Journal of King Saud University – Computer and Information Sciences xxx (xxxx) xxx

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Salp swarm optimizer for modeling the software fault prediction problem

Sofian Kassaymeh [a,*], Salwani Abdullah Ph.D [a], Mohammed Azmi Al-Betar [b,c], Mohammed Alweshah [d]

[a] Data Mining and Optimization Research Group, Center for Artificial Intelligence Technology, Universiti Kebangsaan Malaysia, Bangi Selangor, Malaysia
[b] Artificial Intelligence Research Center (AIRC), College of Engineering and Information Technology, Ajman University, Ajman, United Arab Emirates
[c] Department of Information Technology, Al-Huson University College, Al-Balqa Applied University, Irbid, Jordan
[d] Department of Computer Science, Prince Abdullah Bin Ghazi Faculty of Communication and Information Technology, Al-Balqa Applied University, Salt, Jordan

ARTICLE INFO

Article history:
Received 2 October 2020
Revised 24 January 2021
Accepted 25 January 2021
Available online xxxx

Keywords:
Salp swarm optimizer
Backpropagation neural network
Software reliability
Software fault estimation

ABSTRACT

This paper proposes the salp swarm algorithm (SSA) combined with a backpropagation neural network (BPNN) to solve the software fault prediction (SFP) problem. The SFP problem is one of the well-known software engineering problems that are concerned with anticipating the software defects that are likely to appear during a software project or thereafter. In order to find the optimal BPNN parameters, a combination of SSA optimizer and BPNN named (SSA-BPNN) is proposed, so as to enhance prediction accuracy. The proposed method is evaluated against several datasets for the SFP problem. These datasets vary in both size and complexity. The results obtained are evaluated using a variety of performance measures (i.e., the AUC, Confusion Matrix, Sensitivity, Specificity, Accuracy, and Error Rate). The results obtained by SSA-BPNN are better than those obtained by the conventional BPNN over all of the datasets. The proposed method also has the ability to outperform several state-of-the-art methods over the same datasets in respect of most of the aforementioned performance measures. Therefore, the hybridization of SSA with BPNN is a valuable addition to the software engineering issues and can be utilized to achieve higher prediction accuracy for a variety of prediction problems.

1. Introduction

Prediction is a supervised learning method for discovering patterns, knowledge, or relationships based on previous data. Over the years, numerous prediction methods have been built based on various learning models, such as Naïve Bayes, decision trees, support vector machines (SVMs) (Rathore and Kumar, 2017; Malhotra, 2015; Bowes et al., 2018; Menzies et al., 2006; Khoshgoftaar and Seliya, 2003), Case-based Reasoning (CR) (Alshareef et al., 2015), Bayesian Network (BN) (Carrozza et al., 2013), Fuzzy Clustering (FC) (Yuan et al., 2000), Artificial Neural Networks (ANNs) (Thwin et al., 2005), Multi-layer Perception (MLP) (Carrozza et al., 2013), and Logistic Regression (LR) (Yuan et al., 2000).

The ANN is a type of ML algorithm where the computer system learns to accomplish a plethora of tasks through analyzing training examples. In the context of software prediction problems, the BPNN is the most popular form of NN (Rashid et al., 2020), because it involves the use of a reliable training method that enables the BPNN to handle both simple and complex problems (Abdullah et al., 1361). Indeed, backpropagation is one of the most popular learning methods due to its accuracy and self-organization capabilities for prediction problems (Chaudhary et al., 2015; Kuldip Vora, 2014; Ever et al., 2019). Hence, backpropagation is still one of the algorithms most widely applied to prediction problems in comparison with other ML techniques (Sekeroglu et al., 2019; Ogidan et al., 2018). However, traditional backpropagation training algorithms have some inherent drawbacks, such as a slow convergence rate (Abusnaina et al., 2018), a propensity to easily fall into local minima (Abusnaina et al., 2018), and a need for too many parameter settings (Khazaiepoor et al., 2020). Therefore, the identification of an optimization method that can be used in combination with the BPNN to overcome these issues would be a valuable contribution to the field of ML.

* Corresponding author.
E-mail addresses: samsaak@gmail.com (S. Kassaymeh), salwani@ukm.edu.my (S. Abdullah), m.albetar@ajman.ac.ae, mohbetar@bau.edu.jo (M.A. Al-Betar), weshah@bau.edu.jo (M. Alweshah).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2021.01.015
1319-1578/

Over the years, various metaheuristic algorithms have been hybridized with different kinds of prediction models to obtain an optimized weight that can be fed to the prediction model in an attempt to achieve better prediction accuracy. These endeavors have resulted in the development of hybrid methods that have attained better performance when compared to standard prediction approaches (Bui, 2019; Ojha et al., 2017). Metaheuristic algorithms have been used in hybrid forms with several swarm intelligence optimization algorithms such as artificial bee colony optimization (Karaboga and Ozturk, 2009), the particle swarm optimization (Rakitianskaia and Engelbrecht, 2012; Alsaeedi et al., 2020), the cuckoo search algorithm (Nawi et al., 2013), firefly algorithm (Mandal et al., 2015), genetic algorithm (GA) (Baklacioglu et al., 2020; Baklacioglu et al., 2018; Tian et al., 2018), biogeography-based optimization (BBO), gravitational search algorithm (GSA) and grey wolf optimization (GWO) (Bui, 2019), and the bat-inspired algorithm (Jaddi et al., 2015). These optimization methods enhance the BPNN algorithm in terms of the accuracy achieved during the training process, and thus overcome the shortcoming of the conventional gradient-based procedure in respect of adjusting the required parameters (e.g., weights and biases) because the BPNN often gets trapped in local optima (Mavrovouniotis and Yang, 2015).

One of the more attractive metaheuristic algorithms is the salp swarm algorithm (SSA) (Mirjalili et al., 2017). It is a type of swarm intelligence algorithm inspired by the behavior of a marine invertebrate, the sea salp. SSA has been tested on different benchmark optimization and ML problems, and its efficiency has been proven concerning a wide range of optimization problems, such as multi-objective problems (Aljarah et al., 2020), image thresholding (Elaziz et al., 2020), feature selection (Faris et al., 2020; Aljarah et al., 2020; Aljarah et al., 2018) constrained engineering optimization problems (Zhang et al., xxxx), load frequency control (Barik and Das, 2018; Guha et al., 2018; Ekinci and Hekimoglu, 2018), estimation and forecasting (Zhang et al., 2018; El-Fergany, 2018; Hussien et al., 2017; Wang et al., 2018), detection model (Abbassi et al., 2019), and integrated circuit design (Asaithambi and Rajappa, 2018). On the other hand, the SSA success over other swarm intelligence algorithms is due to several advantages. Firstly, SSA has very few parameters to be tune (El-Fergany et al., 2019; Abualigah et al., 2019). Also, SSA appropriate for a wide range of optimization problems and can cover a wide search space (Abualigah et al., 2019). Finally, it has a strong neighborhood search ability (Abualigah et al., 2019). In spite of the above, the SSA algorithm still suffers from the disadvantage of low exploitation, which may lead to slow convergence (Ibrahim et al., 2018; Abusnaina et al., 2018; Abualigah et al., 2019).

Therefore, drawing on the relative advantages of the SSA and the BPNN, this paper proposes a new general framework that, for the first time, combines the SSA metaheuristic approach with a BPNN to examine how the combination of optimization techniques along with machine learning methods enhances the software fault prediction techniques. In addition to investigate how the manipulating of the BPNN parameters could lead to introduce any performance improvement in terms of prediction accuracy. The proposed method is named SSA-BPNN. The initial population of the proposed method is derived from the BPNN, where the SSA is utilized to optimize the BPNN parameters iteratively. Therefore, the BPNN takes the best solution found by the SSA to improve prediction accuracy. For performance evaluation, this work uses SFP problem datasets taken from several resources. These datasets vary in terms of both their size and complexity. The results of the proposed method are compared against conventional BPNN and several methods in the literature. These comparisons show that SSA-BPNN can provide the best solution for most SFP problem datasets.

The remainder of this paper is organized as follows: First, an introduction to the SFP problem and its objective function and solution representation is provided in Section 2. Next, the research background for Backpropagation Neural Network and Salp Swarm Optimizer is reviewed in Section 3. Then, the proposed SSA optimizer for the BPNN predictor is described in Section 4. After that, the experiments and results including parameters configuration, performance measure, employed datasets in addition to comparisons and analysis are presented and discussed in Section 5. Finally, the conclusion, together with some suggestions for possible directions for future enhancements, is provided in Section 6.

2. Software fault prediction problem

In this section, one of the most critical software reliability problems, the SFP problem is defined. The engineering of software quality involves a variety of activities that are performed for the purpose of quality assurance. These activities include verification, validation, testing, fault tolerance identification, and, importantly, SFP. Software fault prediction can be defined as the process of estimating the errors (sometimes called defects) in a software product under and after development, based on previously defined metrics, or based on historical defect data elicited from prior similar projects (Catal, 2011; Porter and Selby, 1990). The ability to estimate software faults early on during the development process or even before starting a project can be indispensable in minimizing software development time and effort, where accurate SFP can reduce the efforts needed to detect software errors throughout the software life cycle and minimize the number of modules developed in each activity (Turabieh et al., 2019).

The significance of the SFP problem can be summarized by the following observation: Developing and producing a software product that contains errors (of whatever kind and however many) will adversely affect the quality of the ensuing releases of said software because there is an interrelation among software versions that are developed sequentially (Turabieh et al., 2019). However, the performance of the SFP models developed to address this problem is affected by the modeling techniques (Qasem et al., 2020; Abaei et al., 2015a; Abaei et al., 2015b) and metrics (Aziz et al., 2020) that are used. Modeling techniques have moderation of performance difference, and there is a little classification accuracy impact of modeling technique than the used metrics (Qasem et al., 2020).

In general, when designing an optimization algorithm two key issues ought to be considered: the objective function and the solution representation (Turabieh et al., 2019).

2.1. Objective function for software fault prediction

The main goal of developing a SFP model is to estimate the number of expected faults in a forthcoming software project or in a project that is under development, whereas obtaining a high accuracy model. The proposed method SSA-BPNN uses the ANN as an evaluator. In addition, the objective function employed in this research is the area under the ROC curve (AUC), which will be discussed in subsection 5.2.

2.2. Solution representation

The SFP model solution is represented as a vector of length equal to the number of ANN weights and biases, as will be shown later in Fig. 4, which is extracted from the BPNN after the initial training process accomplished, then passed to the SSA, as discussed in Section 4-Step-4. Next, the training part of the dataset is employed to calculate the gradient and update the network

weights and biases in the BPNN training process. The testing part of the dataset is employed to evaluate the gained model through the BPNN testing process.

3. Research background

The BPNN is an ML algorithm that can solve prediction-related problems, such as the SFP problem. The effectiveness of the BPNN is attributable to its parameters, where optimized parameter values enable the BPNN to achieve high prediction accuracy. The basic principles of the BPNN and the BP concept are introduced in subsection 3.1. Then, the fundamentals of the traditional SSA algorithm are described in subsection 3.2.

3.1. Backpropagation neural network

The BPNN is a type of ANN. It consists of a multi-layer feedforward NN based on the error backpropagation learning algorithm, and there are two parts to the working mechanism: the data forward transmission part and the error feedback propagation part (Wang et al., 2019). Backpropagation is a technique that is used to train the feedforward NN. The main target of BP is to optimize the parameters (i.e., weights w, and biases b) in the network as a whole, in order to enable the BPNN network outcome to get as close as possible to the desired result. In other words, the BP technique gives the ANN the ability to adjust the parameter values based on back-propagated output errors from network outputs, which ultimately leads to minimizing the output error.

The network of the BPNN is composed of three types of layer: the input layer X, hidden layer H, and output layer Y, and each of these layers contains a particular number of neurons (sometimes called nodes). Each node has an activation function; this activation function is unique at each layer for all neurons. Besides, the nodes are connected between layers, where each connection has an appointed weight w and bias b. Fig. 1 shows the basic structure of a BPNN network.

The first part of the BP technique process (*data forward transmission*) involves receiving the input data by the input layer which then goes into the output layer via the hidden layer after some specific calculations have been applied to the data. The second part (*error feedback propagation*) of the BP technique process starts by calculating the variance between the actual output and the desired output which may not close to or far away from each other. This variance is called the "prediction error". Thereafter, the prediction error is backpropagated to the input layer as "error feedback propagation". Through several iterations of this process, the prediction error is utilized to adjust the BPNN parameters until the prediction error is close to zero (Li et al., 2019; Hecht-Nielsen, 1992). Consequently, the objective of the BP technique is to minimize the output error and produce a network with high accuracy (Felipe et al., 2014). In the conventional BPNN, prediction accuracy is closely related to the network parameters, so if the network parameters are not optimal, the output accuracy will be poor until the parameters are adjusted appropriately.

The abbreviations w_{ij} and w_{jk} represent the connection weights, the terms $(x_1, x_2, \ldots, x_z)$ represent the input values, and the terms $(y_1, y_2, \ldots, y_m)$ represent the estimated values. The BP technique can be described by the following steps (Ren et al., 2014; Sun and Huang, 2020):

Step 1: Initialize network structure: First, the main structure of the network, or more precisely the number of hidden layers H, is determined. There are no particular rules for determining the best number of hidden layers, as well as the best number of nodes in the hidden layer, but the number of hidden layers

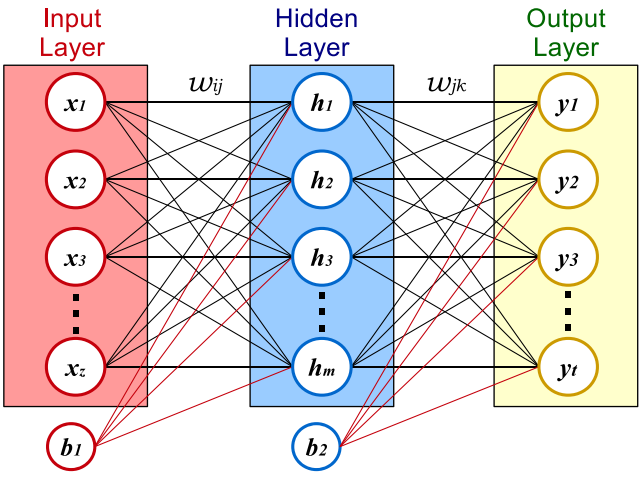

Fig. 1. BPNN basic structure.

and number of nodes in those layers does have a significant impact on network performance and accuracy (Han et al., 2011). Mostly, when designing an ANN the trial-and-error method is utilized (Han et al., 2011). However, by default, there is one input layer X and one output layer Y. The number of nodes in the input and output layers depends on the prediction requirements.

Step 2: Initialize network parameters: The weights w_{ij} and w_{jk} and biases b_1 and b_2 are initialized randomly. The learning rate η is also initialized in this step. The vector of weights w_{ij} is a randomly set of connection weights for nodes i in the input layer and nodes j in the hidden layer. Also, the vector of weights w_{jk} is a randomly set of connection weights for nodes j in the hidden layer and nodes k in the output layer.

Step 3: Calculate the hidden layer output: The values of X are acquired and initialized to calculate w_{ij} and b_1, so the hidden layer H output can be calculated by using Eq. (1):

$$H_j = f\left(\sum_{i=1}^{z} w_{ij} X_i + b_1\right), \tag{1}$$

where $\forall i = (1, 2, \ldots, z)$, z is the number of nodes in the input layer X, $\forall j = (1, 2, \ldots, m)$ and m is the number of nodes in the hidden layer H.

Step 4: Calculate the output layer output: By finding the value of H from the previous step, and initializing/calculating the values of w_{jk} and b_2, the output layer Y output can be calculated by using Eq. (2):

$$Y_k = \sum_{j=1}^{m} w_{jk} H_j + b_2 \tag{2}$$

where $j = (1, 2, \ldots, m), m$ is the number of nodes in the hidden layer H, $k = (1, 2, \ldots, t)$ and t is the number of nodes in the output layer Y.

Step 5: Calculate the prediction error: The model prediction error e is calculated by determining the variance between the output layer Y outcome and the desired output D by using Eq. (3):

$$e_k = Y_k - D_k \tag{3}$$

where $\forall k = (1, 2, \ldots, t)$ and t is the number of nodes in the output layer Y.

Step 6: Adjust the weights: The adjustment of weights w_{ij} and w_{jk} is based on the prediction error e by using Eq. (4) and Eq. (5):

$$w_{ij} = w_{ij} + \eta H_i(1 - H_j)X(i)\sum_{k=1}^{t} w_{jk}e_k \quad (4)$$

$$w_{jk} = w_{jk} + \eta H_j e_k \quad (5)$$

where $\forall i = (1,2,\ldots,z)$, $\forall j = (1,2,\ldots,m)$, $\forall k = (1,2,\ldots,t)$, and η is the learning rate.

Step 7: Adjust the biases: The adjustment of biases b_1 and b_2 is based on prediction error e by using Eq. (6):

$$b_j = b_j + \eta H_j(1 - H_j)\sum_{k=1}^{t} w_{jk}e_k \quad (6)$$

$$b_k = b_k + e_k \quad (7)$$

where $\forall j = (1,2,\ldots,m)$ and m is the number of nodes in the hidden layer H, $\forall k = (1,2,\ldots,t)$ and t is the number of nodes in the output layer Y, and η is the learning rate.

Step 8: End rules: If the convergence of the model is not satisfied, go to Step 3. Iterate the above-mentioned steps until the end rules are fulfilled.

3.2. Salp swarm optimizer

The SSA was recently proposed by Mirjalili et al. (2017). It is a type of swarm intelligence algorithm that draws its inspiration from a member of the Salpidae family of marine invertebrates. The salp is shaped like a barrel and is transparent. It seems generally similar to a jellyfish in that it moves by pumping water inside its body to give it forward propulsion (Madin, 1990). The shape of a salp is presented in Fig. 2(a) (Mirjalili et al., 2017). The salp exhibits a swarming behavior in deep water to form a "salp chain", which is illustrated in Fig. 2(b) (Mirjalili et al., 2017).

A salp chain can be modeled mathematically in terms of optimization rules, where the salp population is separated into two sets: a leader and a group of followers. At the front of the chain is the leader, and behind the leader is the rest of the chain consisting of the followers.

As salp exhibits a swarming behavior, swarm-based techniques can be applied to it. Hence, the position of the salps can be characterized in an n-dimensional search area. Note that here n is a problem variable number, and all the salps' positions can be formulated as a two-dimensional array symbolized by X. In addition, the search area includes a food source, which is symbolized by F and is the main chain target. Eq. (8) represents the renewal of the leader's position:

$$x_j^1 = \begin{cases} F_j + p_1 * ((ub_j - lb_j) * p_2 + lb_j) & p_3 \geqslant 0.5 \\ F_j - p_1 * ((ub_j - lb_j) * p_2 + lb_j) & p_3 < 0.5 \end{cases} \quad (8)$$

where x_j^1 represents the leader position in the j^{th} dimension, F_j presents the food position in the j^{th} dimension, ub_j, and lb_j are the upper and lower bounds in the j^{th} dimension, respectively, and p_1, p_2, and p_3 are random factors.

Also, from Eq. (8), it can be noted that the leader can change its position only in relation to F. The random factor p_1 can be considered as the most significant parameter in the SSA algorithm because it has the ability to balance exploration and exploitation. This factor can be computed by using Eq. (9):

$$p_1 = 2e^{-\left(\frac{4l}{L}\right)^2} \quad (9)$$

where L is the total number of repetitions (iterations) and l is the present repetition (current iteration).

In addition, the values of the other two random parameter p_2 and p_3 have to be in the [0, 1] range. The two main purposes of these two parameters are to guide the j^{th} dimension toward a positive or negative direction and to set the size of the salp movement step. On the other hand, the followers' positions can be updated by using Eq. (10):

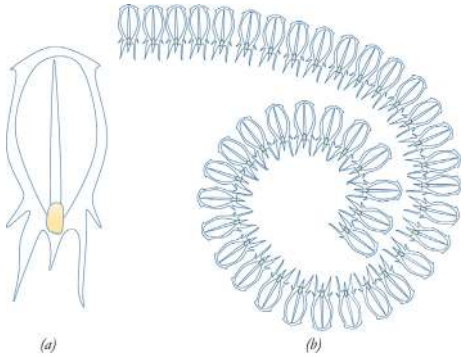

Fig. 2. (a) Single Salp, (b) Salps chain (salps swarm).

$$x_j^{i+1} = \frac{1}{2}\left(x_j^i + x_j^{i-1}\right) \quad (10)$$

where $i \geqslant 2$ and x_j^i shows the position of i^{th} follower salp in j^{th} dimension

The SSA algorithm mechanism begins by randomly initializing a group of solutions and these solutions form the SSA initial population. Then, iteration by iteration, the improvement process begins in order to define the global optimum. Based on the leader position, the follower salps update their positions via this improvement process. This process is repeated until the desired solution is reached. This process can be summarized by the following steps:

Step 1: Generate the SSA population and initialize the parameters: the SSA algorithm optimization process starts by initializing a group of random solutions, and these solutions form the initial population of the SSA and are denoted by size n. The solutions ordinarily are proposed as a vector $\mathbf{x}^i = (x_1^i, x_2^i, \ldots, x_j^i, \ldots, x_d^i)$, where $i = (1,2,\ldots,n), n$ is the total number of decision variables (population size) within the range $[lb_i, ub_i], j = (1,2,\ldots,d)$ and d is the d-dimensional search space for the i^{th} individual. In addition, the SSA parameters (p_1, p_2, and p_3) are adaptively initialized and refreshed through the optimization process.

Step 2: Determine the best solution: the food source F position is considered the best solution, and it is proposed by the d-dimensional vector $F = (F_1, F_2, \ldots, F_j, \ldots, F_d)$, where $j = (1,2,\ldots,d)$. The best solution position is defined through choosing the salp with the best fitness, after computing the fitness values of all the salps.

Step 3: Perform the iterative improvement loop: the maximum size of the iterative improvement loop is called L. The improvement processes for the salps are performed as follows:

Step 3.1: Update the leader position: the position of the salp leader (best solution) can be updated by using Eq. (8).

Step 3.2: Update the followers' positions: the positions of the followers can be updated by using Eq. (10). Where ub_j and lb_j symbolize the upper bound and lower bound of the search space, respectively.

Step 3.3: Update the best solution position: in this sub-step, the fitness value of every salp is computed in order to compare these values with the global optimum fitness value. Any salp with a fitness that is better than the global optimum fitness becomes the global optimum.

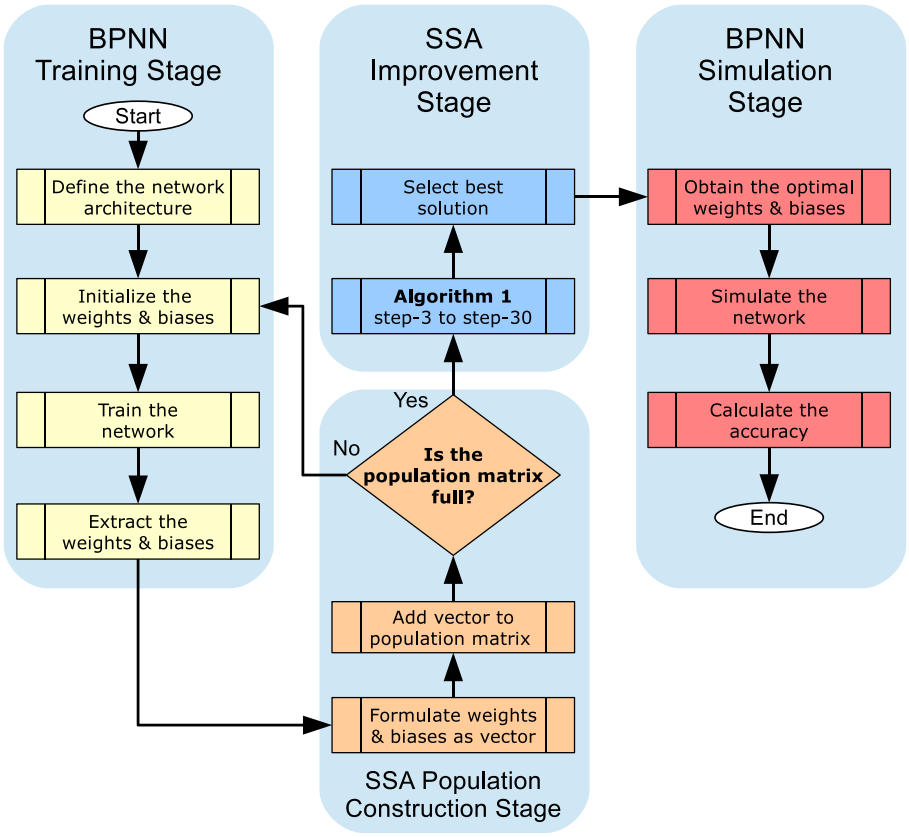

Fig. 3. Flowchart of the Proposed Algorithm SSA-BPNN.

Step 4: Apply the stop criterion: Step-3 is repeated until the maximum number of iterations is reached. Thereafter, the output of the SSA is the best solution position.

The pseudo-code of the SSA optimization process is provided in Algorithm 1.

Algorithm 1. Salp Swarm Algorithm Pseudo Code

=== **Stage 1: Initialization: Random Population** =====
initialize a random initial population as Pop_{init}
calculate the initial fitness of all solutions in Pop_{init}
find the best solution referred as Sol_{best}
=== **Stage 2: Improvement: Salp Swarm Algorithm** ===
set maximum number of iterations L
set counter $l \leftarrow 1$
while $(l < L)$ **do**
  update p_1 using Eq. (9)
  for each solution in Pop_{init} **do**
    update the first salp using Eq. (8)
    update the remaining salps using Eq. (10)
  end for
  update Pop_{init} referred as Pop_{temp}
  for each salp in each solution in Pop_{temp} **do**
    if $x_j^i > ub$ **then**
      $x_j^i = ub$
    else if $x_j^i < lb$ **then**
      $x_j^i = lb$
    end if
  end for
  update Pop_{temp} referred as Pop_{new}
  calculate the fitness for all solutions in Pop_{new}
  $l++$
end while
select best solution in Pop_{new} referred as Sol_{new}
if Sol_{new} is better than Sol_{best} **then**
  $Sol_{best} = Sol_{new}$
end if
return Sol_{best}

4. Proposed method

In order to demonstrate the SSA optimizer efficiency, this section presents an application of the SSA on training the neural network. Whereas to validate of any metaheuristic optimization algorithm involves applying it to improve a real-life complex problem. Therefore, this study combines the SSA with a BPNN to construct a novel hybrid algorithm named SSA-BPNN. The proposed hybrid algorithm is designed to find the optimal parameters (weights and biases) for the BPNN through leveraging salp swarm optimization and to thereby boost the prediction accuracy of the BPNN. The SSA acts as an optimizer for the BPNN weights and biases after finishing its training stage. Then, the optimized weights and biases obtained by the SSA are returned back to the BPNN to complete the implementation of the simulation phase. This procedure is applied to several benchmark datasets, and the results are compared with state-of-the-art methods and other optimization algorithms based approaches. The workflow of the proposed algorithm is illustrated in Fig. 3 and the procedural steps are described below, which consists of four stages:

4.1. BPNN network training

Step 1: Configure the BPNN network architecture: the architecture of any ANN is controlled by two main hyperparameters, the layers number and the nodes (neurons) number in each hidden layer. Values for these hyperparameters must be specified when configuring any network. The first step in the proposed method is initializing the BPNN network architecture, which is as follows:

Step 1.1: Define the number of layers: it is crucial to select an appropriate number of hidden layers because this decision affects the robustness (Ren et al., 2014) and generalization performance (Rumelhart et al., 1995) of the BPNN network. In this study the Kolmogorov's theorem was followed (Wang et al., 2019; Hornik et al., 1989), which states that three layers of a BPNN can approximate any continuous function. That means one hidden layer is considered adequate for the experiments conducted in this study (Rumelhart et al., 1995; Wittek, 2014). So, the architecture of the BPNN used in the proposed method consists of three layers, which allocated as one input layer, one hidden layer, and one output layer.

Step 1.2: Define the number of neurons: in the input layer, the number of input nodes is equal to the number of features in the dataset, which mean that in each dataset application, the input layer nodes number is determined based on the dataset features number, while the features number vary from database to another, so the input layer nodes number will change accordingly. Also, in the output layer, the number of output nodes is determined as one node based on the dataset output variables. On the other hand, to obtain highly accurate prediction results, which highly depend on the hidden layer nodes number, this study utilizes the Hecht-Nelson method (Hecht-Nielsen, 1987) to identify hidden layer nodes number. According to the Hecht-Nelson method, the number of hidden layer nodes must equal $2n+1$, where n is the number of input layer nodes (Ren et al., 2014).

Step 2: Initialize the BPNN parameters (weights and biases): in principle, the weights (ω_{ij}) and biases (b_i) of the BPNN network layers are initialized randomly. This is due to the expectation of the stochastic optimization method that utilizes to train the BPNN network model, which is called stochastic gradient descent.

Step 3: Train the BPNN network: after defining the BPNN network architecture and initializing the weights and biases, the BPNN network is ready for the training step by applying the training input data (the fault data which used as a sample data for BPNN). The training input data consists of 70% of the total content of the dataset, which is chosen randomly. Note that the remaining (30%) of the dataset content is utilized as simulation data.

Then the training process starts by employing the suggested BPNN network architecture and the initialized weights and biases in addition to the chosen training input data. In the training process, the BPNN network weights and biases are updated (improved) based on the estimation error (e) as descried in Section 3.1 Step 5 and 6. However, regardless of the amount of improvement cycles in the BPNN network weights, usually it will be insufficient, because the estimation error (e) will remain higher than the desired value. The training process continues until one of the following termination criteria is met, i.e., by reaching a reasonable (e) value, by reaching the maximum iteration limit, or when no more enhancements can be found.

Step 4: Extract the BPNN network weights and biases: after step 3 is completed, the new updated weights and biases are extracted from the BPNN network, and forwarded to the SSA optimizer, in order to be optimized.

4.2. SSA population construction

Step 1: Formulate weights & **biases as vector**: The updated weights and biases extracted in step-4 of the previous stage were formulated as a vector. This vector as shown in the following model, will be treated later as a single SSA solution.

$x = (w_{11}, w_{12}, \ldots, w_{1m}, w_{21}, w_{22}, \ldots, w_{2m}, \ldots, w_{z1}, w_{z2}, \ldots, w_{zm}, b_1, b_2)$where z is the number of the input layer nodes and m is the number of the hidden layer nodes.

The initial population for the SSA consists of several solutions, where each solution is a vector whose size has been previously specified. Each formulated vector is treated as a single solution for the SSA. So, to construct the initial population matrix for the SSA, steps 2 to 4 in Stage 4.1 have to be repeated according to the SSA population size.

Step 2: Add vector to population matrix: All the generated vectors in the previous step are added iteratively to a 2-dimensional matrix to construct the SSA initial population. Below is the template for the SSA initial population matrix:

$$\begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & \cdots & x_d^1 \\ x_1^2 & x_2^2 & x_3^2 & \cdots & x_d^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^i & x_2^i & x_3^i & \cdots & x_d^i \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^n & x_2^n & x_3^n & \cdots & x_d^n \end{bmatrix}$$

where the initial population matrix is made up of n-solutions with d-dimensions. So, the matrix structure is $(n * d)$.

4.3. SSA improvement

Step 1: Algorithm 1 step-3 to step-30: SSA utilizes the constructed initial population in Stage 4.2 to start its optimization process by calculating the initial fitness for all the solutions and defining the best solution. The SSA optimization process requires several iterations, which specified as in Table 1. In each iteration, the fitness is calculated and compared with the initial fitness which specified by BPNN Network during Stage 4.1, so that the best fitness is obtained and stored. The outcome of the completed optimization process is a new optimized population. Algorithm 1 step-3 to step-30 describes the SSA Improvement process in detail.

Step 2: Select the best solution: The best optimal solution in the population that has the best fitness is selected and redirected back to the BPNN Network to be used as a new optimal weight and biases.

Table 1
Experimental parameters configuration.

Parameter	Value
SSA Population Size	30
SSA Max iteration (L)	300
BPNN hidden layers No.	based on Kolmogorov theorem (Wang et al., 2019; Hornik et al., 1989)
Hidden layer neurons No.	based on Hecht–Nelson method (Ren et al., 2014; Hecht-Nielsen, 1987)
Experiment Runs No.	30

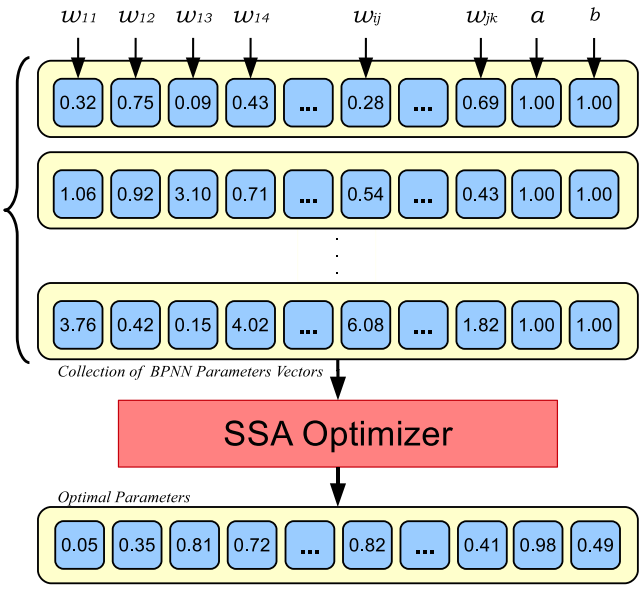

Fig. 4. Example of Handling the BPNN weights and biases by the SSA Optimizer.

4.4. BPNN simulation

Step 1: Obtain the optimal parameters: BPNN network utilized new optimal weights and biases that had been improved by SSA optimizer.
Step 2: Simulate the network: At this step the new optimal weights and biases along with the simulation input data (the fault data which used as a sample data for BPNN) utilized to simulate BPNN network. Based on the simulation results, the estimation error e is calculated.
Step 3: Calculate the accuracy: Based on the estimation error e the prediction accuracy is calculated, by calculating the MSE as the main accuracy measure (main objective function) as well as the several other measurements that are mentioned in Section 5.2. The prediction accuracy determine how much the prediction produced by the proposed method deviates from the target values (simulation output data).

For example, a set of BPNN parameters (weights and biases) vectors are shown in the upper side of the Fig. 4, where they are utilized by the SSA algorithm as a population (each vector is a single solution). After these vectors being optimized by the SSA algorithm, the best one utilized by the BPNN as new optimal parameters as shown in the bottom side of the Fig. 4. In addition, the pseudo code of the proposed method is given in Algorithm 2.

Algorithm 2. Proposed Method Pseudo-code

1: === **Stage 1: Network Initialization** ==========
2: input layer size (z) = dataset features No
3: hidden layer size = $zn + 1$
4: output layer size = 1
5: === **Stage 2: SSA Population (Pop_{init}) Initialization** ===
6: set population size ($PopSize$)
7: set counter $i \leftarrow 1$
8: **while** ($i =< PopSize$) **do**
9: initialize random parameters
10: train network using training data
11: $Pop_{init}(i)$ = getWeights(network)
12: $i++$
13: **end while**
14: calculate the initial fitness of all solutions in Pop_{init}
15: === **Stage 3: Improvement: SSA** ==========
16: as in Algorithm 1 *(from line-6 to line-30)*
17: === **Stage 4: Network Re-initialization** ==========
18: network weights = Sol_{best}
19: test network using test data
20: $Accuracy = |TestOutput - ActualOutput|$
21: Return Accuracy

5. Experiments and results

This section discusses the evaluation of the proposed method performance in improving the prediction accuracy of a software fault problem. In order to achieve this, variety of experiments were conducted over 22 different SFP datasets that described in Section 5.3. The first experiment was done to compare the proposed method performance against traditional BPNN performance in Section 5.4. The second experiment was done to validate the proposed method without and with cross-validation in Section 5.5. The last experiment was done to validate the proposed method against those methods in the literature in Section 5.6. Furthermore, the parameters configuration will be discussed in Section 5.1, then the performance measure will be discussed in Section 5.2.

5.1. Parameters configuration

Table 1 shows the parameter settings for the SSA and BPNN that were utilized in this research. The best settings for some of the SSA parameter values (i.e., population size and max iteration (L)) were chosen based on the result of numerous trial-and-error experiments. In addition, the BPNN parameter values (i.e., number of hidden layers in the network and number of neurons in the hidden layers) were selected as described in subSection 4 Step-1. These parameters were used in all the experiments.

5.2. Performance measure

No one method is considered to be a common standard technique for evaluating the performance of any algorithm. Therefore, several evaluation criteria are used in this research, such as Sensitivity, Specificity, Accuracy, Error Rate, and area under ROC curve (AUC). Except AUC, all the above evaluation criteria are affected by the cut-off value of on the estimated probability of defect instances, where the default value of cut-off is 0.5, and it is not the preferred cut-off value while evaluating a predictor/classifier (Zhang et al., 2016). A plenty of research in the literature encourages the use of AUC value to achieve better evaluation for any model (Ghotra et al., 2015; Lessmann et al., 2008). So, this research considers the AUC value to evaluate the proposed method.

In brief, the AUC value depends on the trade-off among True Positive (tp) rate against False Positive (fp) rate. The AUC value can be calculated on the basis of a confusion matrix as in Fig. 5. Where the term tp refers to the number of correctly predicted defective classes, while the term tn refers to the number of correctly predicted non defective classes. On the other hand, the term fp denotes the number of non-defective classes incorrectly predicted defective classes, while the term fn denotes the number of defective classes incorrectly predicted non defective classes. Also, confusion matrix can provide another important measure: specificity, sensitivity, accuracy, error rate (ER) and that defined as in Eqs. (11a), (11b), (12a), and (12b) respectively.

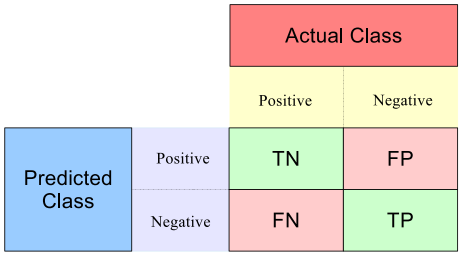

Figure 5: Confusion Matrix

Fig. 5. Confusion Matrix.

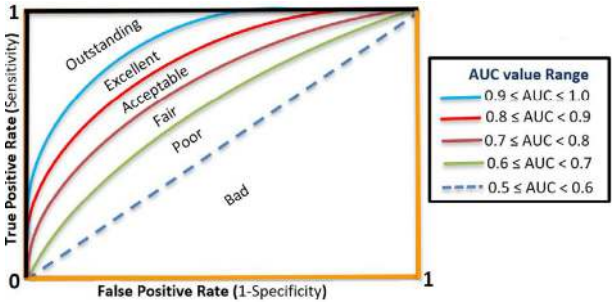

Fig. 6. ROC Curves and AUC values.

In addition, the AUC measures the probability that the randomly selected defective entity will be ranked higher than the randomly selected clean entity. The evaluation rules for any predictor/classifier using AUC is illustrated in Fig. 6 (Wi, 2000). The researchers can generalize the results even if the data distribution is changed by using the AUC measures (Koru et al., 2008).

$$\text{Specificity} = \frac{tn}{tn + fp} \tag{11a}$$

$$\text{Sensitivity} = \frac{tp}{tp + fn} \tag{11b}$$

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn} \tag{12a}$$

$$\text{ER} = 1 - \frac{tp + tn}{tp + tn + fp + fn} \tag{12b}$$

5.3. SFP datasets used

This research paper utilizes variety of public benchmark datasets to make the research verifiable, repeatable, and refutable. All the datasets are characterized for the SFP problem. These datasets are established in the literature (Jayanthi and Florence, 2019; Miholca et al., 2018; Li et al., 2016) and it is collected from PROMISE, which is a public engineering repository.[1] Table 2 contains a brief description of these datasets including their number of features, as well as number of faults and percentage of faults. From the table, it can be seen that these datasets vary in both complexity and size.

The experiments consider twenty-two datasets. Four datasets were obtained from the PROMISE repository, namely, KC1, JM1, PC3, and PC4. The KC1 dataset has a size close to 2k and the JM1 dataset has a size close to 9k, and each of these datasets has twenty-two features. The PC3 and PC4 datasets have a size of more than 1 k and each of them has thirty-eight features (Jayanthi and Florence, 2019).

In addition, five of these datasets, namely, Ar1, Ar3, Ar4, Ar5, and Ar6, were acquired from publicly available embedded software developed in C language Marian et al., 2015, and available in OpenML repository.[2] Another five were JEdit-4.0, JEdit-4.2, JEdit-4.3, Ant-1.7, and Tomcat-6.0, which are also publicly available datasets and have been utilized in most previous studies (Miholca et al., 2018), and were obtained from the PROMISE repository. All these dataset's descriptions are summarized in Table 2 and each one corresponds to a software application. The class label denoting whether the entity was defective or non-defective.

This research also utilizes eight NASA datasets (kc2, kc3, cm1, mc2, mw1, pc1, pc2, and pc5) that contain satellite, ground station, and simulation data, which were extracted from the PROMISE and OpenML repositories. The dataset descriptions are shown in Table 2. Each dataset fault percentage (*Faults*(%)) indicates that the SFP problem is a sort of imbalanced learning problem (Li et al., 2016). Variety of attributes is applied to describe software application, such as McCabe measure, Halstead measure, number of operators, number of code lines, etc.

5.4. Results of the proposed method against traditional BPNN

As mentioned before, an independent of thirty replicated runs were conducted to evaluate the performance of the proposed method without cross-validation and that of the conventional BPNN. The results are presented in Table 3. The values of the performance measures for each method on each dataset were calculated based on the average results over thirty replicated runs.

It can be noted from the table that the SSA-BPNN results (highlighted in gray) were better than those of the conventional BPNN algorithm in respect of accuracy for all of the datasets. In short, the outcomes of the proposed method show that using the SSA algorithm to optimize the BPNN parameters leads to a notable improvement in performance in terms of prediction accuracy, as compared to the conventional BPNN.

5.5. Results of the proposed method without and with cross-validation

In traditional ANN training process, data is divided into two main parts, training part 70%, and testing part 30%. This type of data division may lead to over-learning the problem, where the ANN gain a lot of information related to individual cases, and leaving too much information related to a general case. In order to solve this problem, this research utilizes the cross-validation (CV) method that divides the dataset into k-fold number, where self-training is applied based on k cross-validation (k = 10). Each employed the dataset was divided into 10 random portions. Repeatedly, 9-portions were used for training (90%), and 1-portion was used for testing (10%) in each validation session, where the average results for all sessions was taken into account. The main aim of the cross-validation technique is to predict the anticipated level of fit of a model for a data-independent dataset that utilized to train the model.

The results of the proposed method with and without cross-validation are reported in Table 4. The performance for the proposed BPNN-SSA with cross-validation for KC1, KC2, KC3, Ar4, Ar5, JEdit-4.0, JEdit-4.2, JEdit-4.3, Ant-1.7, Tomcat-6.0, CM1, MC2, MW1, PC1, and PC5 datasets is improved in term of the average

[1] http://promise.site.uottawa.ca/SERepository.

[2] http://www.openml.org/.

Table 2
Description of the Datasets.

Dataset	Features No.	Faults No.	Faults(%)	Dataset	Features No.	Faults No.	Faults(%)
KC1	22	325	15.50	Tomcat-6.0	20	77	8.97
KC2	21	107	20.49	MW1	37	31	7.69
KC3	39	43	9.38	JEdit-4.0	20	75	24.50
JM1	22	1759	18.34	JEdit-4.2	20	48	13.08
Ar1	29	8	7.40	JEdit-4.3	29	10	2.03
Ar3	29	8	12.70	Ant-1.7	20	166	22.28
Ar4	29	20	18.69	PC1	37	76	6.94
Ar5	29	8	22.22	PC2	36	23	2.15
Ar6	29	15	14.85	PC3	38	140	12.40
CM1	22	49	9.83	PC4	38	178	12.72
MC2	39	52	32.29	PC5	38	516	3.00

Table 3
Performance Results of the Proposed SSA-BPNN Against Traditional BPNN.

Dataset	Method	Sensitivity	Specificity	Accuracy	ER	AUC
KC1	BPNN	0.1529	0.9726	0.8626	0.1374	0.76
	SSA-BPNN	0.3441	0.9833	0.8892	0.1108	**0.79**
KC2	BPNN	0.4545	0.9516	0.8471	0.1529	0.77
	SSA-BPNN	0.4848	0.9919	0.8854	0.1146	**0.85**
KC3	BPNN	0.2222	0.9225	0.8768	0.1232	0.68
	SSA-BPNN	0.3333	0.9767	0.9348	0.0652	**0.92**
JM1	BPNN	0.1350	0.9722	0.8184	0.1816	0.68
	SSA-BPNN	0.0767	0.9876	0.8203	0.1797	**0.70**
Ar1	BPNN	0.3333	0.8824	0.8378	0.1622	0.59
	SSA-BPNN	1.0000	0.8000	0.9459	0.0541	**0.98**
Ar3	BPNN	0.2000	0.9286	0.7368	0.2632	0.57
	SSA-BPNN	0.5000	1.0000	0.8947	0.1053	**0.95**
Ar4	BPNN	0.1429	0.9615	0.7879	0.2121	0.70
	SSA-BPNN	0.7143	0.8462	0.8182	0.1818	**0.89**
Ar5	BPNN	0.6667	0.7500	0.7273	0.2727	0.89
	SSA-BPNN	0.6667	0.8750	0.8182	0.1818	**0.91**
Ar6	BPNN	0.1429	0.9583	0.7742	0.2258	0.67
	SSA-BPNN	0.4286	0.9583	0.8387	0.1613	**0.92**
JEdit-4.0	BPNN	0.3043	0.8406	0.7065	0.2935	0.64
	SSA-BPNN	0.3478	0.9275	0.7826	0.2174	**0.85**
JEdit-4.2	BPNN	0.4375	0.9263	0.8559	0.1441	0.72
	SSA-BPNN	0.1875	0.9789	0.8649	0.1351	**0.90**
JEdit-4.3	BPNN	0.0000	0.9586	0.9392	0.0608	0.85
	SSA-BPNN	0.0000	0.9862	0.9662	0.0338	**0.97**
Ant-1.7	BPNN	0.3889	0.9118	0.7857	0.2143	0.74
	SSA-BPNN	0.2222	0.9765	0.7946	0.2054	**0.79**
Tomcat-6.0	BPNN	0.2727	0.9407	0.8837	0.1163	0.73
	SSA-BPNN	0.5000	0.9025	0.8682	0.1318	**0.89**
CM1	BPNN	0.0000	0.9621	0.8467	0.1533	0.74
	SSA-BPNN	0.1667	0.9773	0.8800	0.1200	**0.85**
MC2	BPNN	0.6364	0.9211	0.8571	0.1429	0.69
	SSA-BPNN	0.7000	0.9474	0.8958	0.1042	**0.82**
MW1	BPNN	0.2000	0.9730	0.9091	0.0909	0.74
	SSA-BPNN	0.7000	0.9640	0.9421	0.0579	**0.93**
PC1	BPNN	0.0385	0.9902	0.9159	0.0841	0.72
	SSA-BPNN	0.1154	0.9739	0.9069	0.0931	**0.79**
PC2	BPNN	0.8750	0.3134	0.3160	0.6840	0.70
	SSA-BPNN	0.2500	1.0000	0.9964	0.0036	**0.93**
PC3	BPNN	0.2200	0.9499	0.8721	0.1279	0.73
	SSA-BPNN	0.1000	0.9809	0.8870	0.1130	**0.87**
PC4	BPNN	0.4615	0.9663	0.9064	0.0936	0.87
	SSA-BPNN	0.4231	0.9741	0.9087	0.0913	**0.90**
PC5	BPNN	0.1950	0.9948	0.9701	0.0299	0.90
	SSA-BPNN	0.2075	0.9946	0.9703	0.0297	**0.92**

of AUC results. While the performance for the proposed BPNN-SSA without cross-validation for JM1, Ar1, Ar3, Ar6, PC2, PC3, and PC4 have seen a decline. In conclusion, the BPNN-SSA with cross-validation outperforms those without cross-validation in 15 out of 22 datasets.

5.6. Results of the proposed method against state-of-the-art

A further evaluation for the proposed method performance was conducted by comparing its results with those produced by thirteen state-of-the-art algorithms. The same datasets were used to

Table 4
Performance Results of the Proposed SSA-BPNN without and with Cross-Validation.

Dataset	CV	Sensitivity	Specificity	Accuracy	ER	AUC
KC1	without	0.3441	0.9833	0.8892	0.1108	0.7900
	with	0.3882	0.9580	0.8815	0.1185	**0.8076**
KC2	without	0.4848	0.9919	0.8854	0.1146	0.8500
	with	0.6667	0.9597	0.8981	0.1019	**0.8512**
KC3	without	0.3333	0.9767	0.9348	0.0652	0.9200
	with	1.0000	0.4940	0.5435	0.4565	**0.9330**
JM1	without	0.0767	0.9876	0.8203	0.1797	**0.7000**
	with	0.0304	0.9962	0.8114	0.1886	0.6934
Ar1	without	1.0000	0.8000	0.9459	0.0541	**0.9800**
	with	0.0000	1.0000	0.9583	0.0417	0.9792
Ar3	without	0.5000	1.0000	0.8947	0.1053	**0.9500**
	with	0.5000	1.0000	0.8947	0.1053	0.8947
Ar4	without	0.7143	0.8462	0.8182	0.1818	0.8900
	with	1.0000	0.6923	0.7576	0.2424	**0.8939**
Ar5	without	0.6667	0.8750	0.8182	0.1818	0.9100
	with	0.0000	1.0000	0.8571	0.1429	**0.9286**
Ar6	without	0.4286	0.9583	0.8387	0.1613	**0.9200**
	with	0.8750	1.0000	0.9677	0.0323	0.8710
JEdit-4.0	without	0.3478	0.9275	0.7826	0.2174	0.8500
	with	1.0000	0.3333	0.4754	0.5246	**0.8758**
JEdit-4.2	without	0.1875	0.9789	0.8649	0.1351	0.9000
	with	0.7500	1.0000	0.9640	0.0360	**0.9279**
JEdit-4.3	without	0.0000	0.9862	0.9662	0.0338	0.9700
	with	0.3333	1.0000	0.9800	0.0200	**0.9767**
Ant-1.7	without	0.2222	0.9765	0.7946	0.2054	0.7900
	with	0.3725	0.9760	0.8349	0.1651	**0.8058**
Tomcat-6.0	without	0.5000	0.9025	0.8682	0.1318	0.8900
	with	0.0000	0.9915	0.9070	0.0930	0.9008
CM1	without	0.1667	0.9773	0.8800	0.1200	0.8500
	with	1.0000	0.0606	0.1733	0.8267	**0.8609**
MC2	without	0.7000	0.9474	0.8958	0.1042	0.8200
	with	0.1000	1.0000	0.7188	0.2813	**0.8438**
MW1	without	0.7000	0.9640	0.9421	0.0579	0.9300
	with	0.9000	0.9730	0.9669	0.0331	**0.9496**
PC1	without	0.1154	0.9739	0.9069	0.0931	0.7900
	with	0.0000	0.9854	0.9186	0.0814	**0.8279**
PC2	without	0.2500	1.0000	0.9964	0.0036	**0.9300**
	with	1.0000	0.0009	0.0054	0.9946	0.9235
PC3	without	0.1000	0.9809	0.8870	0.1130	**0.8700**
	with	0.5400	0.9332	0.8913	0.1087	0.8570
PC4	without	0.4231	0.9741	0.9087	0.0913	**0.9000**
	with	0.4444	0.9961	0.9281	0.0719	0.8986
PC5	without	0.2075	0.9946	0.9703	0.0297	0.9200
	with	1.0000	0.6996	0.7089	0.2911	**0.9484**

ensure direct comparability of the results. Table 5 provides the abbreviations used for the comparative algorithms employed in this evaluation.

Table 6 present the outcomes of eight of the above-listed comparative methods Jayanthi and Florence, 2019 versus those of the SSA-BPNN method for four datasets in respect of four performance measures (AUC, sensitivity, specificity, and accuracy). Results show that the SSA-BPNN method outperforms the compared algorithms in the case of only one dataset in respect of the AUC performance measure. On the other hand, Table 7 shows that the SSA-BPNN method outperforms four of the comparative algorithms (Jayanthi and Florence, 2019) in most datasets in respect of sensitivity, specificity and accuracy. So, in a nutshell, it can be said that the proposed SSA-BPNN method has a powerful ability to solve the SFP problem.

The performance of the proposed SSA-BPNN method was also compared against that of the HyGRAR method (Miholca et al., 2018) for ten datasets with respect to five performance measures (confusion matrix, sensitivity, specificity, accuracy, and AUC). From the results in Table 8, it can be seen that the proposed SSA-BPNN algorithm outperforms the other methods (A41 through Ar6) for the vast majority of datasets.

Furthermore, the results of the proposed SSA-BPNN method were compared against those of three comparative methods (Li et al., 2016) for eleven datasets in respect of one performance measure (i.e., accuracy). The results in Table 9 shows that the proposed SSA-BPNN outperforms the other three methods in all datasets, from which it can be concluded that it is more efficient for the SFP problem than the 3WD, TS3WD and 2WD methods.

Finally, for further validation, the results of the proposed SSA-BPNN method were compared against other optimization algorithm (Turabieh et al., 2019) for two situations (i.e., with and without cross-validation) over four datasets in term of AUC. The results in Table 10 shows that the proposed SSA-BPNN outperforms the L-RNN algorithm without and with cross-validation in all datasets. It can be concluded that proposed SSA-BPNN is more efficient for the SFP problem.

In conclusion, from the above evaluations, it can be said that the combination between the SSA algorithm and the Backpropagation neural network added a significant improvement to the results in majority of the databases. This might be due to the type of datasets used as opposed to any intrinsic drawbacks in the proposed technique itself.

Table 5
State-of-the-Art Comparative Methods for SFP.

Abbr.	Method Name	Ref.	Abbr.	Method Name	Ref.
k-NN	k Nearest Neighbor	Jayanthi and Florence (2019) and Lessmann et al. (2008)	ANN-PCA	ANN Principal Component Analysis	Jayanthi and Florence (2019) and Nastac and Dan Cristea (2012)
SVM	Support Vector Machine	Jayanthi and Florence (2019) and Lessmann et al. (2008)	HyGRAR	Gradual Association Rule Mining and ANN	Miholca et al. (2018)
L-SVM	Lagrangian Support Vector Machine	Jayanthi and Florence (2019) and Lessmann et al. (2008)	3WD	Three-way decisions based Naive Bayes	Li et al. (2016)
LS-SVM	Least Squares Support Vector Machine	Jayanthi and Florence (2019) and Lessmann et al. (2008)	TS3WD	Three-way decisions based two-stage method	Li et al. (2016)
NB	Naïve Bayes	Jayanthi and Florence (2019)	2WD	Two-way decisions based Naive Bayes	Li et al. (2016)
LDA	Linear Discriminant Analysis	Jayanthi and Florence (2019) and Lessmann et al. (2008)	L-RNN	Layered Recurrent Neural Network	Turabieh et al. (2019)

Table 6
Results of the Proposed SSA-BPNN Against Seven other Algorithms in terms of AUC.

Dataset	k-NN	SVM	L-SVM	LS-SVM	NB	LDA	ANN-PCA	SSA-BPNN
KC1	0.70	0.76	0.76	0.77	0.76	0.78	**0.79**	**0.79**
JM1	0.69	0.72	0.73	0.74	0.69	0.73	**0.81**	0.70
PC3	0.77	0.77	0.84	0.83	0.81	0.82	**0.89**	0.87
PC4	0.87	0.92	0.92	0.94	0.85	0.88	**0.97**	0.90

Table 7
Results of the Proposed SSA-BPNN Against ANN-PCA Algorithm.

Dataset	Method	Sensitivity	Specificity	Accuracy
KC1	ANN-PCA	0.2270	0.9865	0.8691
	SSA-BPNN	0.3441	0.9833	**0.8892**
JM1	ANN-PCA	0.9805	0.1615	**0.8303**
	SSA-BPNN	0.0767	0.9876	0.8203
PC3	ANN-PCA	0.9959	0.3000	**0.9093**
	SSA-BPNN	0.1000	0.9809	0.8870
PC4	ANN-PCA	0.9787	0.6461	**0.9364**
	SSA-BPNN	0.4231	0.9741	0.9087

5.7. Statistical analysis

For more evaluation, the T-test will be utilized in this paper. This test uses regression and correlation values to compare algorithms and to evaluate how the two methods differ from each other. The P-value represents the probability of random validity of the hypothesis. The lower the P-value, the higher the tests statistical significance, which is confirmation against the null hypothesis, because when the null hypothesis is true, the data detected is unlikely. The null hypothesis is rejected or accepted on the basis of the P-value. Dependent on a probability threshold called (α), rejection or approval of the null hypothesis. The null hypothesis is denied if the P-value $<= \alpha$. In this sample, the value of alpha is 0.05, any probability value lower than 0.05 means that less than 5% of the experiment outcomes are attributed to chance and not the experimental factors i.e. the experimental factors are the ones that affected the experiment's digital observations, and thus there is a difference of statistical significance. The P-value and its statistical significance are shown in Table 11. As seen in Table 11, the

Table 8
Results of the Proposed SSA-BPNN Against HyGRAR algorithm.

Dataset	Method	Sensitivity	Specificity	Accuracy	AUC
Ar1	HyGRAR	0.556	0.991	0.959	0.773
	SSA-BPNN	1.000	0.800	0.946	**0.980**
Ar3	HyGRAR	1.000	0.800	0.825	0.900
	SSA-BPNN	0.500	1.000	0.895	**0.950**
Ar4	HyGRAR	0.900	0.989	0.972	**0.944**
	SSA-BPNN	0.714	0.846	0.818	0.890
Ar5	HyGRAR	0.875	0.964	0.944	**0.920**
	SSA-BPNN	0.667	0.875	0.818	0.910
Ar6	HyGRAR	0.667	0.988	0.941	0.828
	SSA-BPNN	0.429	0.958	0.839	**0.920**
Jedit-4.0	HyGRAR	0.720	0.996	0.928	**0.858**
	SSA-BPNN	0.348	0.928	0.783	0.850
Jedit-4.2	HyGRAR	0.813	0.994	0.970	**0.903**
	SSA-BPNN	0.188	0.979	0.865	0.900
Jedit-4.3	HyGRAR	0.636	0.994	0.986	0.815
	SSA-BPNN	0.000	0.987	0.966	**0.970**
Ant-1.7	HyGRAR	0.855	0.997	0.965	**0.926**
	SSA-BPNN	0.222	0.977	0.795	0.790
Tomcat 6.0	HyGRAR	0.844	0.997	0.984	**0.921**
	SSA-BPNN	0.500	0.903	0.868	0.884

Table 9
Results of the Proposed SSA-BPNN Against Three other Algorithms in terms of Accuracy.

Dataset	3WD	TS3WD	2WD	SSA-BPNN
CM1	0.8253	0.8223	0.8229	**0.8800**
JM1	0.7835	0.7823	0.7819	**0.8203**
KC2	0.8376	0.8351	0.8352	**0.8854**
KC3	0.7894	0.7864	0.7892	**0.9348**
MC2	0.7162	0.7174	0.7192	**0.8958**
MW1	0.8225	0.819	0.8135	**0.9421**
PC1	0.887	0.8851	0.8808	**0.9069**
PC2	0.9194	0.918	0.9072	**0.9964**
PC3	0.4019	0.4253	0.3565	**0.8870**
PC4	0.8748	0.8765	0.8704	**0.9087**
PC5	0.9647	0.9647	0.9642	**0.9703**

Table 10
Results of the Proposed Method Against L-RNN Algorithm in terms of AUC.

Dataset	Results Without Cross-Validation		Results With Cross-Validation	
	L-RNN	SSA-BPNN	L-RNN	SSA-BPNN
Ant-1.7	0.6047	**0.7900**	0.7382	**0.8058**
Jedit-4.0	0.8442	**0.8500**	0.8301	**0.8758**
Jedit-4.2	0.6178	**0.9000**	0.8555	**0.9279**
Jedit-4.3	0.6955	**0.9700**	0.9091	**0.9767**

Table 11
The Statistics and P-Values of T-test for Accuracy of BPNN and SSA-BPNN.

Dataset	Method	Average	Standard Deviation (std)	Mean std Error	P-Value	Dataset	Method	Average	Standard Deviation (std)	Mean std Error	P-Value
Ant-1.7	BPNN	0.7602	0.0085	0.0016	0.00	KC1	BPNN	0.8601	0.0090	0.0016	0.00
	SSA-BPNN	0.7936	0.0059	0.0011			SSA-BPNN	0.8989	0.0075	0.0014	
Ar1	BPNN	0.8014	0.1258	0.0230	0.00	KC2	BPNN	0.8390	0.0080	0.0015	0.00
	SSA-BPNN	0.9792	0.0000	0.0000			SSA-BPNN	0.8843	0.0029	0.0005	
Ar3	BPNN	0.9278	0.0408	0.0075	0.00	KC3	BPNN	0.8087	0.0353	0.0065	0.00
	SSA-BPNN	0.9583	0.0000	0.0000			SSA-BPNN	0.9108	0.0124	0.0023	
Ar4	BPNN	0.9667	0.0262	0.0048	0.00	MC2	BPNN	0.6912	0.0428	0.0078	0.00
	SSA-BPNN	0.9762	0.0000	0.0000			SSA-BPNN	0.8398	0.0061	0.0011	
Ar5	BPNN	0.9024	0.0663	0.0121	0.00	MW1	BPNN	0.8235	0.0446	0.0081	0.00
	SSA-BPNN	0.9286	0.0000	0.0000			SSA-BPNN	0.9485	0.0056	0.0010	
Ar6	BPNN	0.7142	0.1316	0.0240	0.00	PC1	BPNN	0.7757	0.0282	0.0051	0.00
	SSA-BPNN	0.9750	0.0000	0.0000			SSA-BPNN	0.8819	0.0153	0.0028	
CM1	BPNN	0.8028	0.0499	0.0091	0.00	PC2	BPNN	0.8745	0.0770	0.0141	0.00
	SSA-BPNN	0.9212	0.0033	0.0006			SSA-BPNN	0.9956	0.0014	0.0003	
Jedit-4.0	BPNN	0.7433	0.0255	0.0047	0.00	PC3	BPNN	0.8197	0.0147	0.0027	0.00
	SSA-BPNN	0.8429	0.0137	0.0025			SSA-BPNN	0.8604	0.0041	0.0008	
Jedit-4.2	BPNN	0.9506	0.0570	0.0104	0.00	PC4	BPNN	0.8752	0.0054	0.0010	0.00
	SSA-BPNN	0.9795	0.0000	0.0000			SSA-BPNN	0.8936	0.0023	0.0004	
Jedit-4.3	BPNN	0.9202	0.0723	0.0132	0.00	PC5	BPNN	0.9573	0.0066	0.0012	0.00
	SSA-BPNN	0.9831	0.0008	0.0002			SSA-BPNN	0.9693	0.0006	0.0006	
Tomcat-6.0	SSA-BPNN	0.8623	0.0404	0.0074	0.00	JM1	BPNN	0.7782	0.0082	0.0015	0.00
	BPNN	0.9356	0.0044	0.0008			SSA-BPNN	0.8078	0.0332	0.0061	

statistical data indicates that SSA-BPNN is highly valuable for BPNN since all P-values are less than 0.01, which means that less than 1% of the experiment's findings are due to chance and not to experimental factors.

6. Conclusion and future directions

This paper proposed combining the SSA with the BPNN to solve the SFP. In this hybridized model, the SSA is used to define the optimal parameters for BPNN, to thus enhance prediction accuracy. The proposed method – SSA-BPNN – was evaluated by applying it to a variety of SFP problem datasets. These datasets included twenty-two datasets with various characteristics that were obtained from the PROMISE repository. Six performance measures (AUC, confusion matrix, sensitivity, specificity, accuracy, and ER) were used in the evaluation of the proposed method. The evaluation was carried out in two phases. In the first phase, a comparison of the results obtained by the proposed SSA-BPNN and those obtained by the conventional BPNN was carried out. Overall, the comparison showed that SSA-BPNN was better than the conventional BPNN for all twenty-two datasets. In the second phase, the performance of the proposed SSA-BPNN was compared against that of thirteen state-of-the-art algorithms on the same datasets. The results showed that SSA-BPNN outperformed the other algorithms in most datasets in respect of most of the performance measures. Hence, it can be concluded that the hybridized model SSA-BPNN is a valuable addition to the toolbox for solving software engineering issues that can be utilized to obtain higher prediction accuracy in respect of a wide range of SFP problems, in addition to copes the drawbacks of earlier SFP methods by having a higher resolution and lower error rate. Furthermore, because the proposed SSA-BPNN produced fruitful results for SFP problems, this implies that the model can be utilized for other prediction problems in the future. A one limitation noted in the proposed method is the high computational cost over the majority of data sets. Therefore, a new strategy to optimize the proposed algorithm in terms of computational cost can be developed as potential future work.

Declaration of Competing Interest

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

Acknowledgment

This work is supported by University Kebangsaan Malaysia (FRGS/1/2019/ICT02/UKM/01/1).

References

Abaei, M. Golnoush, Mashinchi, Reza, Selamat, Ali, 2015a. Software fault prediction using bp-based crisp artificial neural networks. International Journal of Intelligent Information and Database Systems 9 (1), 15–31.

Abaei, Golnoush, Selamat, Ali, Fujita, Hamido, 2015b. An empirical study based on semi-supervised hybrid self-organizing map for software fault prediction. Knowledge-Based Systems 74, 28–39.

Abbassi, Rabeh, Abbassi, Abdelkader, Heidari, Ali Asghar, Mirjalili, Seyedali, 2019. An efficient salp swarm-inspired algorithm for parameters identification of photovoltaic cell models. Energy Conversion and Management 179, 362–372.

Abdullah, Dahlan, Pardede, A.M.H., Umami, Liza, Manurung, RosidaTiurma, Suryani, Rini, Surya, Sara, Saddhono, Kundharu, Mulyaningsih, Indrya, Ketut Sudarsana, I., Brata, DiahPujiNali, et al., 2019. Drug users prediction using backpropagation educational method. In: Journal of Physics: Conference Series, IOP Publishing, vol. 1361, p. 012055.

Abualigah, Laith, Shehab, Mohammad, Alshinwan, Mohammad, Alabool, Hamzeh, 2019. Salp swarm algorithm: a comprehensive survey. Neural Computing and Applications, 1–21.

Abusnaina, Ahmed A., Ahmad, Sobhi, Jarrar, Radi, Mafarja, Majdi, 2018. Training neural networks using salp swarm algorithm for pattern classification. In: Proceedings of the 2nd International Conference on Future Networks and Distributed Systems. ACM, p. 17.

Aljarah, Ibrahim, Mafarja, Majdi, Heidari, Ali Asghar, Faris, Hossam, Zhang, Yong, Mirjalili, Seyedali, 2018. Asynchronous accelerating multi-leader salp chains for feature selection. Applied Soft Computing, 71, 964–979.

Aljarah, Ibrahim, Habib, Maria, Faris, Hossam, Al-Madi, Nailah, Asghar Heidari, Ali, Mafarja, Majdi, Elaziz, Mohamed Abd, Mirjalili, Seyedali, 2020. A dynamic locality multi-objective salp swarm algorithm for feature selection. Computers & Industrial Engineering, 147, 106628.

Alsaeedi, Ali Hakem, Aljanabi, Ali Hussein, Manna, Mehdi Ebady, Albukhnefis, Adil L., 2020. A proactive metaheuristic model for optimizing weights of artificial neural network. Indonesian Journal of Electrical Engineering and Computer Science 20 (2), 976–984.

Alshareef, Almahdi Mohammed, Bakar, Azuraliza Abu, Hamdan, Abdul Razak, Syed Abdullah, Sharifah Mastura, Alweshah, Mohammed, 2015. A case-based reasoning approach for pattern detection in malaysia rainfall data. International Journal of Big Data Intelligence, 2 (4), 285–302.

Asaithambi, Sasikumar, Rajappa, Muthaiah, 2018. Swarm intelligence-based approach for optimal design of cmos differential amplifier and comparator circuit using a hybrid salp swarm algorithm. Review of Scientific Instruments 89, (5) 054702.

Aziz, Syed Rashid, Khan, Tamim Ahmed, Nadeem, Aamer, 2020. Efficacy of inheritance aspect in software fault prediction—a survey paper. IEEE Access 8, 170548–170567.

Baklacioglu, Tolga, Turan, Onder, Aydin, Hakan, 2018. Metaheuristic approach for an artificial neural network: exergetic sustainability and environmental effect of a business aircraft. Transportation Research Part D: Transport and Environment 63, 445–465.

Baklacioglu, Tolga, Turan, Onder, Aydin, Hakan, 2020. Metaheuristics optimized machine learning modelling for estimation of exergetic emissions of a propulsion system. In: MATEC Web of Conferences, EDP Sciences, vol. 314, p. 02001.

Barik, Amar Kumar, Das, Dulal Chandra, 2018. Active power management of isolated renewable microgrid generating power from rooftop solar arrays, sewage waters and solid urban wastes of a smart city using salp swarm algorithm. In 2018 Technologies for Smart-City Energy Security and Power (ICSESP), pages 1–6. IEEE, 2018.

Bowes, David, Hall, Tracy, Petrić, Jean, 2018. Software defect prediction: do different classifiers find the same defects? Software Quality Journal 26 (2), 525–552.

Bui, Quang-Thanh, 2019. Metaheuristic algorithms in optimizing neural network: a comparative study for forest fire susceptibility mapping in dak nong, Vietnam. Geomatics, Natural Hazards and Risk 10 (1), 136–150.

Carrozza, Gabriella, Cotroneo, Domenico, Natella, Roberto, Pietrantuono, Roberto, Russo, Stefano, 2013. Analysis and prediction of mandelbugs in an industrial software system. In: 2013 IEEE sixth international conference on software testing, verification and validation. IEEE, pp. 262–271.

Catal, Cagatay, 2011. Software fault prediction: a literature review and current trends. Expert Systems with Applications 38 (4), 4626–4636.

Chaudhary, Rashmi, Patel, Hitul, Scholar, M.E., 2015. A survey on backpropagation algorithm for neural networks. International Journal of Research in Engineering and Technology 2 (7).

Ekinci, Serdar, Hekimoglu, Baran, 2018. Parameter optimization of power system stabilizer via salp swarm algorithm. In: 2018 5th International Conference on Electrical and Electronic Engineering (ICEEE), IEEE, pp. 143–147.

Elaziz, Mohamed Abd, Heidari, Ali Asghar, Fujita, Hamido, Moayedi, Hossein, 2020. A competitive chain-based harris hawks optimizer for global optimization and multi-level image thresholding problems. Applied Soft Computing, 106347.

El-Fergany, Attia A, 2018. Extracting optimal parameters of pem fuel cells using salp swarm optimizer. Renewable Energy 119, 641–648.

El-Fergany, Attia A, Hasanien, Hany M, 2019. Salp swarm optimizer to solve optimal power flow comprising voltage stability analysis. Neural Computing and Applications, 1–17.

Ever, Yoney Kirsal, Dimililer, Kamil, Sekeroglu, Boran, 2019. Comparison of machine learning techniques for prediction problems. In: Workshops of the International Conference on Advanced Information Networking and Applications. Springer, pp. 713–723.

Faris, Hossam, Heidari, Ali Asghar, Ala'M, Al-Zoubi, Mafarja, Majdi, Aljarah, Ibrahim, Eshtay, Mohammed, Mirjalili, Seyedali, 2020. Time-varying hierarchical chains of salps with random weight networks for feature selection. Expert Systems with Applications, 140, 112898.

Felipe, Natália França, Cavalcanti, Raphael Pena, Bechelane Maia, Eduardo Habib, Amaral, Weber Porto, Farnese, Augusto Campos, Tavares, Leonardo Daniel, de Faria, Eustáquio São José, Pereira da Silva, Clarindo Isaias, de Pádua Paula Filho, Wilson, et al., 2014. A comparative study of three test effort estimation methods. Revista Cubana de Ciencias Informáticas, 8.

Ghotra, Baljinder, McIntosh, Shane, Hassan, Ahmed E., 2015. Revisiting the impact of classification techniques on the performance of defect prediction models. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, IEEE, pp. 789–800.

Guha, Dipayan, Roy, Provas, Banerjee, Subrata, 2018. A maiden application of salp swarm algorithm optimized cascade tilt-integral-derivative controller for load frequency control of power systems. IET Generation, Transmission and Distribution.

Han, Jiawei, Pei, Jian, Kamber, Micheline, 2011. Data Mining: Concepts and Techniques. Elsevier.

Hecht-Nielsen, Robert, 1987. Kolmogorov's mapping neural network existence theorem. Proceedings of the International Conference on Neural Networks, vol. 3. IEEE Press, New York, pp. 11–14.

Hecht-Nielsen, Robert, 1992. Theory of the backpropagation neural network. In: Neural Networks for Perception. Elsevier, pp. 65–93.

Hornik, Kurt, Stinchcombe, Maxwell, White, Halbert, et al., 1989. Multilayer feedforward networks are universal approximators. Neural Networks 2 (5), 359–366.

Hussien, Abdelazim G., Hassanien, Aboul Ella, Houssein, Essam H., 2017. Swarming behaviour of salps algorithm for predicting chemical compound activities. In: 2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS), IEEE, pp. 315–320.

Ibrahim, Rehab Ali, Ewees, Ahmed A., Oliva, Diego, Elaziz, Mohamed Abd, Lu, Songfeng, 2018. Improved salp swarm algorithm based on particle swarm optimization for feature selection. Journal of Ambient Intelligence and Humanized Computing, 1–15.

Jaddi, Najmeh Sadat, Abdullah, Salwani, Hamdan, Abdul Razak, 2015. Optimization of neural network model using modified bat-inspired algorithm. Applied Soft Computing 37, 71–86.

Jayanthi, R., Florence, Lilly, 2019. Software defect prediction techniques using metrics based on neural network classifier. Cluster Computing 22 (1), 77–88.

Karaboga, Dervis, Ozturk, Celal, 2009. Neural networks training by artificial bee colony algorithm on pattern classification. Neural Network World 19 (3), 279.

Khazaiepoor, Mahdi, Bardsiri, Amid Khatibi, Keynia, Farshid, 2020. A hybrid approach for software development effort estimation using neural networks, genetic algorithm, multiple linear regression and imperialist competitive algorithm. International Journal of Nonlinear Analysis and Applications 11 (1), 207–224.

Khoshgoftaar, Taghi M., Seliya, Naeem, 2003. Software quality classification modeling using the sprint decision tree algorithm. International Journal on Artificial Intelligence Tools 12 (03), 207–225.

Koru, A. Güneş, El Emam, Khaled, Zhang, Dongsong, Liu, Hongfang, Mathew, Divya, 2008. Theory of relative defect proneness. Empirical Software Engineering, 13 (5), 473.

Lessmann, Stefan, Baesens, Bart, Mues, Christophe, Pietsch, Swantje, 2008. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Transactions on Software Engineering 34 (4), 485–496.

Li, Weiwei, Huang, Zhiqiu, Li, Qing, 2016. Three-way decisions based software defect prediction. Knowledge-Based Systems 91, 263–274.

Li, Xiaoqing, Jiang, Qingquan, Hsu, Maxwell K, Chen, Qinglan, 2019. Support or risk? Software project risk assessment model based on rough set theory and backpropagation neural network. Sustainability 11 (17), 4513.

Madin, Larry P, 1990. Aspects of jet propulsion in salps. Canadian Journal of Zoology 68 (4), 765–777.

Malhotra, Ruchika, 2015. A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing 27, 504–518.

Mandal, Sudip, Saha, Goutam, Pal, Rajat K., 2015. Neural network training using firefly algorithm. Global Journal on Advancement in Engineering and Science (GJAES), 1 (1).

Marian, Zsuzsanna, Czibula, Gabriela, Czibula, Istvan-Gergley, Sotoc, Sergiu, 2015. Software defect detection using self-organizing maps. Studia Universitatis Babes-Bolyai, Informatica 60 (2), 55–69.

Mavrovouniotis, Michalis, Yang, Shengxiang, 2015. Training neural networks with ant colony optimization algorithms for pattern classification. Soft Computing 19 (6), 1511–1522.

Menzies, Tim, Greenwald, Jeremy, Frank, Art, 2006. Data mining static code attributes to learn defect predictors. IEEE Transactions on Software Engineering 33 (1), 2–13.

Miholca, Diana-Lucia, Czibula, Gabriela, Czibula, Istvan Gergely, 2018. A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks. Information Sciences 441, 152–170.

Mirjalili, Seyedali, Gandomi, Amir H., Zahra Mirjalili, Seyedeh, Saremi, Shahrzad, Faris, Hossam, Mirjalili, Seyed Mohammad, 2017. Salp swarm algorithm: A bio-inspired optimizer for engineering design problems. Advances in Engineering Software, 114, 163–191.

Nastac, Dumitru Iulian, Dan Cristea, Paul, 2012. An ann-pca adaptive forecasting model. In: 2012 19th International Conference on Systems, Signals and Image Processing (IWSSIP), IEEE, pp. 514–517.

Nawi, Nazri Mohd, Khan, Abdullah, Rehman, Mohammad Zubair, 2013. A new back-propagation neural network optimized with cuckoo search algorithm. In: International Conference on Computational Science and Its Applications. Springer, pp. 413–426.

Ogidan, Ezekiel T., Dimililer, Kamil, Ever, Yoney Kirsal, 2018. Machine learning for expert systems in data analysis. In: 2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT). IEEE, pp. 1–5.

Ojha, Varun Kumar, Abraham, Ajith, Snášel, Václav, 2017. Metaheuristic design of feedforward neural networks: A review of two decades of research. Engineering Applications of Artificial Intelligence 60, 97–116.

Porter, Adam A., Selby, Richard W., 1990. Empirically guided software development using metric-based classification trees. IEEE Software 7 (2), 46–54.

Qasem, Osama Al, Akour, Mohammed, Alenezi, Mamdouh, 2020. The influence of deep learning algorithms factors in software fault prediction. IEEE Access 8, 63945–63960.

Rakitianskaia, Anna S., Engelbrecht, Andries Petrus, 2012. Training feedforward neural networks with dynamic particle swarm optimisation. Swarm Intelligence 6 (3), 233–270.

Rashid, Junaid, Nisar, Muhammad Wasif, Mahmood, Toqeer, Rehman, Amjad, Arafat, Syed Yasser, et al., 2020. Study of software development cost estimation techniques and models. Mehran University Research Journal Of Engineering & Technology, 39 (2), 413.

Rathore, Santosh Singh, Kumar, Sandeep, 2017. Towards an ensemble based system for predicting the number of software faults. Expert Systems with Applications, 82, 357–382.

Ren, Chao, An, Ning, Wang, Jianzhou, Li, Lian, Bin, Hu., Shang, Duo, 2014. Optimal parameters selection for bp neural network based on particle swarm optimization: a case study of wind speed forecasting. Knowledge-based Systems 56, 226–239.

Rumelhart, David E., Durbin, Richard, Golden, Richard, Chauvin, Yves, 1995. Backpropagation: The basic theory. Backpropagation: Theory, Architectures and Applications, pp. 1–34.

Sekeroglu, Boran, Dimililer, Kamil, Tuncal, Kubra, 2019. Student performance prediction and classification using machine learning algorithms. In: Proceedings of the 2019 8th International Conference on Educational and Information Technology. ACM, pp. 7–11.

Sun, Wei, Huang, Chenchen, 2020. A carbon price prediction model based on secondary decomposition algorithm and optimized back propagation neural network. Journal of Cleaner Production 243, 118671.

Thwin, Mie Mie Thet, Quah, Tong-Seng, 2005. Application of neural networks for software quality prediction using object-oriented metrics. Journal of Systems and Software 76 (2), 147–156.

Tian, David, Deng, Jiamei, Vinod, Gopika, Santhosh, T.V., Tawfik, Hissam, 2018. A constraint-based genetic algorithm for optimizing neural network architectures for detection of loss of coolant accidents of nuclear power plants. Neurocomputing, 322, 102–119.

Turabieh, Hamza, Mafarja, Majdi, Li, Xiaodong, 2019. Iterated feature selection algorithms with layered recurrent neural network for software fault prediction. Expert Systems with Applications 122, 27–42.

Vora, Kuldip, Yagnik, Shruti, 2014. A survey on backpropagation algorithms for feedforward neural networks. International Journal of Engineering Development and Research (IJEDR).

Wang, Jiyang, Gao, Yuyang, Chen, Xuejun, 2018. A novel hybrid interval prediction approach based on modified lower upper bound estimation in combination with multi-objective salp swarm algorithm for short-term load forecasting. Energies 11 (6), 1561.

Wang, Yaoli, Wang, Lipo, Chang, Qing, Yang, Chunxia, 2019. Effects of direct input–output connections on multilayer perceptron neural networks for time series prediction. Soft Computing, 1–10.

Wi, David, 2000. Applied Logistic Regression. Wiley-Interscience Piblication, 2nd edition.

Wittek, Peter, 2014. Pattern recognition and neural networks. Quantum Machine Learning, 63–71.

Yuan, Xiaohong, Khoshgoftaar, Taghi M., Allen, Edward B., Ganesan, K., 2000. An application of fuzzy clustering to software quality prediction. In: Proceedings 3rd IEEE symposium on application-specific systems and software engineering technology. IEEE, pp. 85–90.

Zhang, Feng, Mockus, Audris, Keivanloo, Iman, Zou, Ying, 2016. Towards building a universal defect prediction model with rank transformed predictors. Empirical Software Engineering 21 (5), 2107–2145.

Zhang, Jing, Wang, Zhenhua, Luo, Xiong, 2018. Parameter estimation for soil water retention curve using the salp swarm algorithm. Water 10 (6), 815.

Zhang, Hongliang, Wang, Zhiyan, Chen, Weibin, Heidari, Ali Asghar, Wang, Mingjing, Zhao, Xuehua, Liang, Guoxi, Chen, Huiling, Zhang, Xin. Ensemble mutation-driven salp swarm algorithm with restart mechanism: Framework and fundamental analysis. Expert Systems with Applications, 165, 113897.