-DETA viz., with high dimensional datasets in terms of AUC but the cardinality of the selected feature subset is higher than the later. Moreover, P-DE-FS^{PM} outperformed both the PB-DE and PB-ADE in most of the datasets.

Further, no feature subset selection work is reported in analyzing Microsoft Malware and IEEE Malware datasets to the best of our knowledge. In the Epsilon dataset, Peralta et al. [89] designed MapReduce for evolutionary feature selection. They used CHC as the evolutionary strategy, and logistic regression as the classifier and achieved a 0.6985 AUC with 721 features, whereas PB-TADE obtained an average AUC of 0.8098 with 457.7 average number of features. Moreover, Pes [90] conducted feature selection by using Symmetric Uncertainty (SU), while AUC the scores are computed using Random Forest (RF). They reported an AUC of 0.695 and 0.6 in the OVA_Uterus and OVA_Omentum datasets, respectively. However, they did not report the optimal number of features that obtained these scores. But, PB-TADE achieved an average AUC of 0.8802 and 0.8817. In comparison, the OVA_Uterus and OVA_Omentum datasets have an average number of features, 27.12 and 26.15, respectively. Thus, our proposed methods outperformed the state-of-the-art results in these datasets.

6.3 Repeatability

Repeatability is one of the critical criteria for determining how robust and stable the designed wrapper method is. The more an optimal feature or feature subset repeats itself, the more powerful the underlying EA is said to be. In this subsection, repeatability analysis is conducted in two ways: firstly, concerning the features repeated individually among the often-repeated feature subsets with the highest AUC and secondly, the repetition of a whole feature subset as corresponding to the highest AUC.

6.3.1 Repeatability of the individual features

All the most often repeated features part of an optimal feature subset with the highest AUC are identified and presented in Table 7. The features repeated for more than 50% of the total individuals obtained by 20 runs are considered and presented. Results accommodate the most repeated top five features selected by each approach. It turns out that the repeated features selected by PB-DETA and PB-TADE are almost identical whereas the features chosen by the PB-DE and PB-ADE are slightly different. Interestingly, the most repeated individual features selected by the PB-TADE are also selected by the PB-DETA and P-DE-FS^{PM} in most of the datasets.

6.3.2 Repeatability of the feature subsets

All the feature subsets that yielded the highest AUC and repeated often are reported in Table 8. The #s1 represents the cardinality of the most-repeated feature subset, and the corresponding AUC. In the case of the Epsilon dataset, PB-DE has selected 639 features resulting in 0.79. PB-TADE outperformed the rest in terms of AUC and also by selecting a less cardinal feature subset. In the Microsoft Malware dataset, PB-DE achieved a 0.69 AUC with 31 features, while both PB-DETA and PB-TADE could achieve a better AUC than PB-DE with a less cardinal feature subset. Also, the P-DE-FS^{PM} and PB-ADE

Table 7 Most often repeated features selected by each approach

Dataset	Approach	Most repeated features
Epsilon	PB-DE	1,3,5,7,9
	PB-DETA	1,3,6,12,19
	PB-TADE	1,3,6,12,19
	PB-ADE	1,3,5,7,9
	P-DE-FS^{PM}	1,3,6,12,9
Microsoft Malware	PB-DE	AVProductsInstalled,HasTpm,Isprotected,Census_OEMN_Name Identifier,SmartScreen
	PB-DETA	AVProductsInstalled,HasTpm,IsPassiveMode, OsSuite,SmartScreen
	PB-TADE	AVProductsInstalled,HasTpm,OsSuite, RipStateBuild,SmartScreen
	PB-ADE	AVProductsInstalled,HasTpm,Isprotected,Census_OEMN_Name Identifier,SmartScreen
	P-DE-FS^{PM}	AVProductsInstalled,HasTpm,OsSuite, RipStateBuild,SmartScreen
IEEE Malware	PB-DE	GetProcAddress,GetThreadId,Sleep,FindClose, RaiseException
	PB-DETA	GetProcAddress,GetLastError,Sleep,ReadFile, RaiseException
	PB-TADE	GetProcAddress,GetLastError,Sleep,ReadFile, RaiseException
	PB-ADE	GetProcAddress,GetThreadId,Sleep,FindClose, RaiseException
	P-DE-FS^{PM}	GetProcAddress,GetThreadId,Sleep,FindClose, RaiseException
OVA_Omentum	PB-DE	158765_at,201608_s_at, 206442_at,207096_s_at,210002_s_at
	PB-DETA	1554436_s_at, 201669_s_at, 20644_s_at, 207442_s_at,, 208970_s_at
	PB-TADE	1554436_s_at, 201669_s_at, 20644_s_at, 207442_s_at,, 208970_s_at
	PB-ADE	158765_at,201608_s_at, 206442_at,207096_s_at,210002_s_at
	P-DE-FS^{PM}	158765_at,201608_s_at, 206442_at,207096_s_at,210002_s_at
OVA_Uterus	PB-DE	205866_s_at,209682_s_at,217294_s_at, 222421_s_at,220148_s_at,
	PB-DETA	202125_s_at,205866_s_at,218132_s_at, 222421_s_at,222784_s_at,
	PB-TADE	202125_s_at,205866_s_at,218132_s_at, 222421_s_at,222784_s_at,
	PB-ADE	202125_s_at,205866_s_at,218132_s_at, 222421_s_at,222784_s_at,
	P-DE-FS^{PM}	202125_s_at,205866_s_at,218132_s_at, 222421_s_at,222784_s_at,

Table 8 Cardinalities and the corresponding AUC of the Top-most repeated feature subsets

Dataset	PB-DE		PB-DETA		PB-TADE		PB-ADE		P-DE-FS^{PM}	
	#s1	AUC	#s1	AUC	#s1	AUC	#s1	AUC	#s1	AUC
Epsilon	639	0.7967	564	0.8068	488	**0.8097**	505	0.797	555	0.801
Microsoft Malware	31	0.6983	24	0.7057	17	**0.7061**	17	0.7	20	0.70
IEEE Malware	550	0.7956	486	0.8057	487	**0.8058**	483	0.804	588	0.798
OVA_Omentum	55	0.8701	37	0.8723	33	**0.8779**	66	0.866	33	0.876
OVA_Uterus	37	0.8607	28	0.8712	27	**0.8802**	60	0.86	50	0.877

*Where #s1 is the cardinality of the top-most repeated feature subset

achieved better AUC with a less cardinal feature subset compared to that of PB-DE but achieved less AUC compared to that of PB-DETA and PB-TADE. Even though in the case of the IEEE Malware dataset, the cardinality of the selected feature subsets is the same for the PB-DETA and PB-TADE, the AUC is different because the selected features are not identical. The same is the case in OVA_Omentum and OVA_Uterus datasets. Hence, the results indicate that PB-TADE can achieve lower cardinality with better AUC than PB-DETA and PB-DE in all the datasets in terms of repeatability. Similarly, PB-DETA outperformed PB-DE. In terms of AUC, overall PB-TADE outperformed the rest of the algorithms. PB-DETA and P-DE-FS^{PM} achieved comparable performance in terms of AUC in almost all the datasets. Also, the PB-ADE achieved comparable performance except in Omentum and Uterus datasets.

6.4 Least cardinal feature subset with highest AUC

In this subsection, the least cardinal feature subset among the 20 runs with the highest AUC is discussed. The results are presented in Table 9. It turns out that the PB-TADE outperformed all the algorithms in detecting the least cardinal feature subset. At the same time, PB-DETA and PB-ADE stand second and third in detecting the least cardinal feature subset. Except for the OVA_Omentum dataset, PB-TADE achieved a better AUC with fewer features than the PB-DETA. In the case of Epsilon, even though PB-TADE and PB-DETA have achieved similar AUC but PB-TADE achieved a lesser cardinal feature subset. In all the datasets, PB-DE, PB-ADE, and PB-DE-FS^{PM} were outperformed by the proposed hybrid models. This fact further reinforces the role played by the BTA in this hybridization.

6.5 Speedup

Speedup is defined as the gain obtained by the parallel version of the algorithm with respect to the sequential algorithm executed on a single processor as follows in Eq. (9).

$$\text{Speed Up (S.U)} = \frac{\text{Time taken by Sequential Version}}{\text{Time taken by Parallel Version}} \tag{9}$$

This stands as one of the essential characteristics in evaluating the performance of the parallel version of the algorithm. The results are presented in Table 10. It is to be noted that speedup results are rounded off to two decimals. We observed that the proposed parallel algorithms achieved significant speedup. Speedup achieved ranges from 2.21 to 2.90 times by all proposed algorithms over their sequential counterparts in all datasets. As the number of nodes in the cluster is 4, the maximum possible speedup that could be achieved is 4. The linear speedup is not achieved because of the synchronization junctions in the parallel model.

6.6 Statistical testing of the results

The two-tailed t-test at a 5% level of significance and 38 (20 + 20–2) degrees of freedom is conducted pairwise on the three proposed algorithms and the other two parallelized baselines to make statistically valid statements about their performance. The results are presented in Table 11.

The null hypothesis is *H_0: both the algorithms are statistically equal*,

while the alternate hypothesis is, *H_1: both the algorithms are statistically not equal*.

It is very important to conduct the t-test to check whether the numerical superiority of algorithm A over B, occurred purely by chance or indeed due to the superior nature of the algorithm A. Hence, in this work, the means of the top AUC scores achieved by the best solution in the population by each approach over the 20 runs are considered for the t-test evaluation. Thus, a two-tailed t-statistic value is calculated. Using the t-statistic, the p-values are obtained. The p-value is compared with the level of significance to determine whether to accept null hypothesis H_0 or not. Hence, even though t-statistic is calculated, it is used to determine the p-value which in turn helps in deciding whether to accept or reject H_0. As the p-values for all datasets are less than 0.05, the null hypothesis is rejected, and the alternate hypothesis is accepted. We infer that PB-TADE is significantly different from PB-DE, PB-ADE and P-DE-FS^{PM} as the p-values are significantly small.

In summary, the results indicate that the PB-TADE achieved higher AUC with lesser cardinality when compared to the PB-DETA. However P-DE-FS^{PM}, and PB-ADE are standing second and third respectively, and outperformed the PB-DE in the majority of the datasets. The statistical analysis says that the PB-TADE is significantly different compared to the rest PB-DETA and PB-DE in both exploration and exploration. All the proposed parallel approaches achieved significant speedup compared to their sequential counterparts. Although PB-TADE and PB-DETA consumed more time than the PB-DE, it is mainly

Table 9 Least Cardinal Feature subset with the highest AUC

Dataset	PB-DE		PB-DETA		PB-TADE		PB-ADE		P-DE-FS^{PM}	
	#s1	AUC	#s1	AUC	#s1	AUC	#s1	AUC	#s1	AUC
Epsilon	588	0.7967	471	0.8068	**441**	**0.8068**	505	0.797	555	0.801
Microsoft Malware	27	0.6915	22	0.7007	**17**	**0.7061**	17	0.7	20	0.70
IEEE Malware	550	0.7956	484	0.8057	**487**	**0.8197**	388	0.799	464	0.804
OVA_Omentum	41	0.8504	**29**	**0.8723**	31	0.8699	66	0.866	27	0.864
OVA_Uterus	31	0.8504	24	0.8699	**26**	**0.8712**	60	0.86	48	0.870

*Where #s1 is the cardinality of the feature subset having least cardinal subset with highest AUC

Table 10 Speedup Analysis of parallel versions over the sequential ones

Dataset	Algorithm	Sequential E.T	Parallel E.T	S.U
Epsilon	PB-DE	12,780	4485	2.84
	PB-DETA	12,680	4361	2.90
	PB-TADE	**12,688**	**4369**	**2.90**
	PB-ADE	12,780	4485	2.84
	P-DE-FS^{PM}	12,912	4860	2.65
Microsoft Malware	PB-DE	16,412	6741	2.43
	PB-DETA	15,781	6447	2.44
	PB-TADE	**15,779**	**6432**	**2.45**
	PB-ADE	16,222	6741	2.40
	P-DE-FS^{PM}	15,640	7077	2.21
IEEE Malware	**PB-DE**	**20,412**	**8151**	**2.50**
	PB-DETA	19,793	7936	2.49
	PB-TADE	19,801	7938	2.49
	PB-ADE	20,517	8817	2.32
	P-DE-FS^{PM}	20,331	8996	2.26
OVA_Omentum	PB-DE	14,892	5428	2.74
	PB-DETA	14,651	5226	2.80
	PB-TADE	**14,689**	**5428**	**2.81**
	PB-ADE	14,891	5407	2.75
	P-DE-FS^{PM}	16,860	6882	2.45
OVA_Uterus	PB-DE	14,108	5368	2.62
	PB-DETA	13,979	5222	2.68
	PB-TADE	13,968	5378	2.68
	PB-ADE	**14,891**	**5407**	**2.75**
	P-DE-FS^{PM}	16,042	6712	2.39

Top results are highlighted in bold

*Where E.T is the execution time given in seconds

Table 11 Paired t-test results

Model	Parameter	Dataset				
		Epsilon	Microsoft Malware	IEEE Malware	OVA_ Omentum	OVA_ Uterus
PB-DE	t-statistic	7.72	4.62	8.045	4.168	3.69
vs PB-TADE	p-value	2.66×10^{-09}	4.25×10^{-05}	9.91×10^{-10}	0.00017	0.00069
PB-DETA	t-statistic	3.56	3.106	3.63	2.06	1.744
vs PB-TADE	p-value	0.001	0.0035	0.0008	0.045	0.0891
PB-ADE vs	t-statistic	7.21	3.648	5.22	3.54	6.084
PB-TADE	p-value	1.25×10^{-09}	0.0007	6.57×10^{-06}	0.0010	4.36×10^{-07}
P-DE-FS^{PM} vs	t-statistic	14.21	4.035	6.90	2.45	1.78
PB-TADE	p-value	8.54×10^{-17}	0.00025	3.32×10^{-08}	0.0186	0.0818

due to more function evaluations. Notably, both the hybrid variants resulting in optimal feature subsets are much better than those obtained by the PB-DE, PB-ADE and PB-DE-FS^{PM} in terms of higher AUC and less cardinality.

7 Conclusions and limitations

This paper develops the parallel versions of the BDE, BDETA, and BTADE and employs them for the wrapper based feature subset selection, where logistic regression is chosen as the classifier. We demonstrated their

effectiveness on five high-dimensional datasets taken from literature. The results indicate that the PB-TADE achieved higher AUC with lesser cardinality when compared to the PB-DETA, standing second, P-DE-FS^{PM}, and PB-ADE, giving the comparable performance than the PB-DE in the majority of the datasets. The statistical analysis says that the PB-TADE is significantly different compared to the rest PB-DETA and PB-DE in both exploration and exploration. Further, our proposed methods outperformed the state-of-the-art results, wherever the results were reported.

The limitations of the current work are as follows:

- The current study is conducted in a single-objective function environment where only AUC is considered as an objective function. In future, the current investigation will be carried out on the same datasets but in bi-objective and multi-objective environments.
- The proposed hybrid algorithms are non-adaptive in the sense that we need to manually tweak their hyperparameters rather than tweaking them adaptively.
- Further, we observed that the specific way of parallelizing the EAs in the current study consumes more time when the population size increases. Hence, the method of parallelizing EAs can further be improved.
- A valid alternative for the current proposed hybrid approach could be the parallel particle swarm optimization (PSO) and its hybrid variants with TA in either a tightly or loosely coupled manner. Also, several most recent meta-heuristics such as HGS, CPA, HHO etc., can also be parallelized.

The above-discussed limitations motivate our future work. The current investigation will be carried out on the same problem but in bi-objective and multi-objective environments. In the future, a novel optimization algorithm based on machine learning presented in Hooten et al. [91] can be employed for this problem.

Appendix

See Figs. 5, 6, 7, 8 and 9.

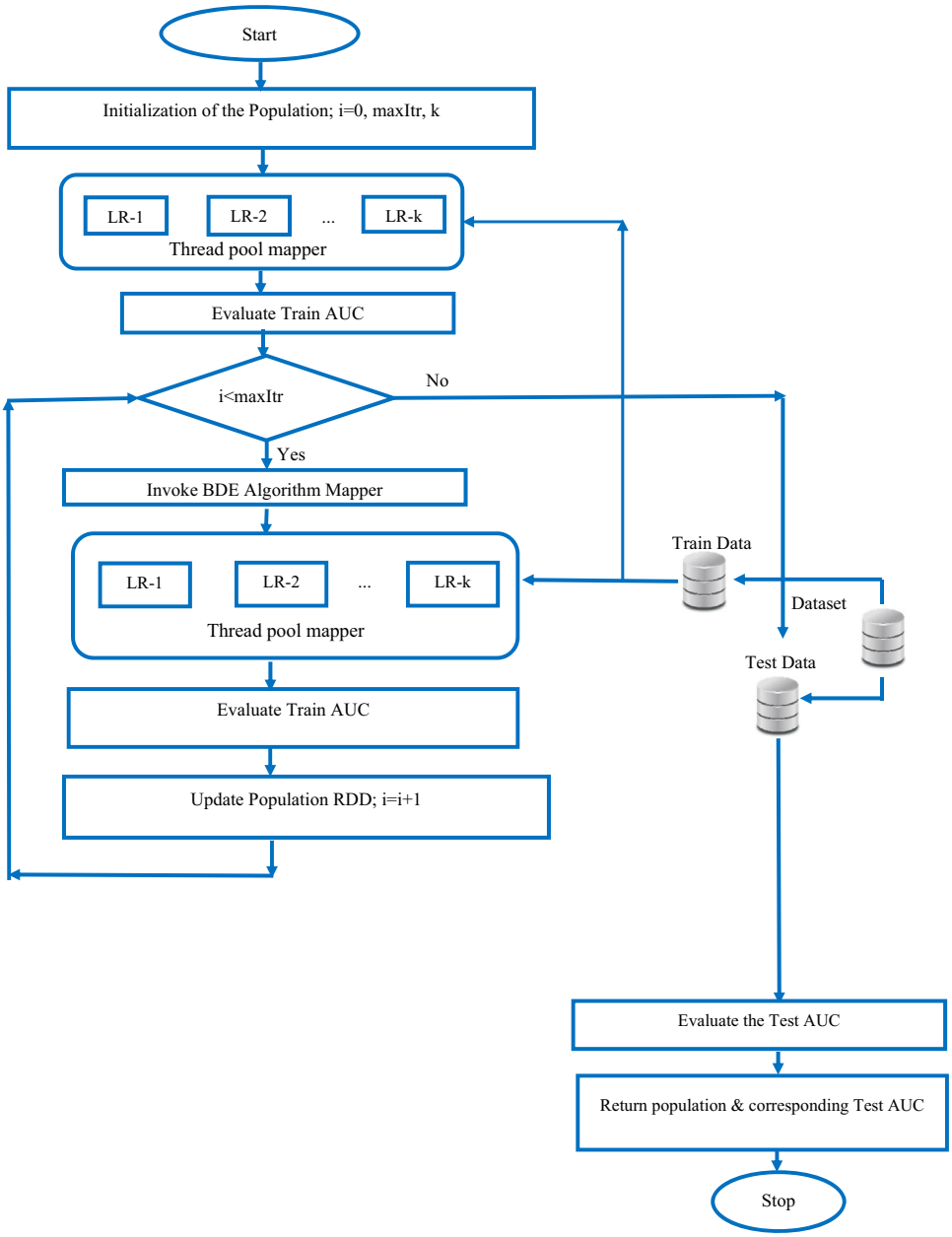

Fig. 5 Flowchart of PB-DE based wrapper

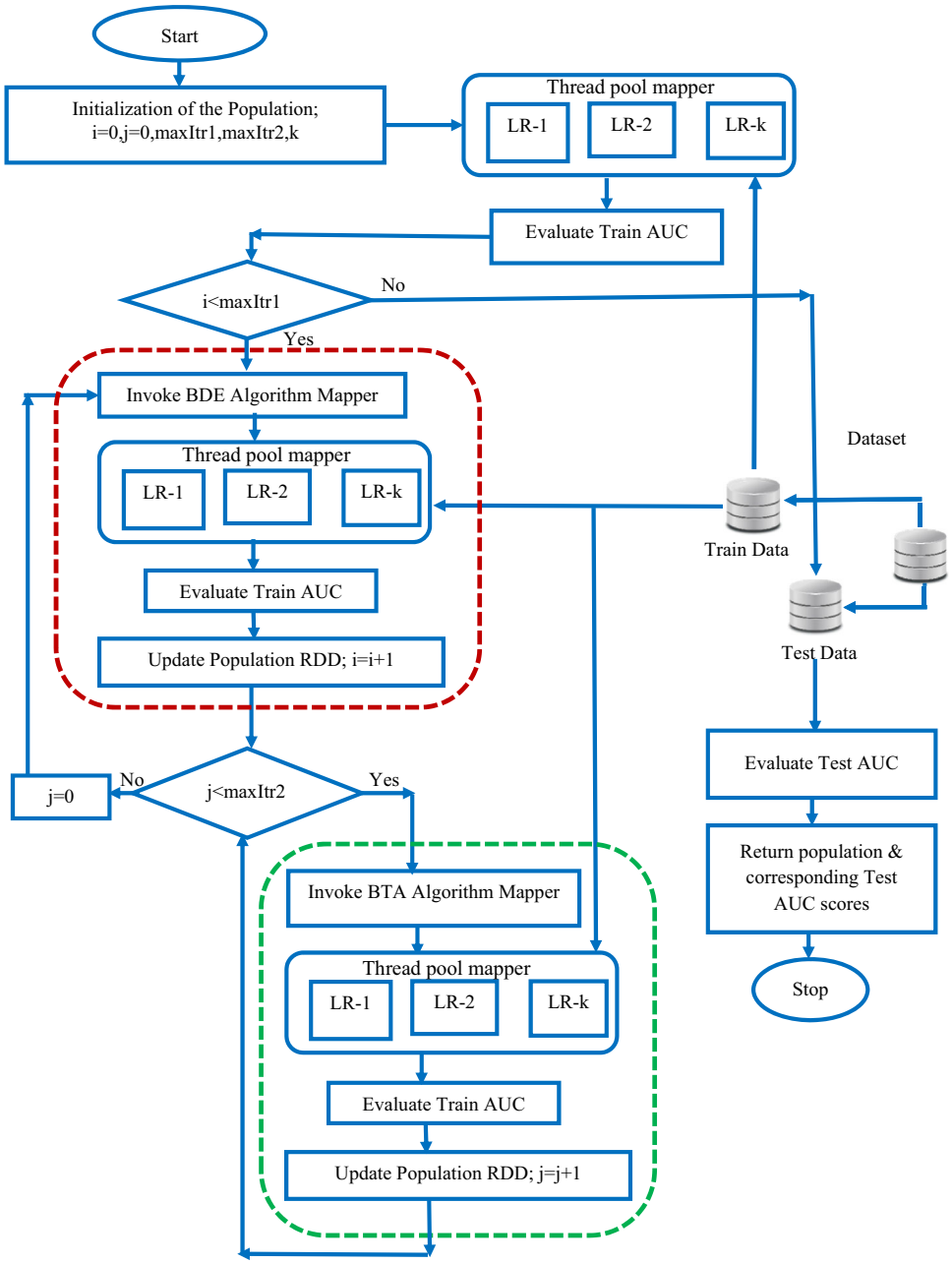

Fig. 6 Flow chart of the PB-DETA based wrapper

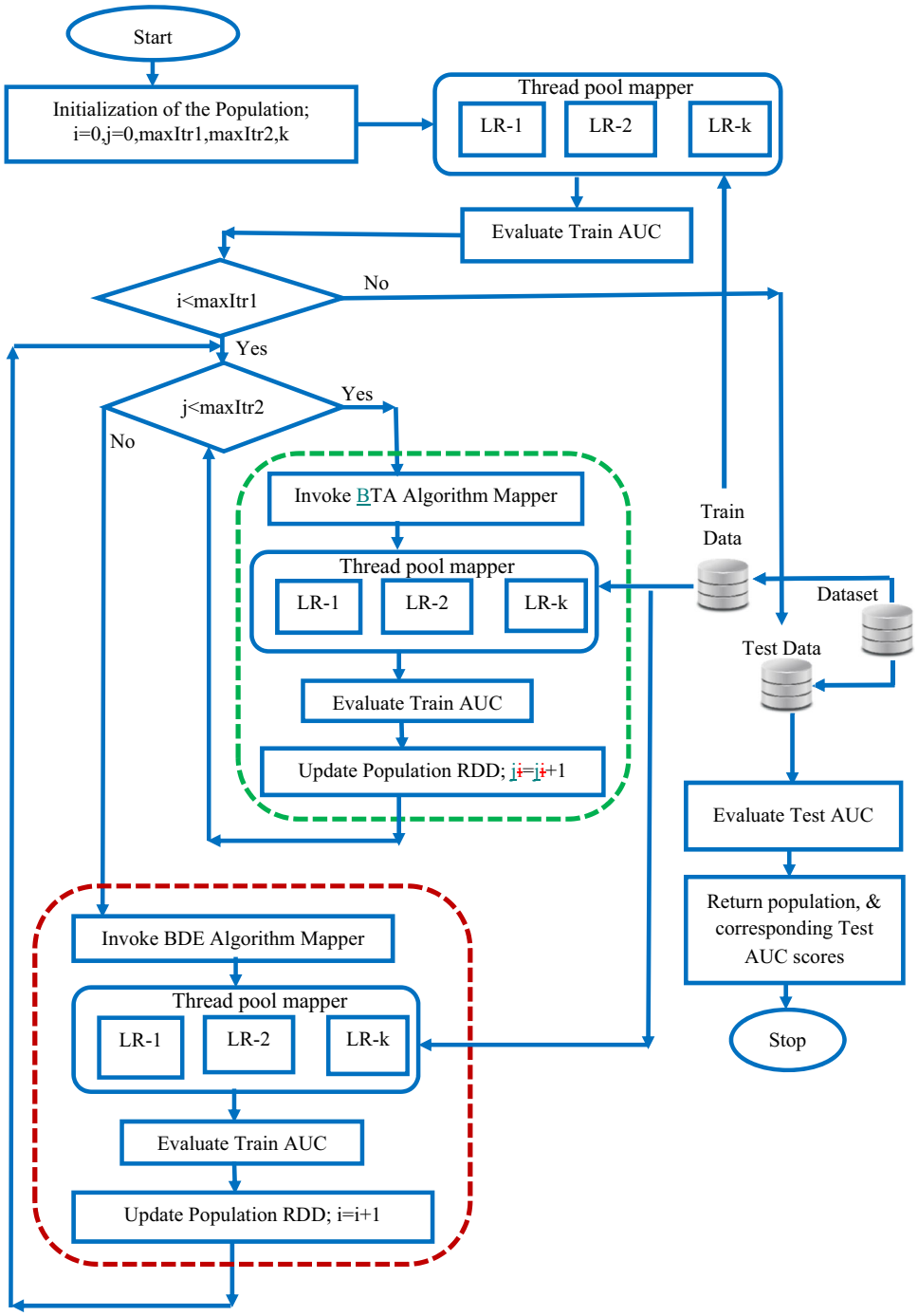

Fig. 7 Flowchart of the parallel PB-TADE based wrapper

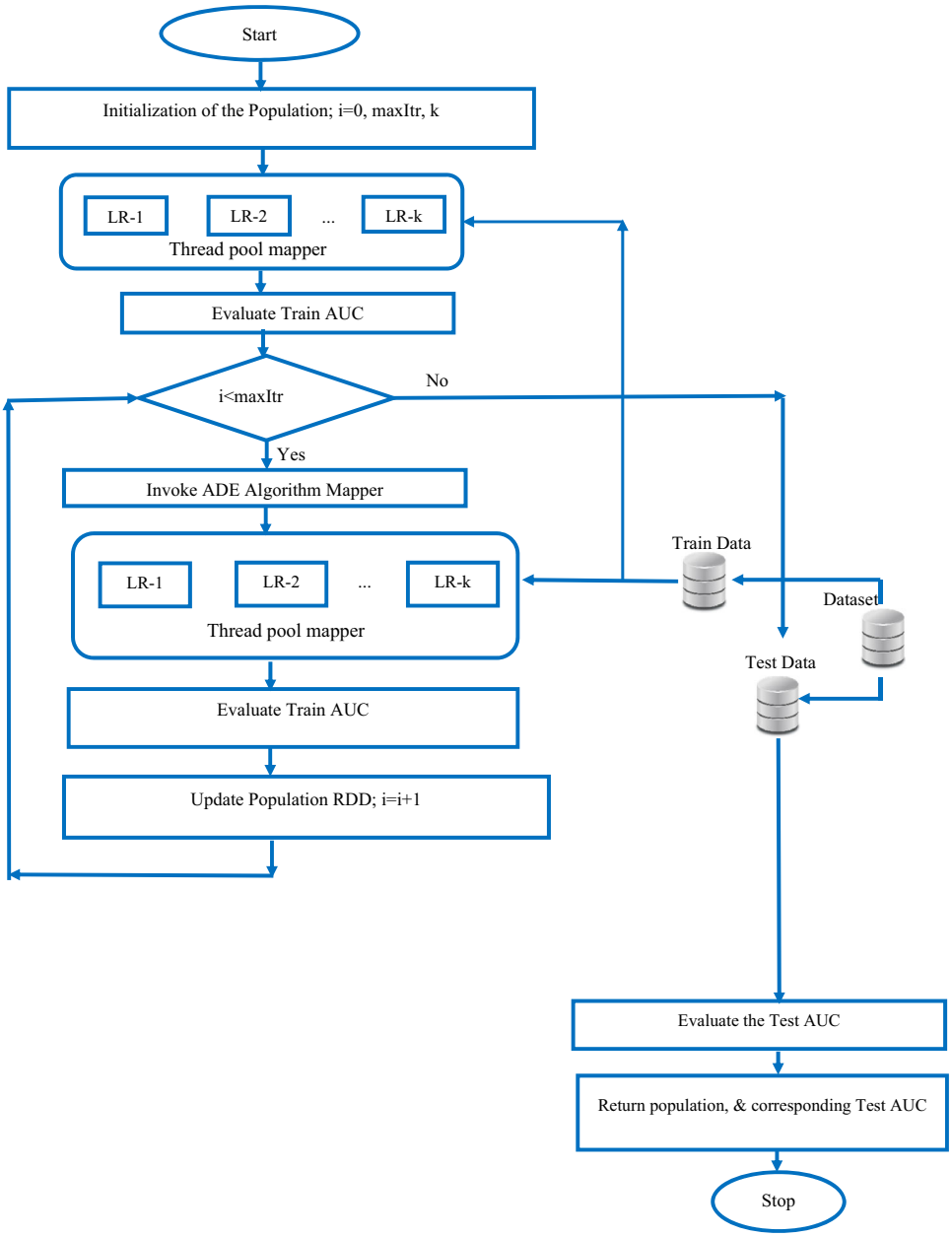

Fig. 8 Flowchart of PB-ADE based wrapper

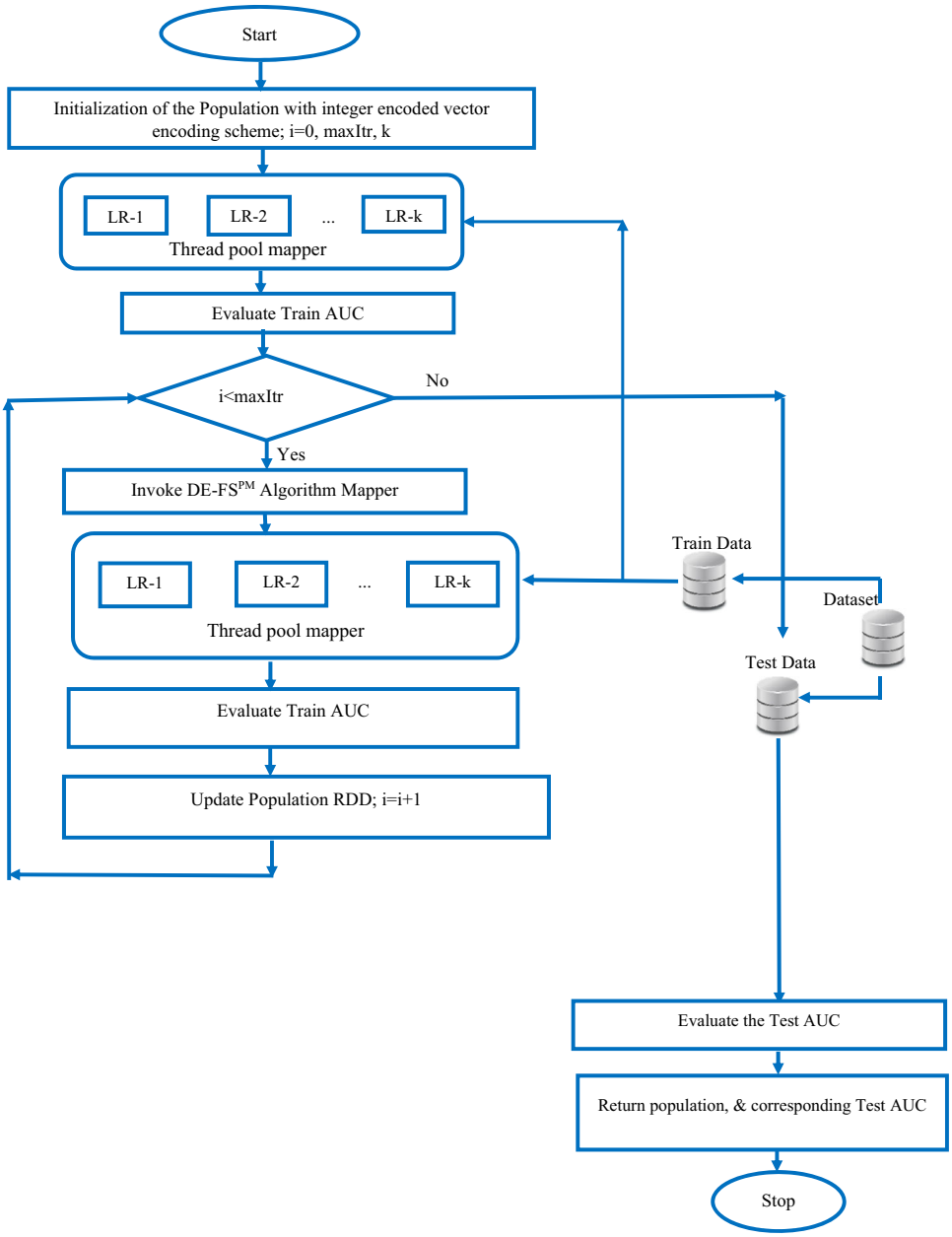

Fig. 9 Flowchart of P-DE-FS^{PM} based wrapper

Author contributions VY: Methodology, software, validation, formal analysis, investigation, data curation, writing-original draft, visualization; VR: conceptualization, methodology, validation, formal analysis, investigation, writing-original draft, writing-review and editing, visualization, supervision, project administration; PRK: writing-review and editing, resources, project administration.

Funding We did not receive any funding for this research.

Data availability The datasets used in this study are taken from the public domain and the appropriate URLs have been cited in the text.

Declarations

Conflict of interest The authors have not disclosed any competing interests.

Ethical approval This manuscript is not under review by any other Journal/Conference. However, for obvious reasons, its earlier version was submitted to arXiv preprint server. Hence, it has a similarity count of 75%.

Informed consent No human participants or animals are involved in this research.

References

1. CRISP DM. https://www.the-modeling-agency.com/crisp-dm.pdf. Accessed 24 Apr 2021
2. Chandrashekar, G., Sahin, F.: A survey on feature selection methods. Comput. Electr. Eng. **40**, 16–28 (2014)
3. Xue, B., Zhang, M., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. IEEE Trans. Evol. Comput. **20**, 606–626 (2016)
4. Wang, G., Deb, S., Cui, Z.: Monarch butterfly optimization. Neural Comput. Appl. **31**, 1995–2014 (2019)
5. Hu, J., Gui, W., Heidari, A.A., Cai, Z., Liang, G., Chen, H., Pan, Z. Dispersed foraging slime mould algorithm: Continuous and binary variants for global optimization and wrapper-based feature selection. Knowl. Syst. **237** (2022).
6. Strumberger, I., Bacanin, N.: Modified moth search algorithm for global optimization problems. Int. J. Comput. **3**, 44–48 (2018)
7. Yang, Y., Chen, H., Heidari, A.A., Gandomi, A.H.: Hunger games search: visions, conception, implementation, deep analysis, perspectives, and towards performance shifts. Expert Syst. Appl. **177**, 114864 (2021)
8. Tu, J., Chen, H., Wang, M., et al.: The colony predation algorithm. J. Bionic Eng. **18**, 674–710 (2021)
9. Heidari, A.A., Mirjalili, S., Faris, H., Aljarah, I., Mafarja, M., Chen, H.: Harris hawks optimization: algorithm and applications. Future Gener Comput Syst **97**, 849–872 (2019)
10. Nguyen, B.H., Xue, B., Zhang, M.: A survey on swarm intelligence approaches to feature selection in data mining. Swarm Evol. Comput. **54**, 100663 (2020)
11. Yang, W.A., Zhou, Q., Tsui, K.L.: Differential evolution-based feature selection and parameter optimisation for extreme learning machine in tool wear estimation. Int. J. Prod. Res. **54**, 4703–4721 (2016)
12. Xie, X., Xu, K., Wang, X.: Cloud computing resource scheduling based on improved differential evolution ant colony algorithm. In: ACM International Conference Proceeding Series, pp. 171–177 (2019).
13. Silva-Filho, A.G., Nunes, L.J.C., Lacerda, H.F.: Differential evolution to reduce energy consumption in three-level memory hierarchy. In: Proceedings of SBCCI 2015—28th Symposium on Integrated Circuits and Systems Design: CHIP in Bahia (2015).
14. Krishna, G.J., Ravi, V.: Anomaly detection using modified differential evolution: an application to banking and insurance. In: Proceedings of the 11th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2019). Advances in Intelligent Systems and Computing, p. 1182. Springer, Cham (2019).
15. Nissen V., Propach. J.: On the robustness of population-based versus point-based optimization in the presence of noise. In: IEEE Transactions on Evolutionary Computation, vol. 2, no. 3, pp. 107–119 (1998).
16. Roeva, O., Slavov, T., Fidanova, S.: Population-based vs. single point search meta-heuristics for a PID controller tuning. In: Handbook of Research on Novel Soft Computing Intelligent Algorithms: Theory and Practical Applications, pp. 200–233. IGI Global (2014).
17. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: OSDI'04 6th Symposium on Operating Systems Design and Implement, pp. 137–150 (2004).
18. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., Stoica, I.: Apache Spark: a unified engine for big data processing. Commun. ACM **59**(11), 56–65 (2016)
19. Das, S., Suganthan, P.N.: Differential evolution: a survey of the state-of-the-art. IEEE Trans. Evol. Comput. **15**(1), 4–31 (2011)
20. Chauhan, N., Ravi, V.: Differential evolution and threshold accepting hybrid algorithm for unconstrained optimization. Int. J. Bio-Inspired Comput. **2**, 169–182 (2010)
21. Krishna, G.J., Ravi, V.: Feature subset selection using adaptive differential evolution: an application to banking. In: ACM International Conference Proceeding Series, pp. 157–163 (2019).
22. Rivera-Lopez, R., Mezura-Montes, E., Canul-Reich, J., Cruz-Chávez, M.A.: A permutational-based differential evolution algorithm for feature subset selection. Pattern Recognit. Lett. **133**, 86–93 (2020)
23. Price, K., Storn, R.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. **11**, 341–359 (1997)
24. Zhang, Y., Gong, D.W., Gao, X.Z., Tian, T., Sun, X.Y.: Binary differential evolution with self-learning for multi-objective feature selection. Inf. Sci. (NY) **507**, 67–85 (2020)
25. Vivekanandan, T., Iyengar, N.C.S.N.: Optimal feature selection using a modified differential evolution algorithm and its effectiveness for prediction of heart disease. Comput. Biol. Med. **90**, 125–136 (2017)
26. Samuel, O.W., Asogbon, G.M., Sangaiah, A.K., Fang, P., Li, G.: An integrated decision support system based on ANN and Fuzzy_AHP for heart failure risk prediction. Expert Syst. Appl. **68**, 163–172 (2017)
27. Nayak, S.K., Rout, P.K., Jagadev, A.K., Swarnkar, T.: Elitism based multi-objective differential evolution for feature selection: a filter approach with an efficient redundancy measure. In: Journal of King Saud University—Computer and Information Sciences, vol. 32, pp. 174–187 (2020).
28. Mlakar, U., Fister, I., Brest, J., Potočnik, B.: Multi-objective differential evolution for feature selection in facial expression recognition systems. Expert Syst. Appl. **89**, 129–137 (2017)
29. Khushaba, R.N., Al-Ani, A., Al-Jumaily, A.: Feature subset selection using differential evolution and a statistical repair mechanism. Expert Syst. Appl. **38**, 11515–11526 (2011)
30. Hancer, E., Xue, B., Zhang, M.: Differential evolution for filter feature selection based on information theory and feature ranking. Knowl. Syst. **140**, 103–119 (2018)
31. Hancer, E.: A new multi-objective differential evolution approach for simultaneous clustering and feature selection. Eng. Appl. Artif. Intell. **87**, 103307 (2020)
32. Ghosh, A., Datta, A., Ghosh, S.: Self-adaptive differential evolution for feature selection in hyperspectral image data. Appl. Soft Comput. J. **13**, 1969–1977 (2013)
33. Bhadra, T., Bandyopadhyay, S.: Unsupervised feature selection using an improved version of differential evolution. Expert Syst. Appl. **42**, 4042–4053 (2015)
34. Baig, M.Z., Aslam, N., Shum, H.P.H., Zhang, L.: Differential evolution algorithm as a tool for optimal feature subset selection in motor imagery EEG. Expert Syst. Appl. **90**, 184–195 (2017)
35. Almasoudy, F.H., Al-Yaseen, W.L., Idrees, A.K.: Differential evolution wrapper feature selection for intrusion detection system. Procedia Comput. Sci. **167**, 1230–1239 (2020)
36. ZorarpacI, E., Ozel, S.A.: A hybrid approach of differential evolution and artificial bee colony for feature selection. Expert Syst. Appl. **62**, 91–103 (2016)
37. Srikrishna, V., Ghosh, R., Ravi, V., Deb, K.: Elitist quantum-inspired differential evolution based wrapper for feature subset selection. In: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9426, pp. 113–124 (2015).
38. Zhao, X.S., Bao, L.L., Ning, Q., Ji, J.C., Zhao, X.W.: An improved binary differential evolution algorithm for feature

selection in molecular signatures. Mol. Inform. **37**, 1–15 (2018). https://doi.org/10.1002/minf.201700081
39. Hancer, E.: Fuzzy kernel feature selection with multi-objective differential evolution algorithm. Conn. Sci. **3**, 323–341 (2019)
40. Li, J., Ding, L., Li, B.: Differential evolution-based parameters optimisation and feature selection for support vector machine. Int. J. Comput. Sci. Eng. **13**, 355–363 (2016)
41. Wang, J., Xue, B., Gao, X., Zhang, M: A differential evolution approach to feature selection and instance selection. In: Proceedings of the 14th Pacific RIM International Conference on Trends in Artificial Intelligence (PRICAI'16). Gewerbestrassse 11 CH-6330, Cham (ZG), CHE, pp. 588–602. Springer (2016).
42. Carrasco, J., García, S., Rueda, M.M., Das, S., Herrera, F.: Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: practical guidelines and a critical review. Swarm Evol. Comput. **54**, 100665 (2020)
43. Cao, B., Fan, S., Zhao, J., Yang, P., Muhammad, K., Tanveer, M.: Quantum-enhanced multiobjective large-scale optimization via parallelism. Swarm Evol. Comput. **57**, 100697 (2020)
44. BenSaid, F., Alimi, A.M.: Moanofs: multi-objective automated negotiation based online feature selection system for big data classification (2018). arXiv:1810.04903.
45. Khan, A., Baig, A.R.: Multi-objective feature subset selection using non-dominated sorting genetic algorithm. J. Appl. Res. Technol. **13**, 145–159 (2015)
46. Khammassi, C., Krichen, S.: A NSGA2-LR wrapper approach for feature selection in network intrusion detection. Comput. Netw. **172**, 107183 (2020)
47. Chaudhuri, A., Sahu, T.P.: Feature selection using Binary Crow search Algorithm with time varying flight length. Expert Syst. Appl. **168**, 114288 (2021)
48. Too, J., Mirjalili, S.: A hyper learning binary dragonfly algorithm for feature selection: a COVID-19 case study. Knowl. Syst. **212** (2021).
49. Hu, J., Chen, H., Heidari, A.A., Wang, M., Zhang, X., Chen, Y., Pan, Z.: Orthogonal learning covariance matrix for defects of grey wolf optimizer: insights, balance, diversity, and feature selection. Knowl. Syst. **213** (2021).
50. Hu, J., Heidari, A.A., Zhang, L., Xue, X., Gui, W., Chen, H., Pan, Z.: Chaotic diffusion-limited aggregation enhanced grey wolf optimizer: insights, analysis, binarization, and feature selection. Int. J. Intell. Syst. **37**(8), 4864–4927 (2021)
51. Too, J., Liang, G., Chen, H.: Memory-based Harris hawk optimization with learning agents: a feature selection approach. Eng. Comput. 1–22 (2021).
52. Zhang, Y., Liu, R., Wang, X., Chen, H., Li, C.: Boosted binary Harris Hawks optimizer and feature selection. Eng. Comput. **37**, 3741–3770 (2021)
53. Hammami, M., Bechikh, S., Hung, C.C., Ben Said, L.: A multi-objective hybrid filter-wrapper evolutionary approach for feature selection. Memetic Comput. **11**, 193–208 (2019)
54. Harada, T., Kaidan, M., Thawonmas, R.: Comparison of synchronous and asynchronous parallelization of extreme surrogate-assisted multi-objective evolutionary algorithm. Nat. Comput. (2020).
55. Peralta, D., Del Río, S., Ramírez-Gallego, S., Triguero, I., Benitez, J.M., Herrera, F.: Evolutionary feature selection for big data classification: a MapReduce approach. Math. Probl. Eng. (2015).
56. Rong, M., Gong, D., Gao, X.: Feature selection and its use in big data: challenges. Methods Trends IEEE Access **7**, 19709–19725 (2019)
57. Zhou, C.: Fast parallelization of differential evolution algorithm Using MapReduce. In: Proceedings of 12th Annual Genetic and Evolutionary Computation Conference (GECCO '10), pp. 1113–1114 (2010).
58. Teijeiro, D., Pardo, X.C., González, P., Banga, J.R., Doallo, R.: Implementing parallel differential evolution on spark. In: Squillero, G., Burelli, P. (eds.) Applications of Evolutionary Computation (EvoApplications 2016). Lecture Notes in Computer Science, p. 9598. Springer, Cham (2016).
59. Cho, P.P.W., Nyunt, T.T.S., Aung, T.T.: Differential evolution for large-scale clustering. In: Proceedings of 2019 9th International Workshop on Computer Science and Engineering (WCSE 2019 SPRING), pp. 58–62 (2019).
60. Al-Sawwa, J., Ludwig, S.A.: Performance evaluation of a cost-sensitive differential evolution classifier using spark—imbalanced binary classification. J. Comput. Sci. **40**, 101065 (2020). https://doi.org/10.1016/j.jocs.2019.101065
61. Chen, Z., Jiang, X., Li, J., Li, S., Wang, L.: PDECO: parallel differential evolution for clusters optimization. J. Comput. Chem. **34**, 1046–1059 (2013)
62. Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J., Tallent, N.R.: HPCTOOLKIT: tools for performance analysis of optimized parallel programs. Concurr. Comput. Pract. Exp. **22**, 685–701 (2010)
63. Deng, C., Tan, X., Dong, X., Tan, Y.: A parallel version of differential evolution based on resilient distributed datasets model. Commun. Comput. Inf. Sci. **562**, 84–93 (2015)
64. He, Z., Peng, H., Chen, J., Deng, C., Wu, Z.: A Spark-based differential evolution with grouping topology model for large-scale global optimization. Clust. Comput. **24**, 515–535 (2021)
65. Wong, T.H., Qin, A.K., Wang, S., Shi, Y.: cuSaDE: a CUDA-based parallel self-adaptive differential evolution algorithm. In: Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, vol. 2, pp. 375–388 (2015).
66. Cao, B., Zhao, J., Lv, Z., Liu, X.: A distributed parallel cooperative coevolutionary multiobjective evolutionary algorithm for large-scale optimization. IEEE Trans. Ind. Inf. **13**, 2030–2038 (2017)
67. Ge, Y., Yu, W., Lin, Y., Gong, Y., Zhan, Z., Chen, W., Zhang, J.: Distributed differential evolution based on adaptive mergence and split for large-scale optimization. IEEE Trans. Cybern. **48**, 2166–2180 (2018)
68. De Falco, I., Scafuri, U., Tarantino, E., Della Cioppa, A.: A distributed differential evolution approach for mapping in a grid environment. In: 15th EUROMICRO international conference on parallel, distributed and network-based processing (PDP'07), pp. 442–449 (2007). https://doi.org/10.1109/PDP.2007.6.
69. Veronese, L.P., Krohling, R.A.: Differential evolution algorithm on the GPU with C-CUDA. In: IEEE Congress on Evolutionary Computation, pp. 1–7 (2010). https://doi.org/10.1109/CEC.2010.5586219.
70. Glotic, A., Glotic, A., Kitak, P., Pihler, J., Ticar, I.: Parallel self-adaptive differential evolution algorithm for solving short-term hydro scheduling problem. IEEE Trans. Power Syst. **29**, 2347–2358 (2014)
71. Daoudi, M., Hamena, S., Benmounah, Z., Batouche, M.: Parallel diffrential evolution clustering algorithm based on MapReduce. In: 2014 6th International Conference of Soft Computing and Pattern Recognition (SoCPaR), pp. 337–341 (2014).
72. Krömer, P., Platoš, J., Snášel, V.: Scalable differential evolution for many-core and clusters in unified parallel C. In: 2013 IEEE International Conference on Cybernetics (CYBCO), pp. 180–185 (2013).
73. Thomert, D.B., Bhattacharya, A. K., Caron, E., Gadireddy, K., Lefevre, L.: Parallel differential evolution approach for cloud workflow placements under simultaneous optimization of multiple objectives. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 822–829 (2016).
74. Abbass, H.A., Sarker, R.: The Pareto differential evolution algorithm. Int. J. Artif. Intell. Tools **11**(4), 531–552 (2002)

75. Ali, M.M., Tom, A.: Population set based global optimization algorithms: some modifications and numerical studies. Comput. Oper. Res. **31**(10), 1703–1725 (2004)
76. Kohavi, R., John, G.H.: Wrappers for feature subset selection. In: Lecture Notes Computer Science (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 97, pp. 273–324 (1997).
77. Dueck, G., Scheuer, T.: Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. J. Comput. Phys. **90**, 161–175 (1990)
78. Ravi, V., Zimmermann, H.J.: Fuzzy rule based classification with FeatureSelector and modified threshold accepting. Eur. J. Oper. Res. **123**, 16–28 (2000)
79. Ravi, V., Reddy, P.J., Zimmermann, H.J.: Fuzzy rule base generation for classification and its minimization via modified threshold accepting. Fuzzy Sets Syst. **120**, 271–279 (2001)
80. Ravi, V., Zimmermann, H.-J.: A neural network and fuzzy rule base hybrid for pattern classification. Soft Comput. **5**, 152–159 (2001)
81. Ravi, V., Pramodh, C.: Threshold accepting trained principal component neural network and feature subset selection: application to bankruptcy prediction in banks. Appl. Soft Comput. J. **8**, 1539–1548 (2008)
82. Tvrdík, J.: Adaptation in differential evolution: a numerical comparison. Appl. Soft Comput. **9**(3), 1149–1155 (2009)
83. Zielinski, K., Peters, D., Laur, R.: Run time analysis regarding stopping criteria for differential evolution and particle swarm optimization. In: Proceedings of 1st International Conference on Experiments/Process/System Modelling/Simulation/Optimization (2005).
84. Kaggle Open source Datasets. https://www.kaggle.com/c/microsoft-malware-prediction/data. Accessed 27 Mar 2021
85. IEEE Dataport. https://ieee-dataport.org/. Accessed 27 Mar 2021
86. OpenML Open Source Datasets. https://www.openml.org/home. Accessed 27 Mar 2021
87. LIBSVM repository for the binary class high dimensional datasets. https://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/. Accessed 27 Mar 2021
88. Apache Spark. https://spark.apache.org/. Accessed 26 Jan 2021
89. Peralta, D., Río, S.D., Gallego, S.R., Triguero, I., Benitez, J.M., Herrera, F.: Evolutionary feature selection for big data classification: a mapreduce approach. Math. Probl. Eng. (2015)
90. Pes, B.: Learning from high-dimensional biomedical datasets: the issue of class imbalance. IEEE Access **8**, 13527–13540 (2020). https://doi.org/10.1109/ACCESS.2020.2966296
91. Hooten, S., Vadlamani, S.K., Beausoleil, R.G., Vaerenbergh, T.V.: Generative neural network based non-convex optimization using policy gradients with an application to electromagnetic design. In: NeurIPS 2021 AI for Science Workshop (2021).
92. Al-Ani, A., Alsukker, A., Khushaba, R.N.: Feature subset selection using differential evolution and a wheel based search strategy. Swarm Evol. Comput. **9**, 15–26 (2013)
93. Liu, X.F., Zhan, Z.H., Lin, J.H., Zhang, J.: Parallel differential evolution based on distributed cloud computing resources for power electronic circuit optimization. In: GECCO 2016 Companion—Proceedings of 2016 Genetic and Evolutionary Computation Conference, pp. 117–118 (2016).

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Yelleti Vivek graduated with an M. Tech. in Computer Science from University of Hyderabad in 2020. He is currently a doctoral student in National Institute of Technology, Warangal and IDRBT. His research areas include big data analytics, machine learning, evolutionary computing and reinforcement learning.

Vadlamani Ravi is with the Institute for Development and Research in Banking Technology (IDRBT), Hyderabad since 2005. He holds a PhD in Soft Computing from Osmania University, Hyderabad and RWTH Aachen, Germany (2001), MS from BITS Pilani (1991) and M.Sc (Statistics &OR) from IIT Bombay (1987). Earlier, he worked as a Faculty at National University of Singapore from 2002-2005. He worked in RWTH Aachen under DAAD Long Term Fellowship from 1997–1999 for conducting advanced research. During 1988-2002, he worked in two CSIR research labs as Scientist B, Scientist C and Assistant Director. During the last 34 years, he has been working in soft computing, evolutionary/neuro/fuzzy computing, data/text mining, global/multicriteria optimization, big data analytics, social media analytics, time series data mining, machine learning/deep learning, bankruptcy prediction and analytical CRM. He has 22 years of teaching experience and 34 years of research experience. He published more than 250 papers in refereed international/national journals/conferences and invited book chapters. He has 9700 citations and an h-index of 45. He also edited three Books published by IGI Global, USA, 2007 and IRT, UK, in 2021. He appears list of the top 1.03% scientists published by Stanford University researchers in PLOS Biology Journal. He won Best research Award in CSIR-IICT, and IDRBT. He is a referee for 40 International Journals. He is an Associate Editor for Swarm and Evolutionary Computation and Editorial Board Member for few International Journals of repute. He was the founding Managing Editor for Journal of Banking and Financial Technology and He is an Advisor for various Indian banks including BoI, Andhra Bank, Canara Bank, NABARD, RBI, IRDAI and SEBI for their DWH, Analytical CRM, Analytics and Fraud Analytics projects. He developed roadmap for implementing Data Science (AI/ML subsumed)) related projects for several banks. He evaluates the project proposals submitted to European Science Foundation, Dutch Science Foundation, Belgium Science Foundation, Irish Science Foundation and book proposals submitted to Elsevier and Springer. He is a referee, PC Member and Chair for any International conferences of repute.

P. Radha Krishna has been in the profession of research, development and technology adoption for about Thirty years. He is currently working as a Professor and Head, Department of Computer Science and Engineering, National Institute of Technology (NIT) Warangal. His research interests include data mining, big data, machine learning and databases and workflow systems. Prior to joining NIT, he served as Principal Research Scientist at Infosys Labs, Infosys Limited, Hyderabad, where he was associated with research projects leading to futuristic intelligent systems and analytical solutions. Krishna conceptualized numerous research projects in the areas of CRM, social network mining, Web, Sequence, text & image mining, e-check clearing and settlement, and financial inclusion, and associated in building innovation labs for leading industries. He also served as adjunct faculty at NIT-Warangal and IIIT-Hyderabad. Dr. Krishna also served as a faculty at the Institute for Development and Research in Banking Technology (IDRBT - a research arm of the Reserve Bank of India & an associate Institute of University of Hyderabad), and as a scientist at National Informatics Centre (Govt. of India), Bhopal. At IDRBT, he was an Advisor to many Public and Private Sector Banks in India for the implementation of End-to-End Enterprise-wide Data Warehouse and Business Intelligence solutions. Krishna was also a member of the IT Advisory Committee, Insurance Regulatory and Development Authority (IRDA), India. He has been a member of the research advisory committees of several academic institutes. Also, Dr. Krishna has been a member of the editorial/review board of international journals; and a reviewer of several international journals including IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Services Computing, Information Sciences, Data and Knowledge Engineering, Information System Frontiers and Soft Computing. Krishna has double PhDs – the first one from Osmania University and the second one from IIIT-Hyderabad; 19 granted patents, authored/co-authored six books, and have over a hundred publications in refereed journals and conferences. He is also a tutorial speaker at several international conferences including ER 2006, ICWS 2007, WWW 2008, IEEE Services 2010, CAiSE 2010, ICSE 2014 and ER 2014.