Bat optimization algorithm for wrapper-based feature selection and performance improvement of android malware detection

Ravi Kiran Varma P1 | S Kumar Reddy Mallidi2 | Santosh Jhansi K1
Pushpa Latha D1

1Department of CSE, MVGR College of Engineering, Vizianagaram, AP, India
2Department of CSE, Sri Vasavi College of Engineering, Tadepalligudem, AP, India

Correspondence
Ravi Kiran Varma P, Department of CSE, MVGR College of Engineering, Vizianagaram, AP, India.
Email: ravikiranvarmap@gmail.com

Abstract
Android malware is a serious threat to the mobile users and their data. The losses incurred are unimaginable, which stretch to the extent of identity theft, financial loss, sensitive information loss, espionage, sabotage, cyber fraud, to mention a few. Android application's permission attributes can be analysed for malware detection using machine learning. However, the high-dimensional permission attributes are the bottleneck in designing optimized malware detection system. Identification of useful permission attributes is an NP-hard problem. Bat Optimization Algorithm for Wrapper-based Feature Selection (BOAWFS) is proposed in this article and evaluated on the CICInves-AndMal2019 benchmark dataset. The performance of BOAWFS is also compared with that of Cuckoo Search Optimization for Wrapper-based Feature Selection (CSOWFS) and Grey Wolf Optimization for Wrapper-based Feature Selection (GOWOFS). Five classifiers, Random Forest (RF), Support Vector Machines (SVMs), K-Nearest Neighbour (KNN), Decision Tree (DT), and Nearest Centroid (NC) are compared for wrapper feature selection. BOAWFS outperformed consistently with all the five classifiers. With 200 agents and 100 iterations, the BOAWFS-DT outperformed with 93.73% accuracy after reducing the features to 518 from 4115. The considerable contribution of BOAWFS is that a 1.67% improvement in accuracy with 87.41% redundancy removal in features is achieved for the very high-dimensional permission-based android malware dataset.

1 | INTRODUCTION

Android Operating System has a share of 74.13% in the worldwide mobile market [1]. Google play store applications (apps) are usually considered as secure and trusted. However, apps that are installed bypassing the Google scan may typically pose a threat. Android malware is a threat vector that can lead to deadly consequences, including theft of credentials from other apps installed in the mobile, credit card data grabbing, password theft, and so forth. Malware is typically relegated as adware, botnets, Trojans, ransomware, scareware, worms, and so forth [2]. Android apps, when installed on our mobile gadget, request approval for few permissions from the user, using which it can perform certain operations using the hardware or software on the mobile [3]. If the user permits knowingly or unknowingly, the malicious apps get permission to use the mobile's assets such as the SMS system, GPS system, camera sensor, contacts, file permissions, including deleting, modifying, creating. There is an urge to study the permissions listed in the Android apps to identify if the app is a malware [4]. Malware can be discovered through static analysis or runtime analysis [5, 6]. Static analysis involves a study on storage media like files, permissions, application or file headers, and so forth, whereas dynamic malware detection tries to discover the malware by observing the runtime behaviour of the device [7]. Several researchers worked on analysing Android malware using the permission data for static analysis using machine learning classifiers. However, there is one major issue with permission extraction, an exhaustive list of permission attributes that make up the dataset, 'the curse of dimensionality'. The huge dimensionality of permission attributes is the bottleneck in static malware classification using permissions. There is an urge to minimize the permission dimensionality by employing learned feature minimization methods.

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2021 The Authors. IET Networks published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.
Feature minimization can be done by two approaches majorly, the filter and the wrapper. In the filter approach, there is no need to ask the classifier about the accuracy, and the values of the dataset attribute itself are enough. The wrapper approach shall make a note of the accuracy of the classifier for every set of attributes selected [8]. There shall be different ways in which a subset of android permission attributes can be chosen from the entire set of permission database, and hence is an NP-hard problem [9]. To minimize the permission attributes and achieve the universally best result in finite time, meta-heuristic search optimization techniques can be chosen. The CICInvesAndMal2019 benchmark dataset [10] is considered, which consists of malicious and benign android permission data samples. Currently, no literature is found in feature optimization on this dataset using bio-inspired wrappers. For feature selection, Bat-inspired global search optimization algorithm is used; furthermore, machine learning classification algorithms are used as wrappers to evaluate and find the globally minimized set of features with improved accuracy. The performance of BOA WFS is also compared with that of Cuckoo Search Optimization for Wrapper-based Feature Selection (CSOWFS), and Grey Wolf Optimization for Wrapper-based Feature Selection (GWO WFS).

2 | RELATED WORK

Aung et al. [11] worked on Crowdroid towards identifying malware in android apps; they have validated with machine learning classifiers. The feature selection algorithm, information gain, was used to determine the most prominent features. Finally, validated using decision trees (DTs), J48, and Random forest (RF), Machine Learning (ML) classifiers, and reported 91.75% and 91.58% accuracies using the RF algorithm. Kakavand et al. [12] focus on Android malware analysis using the Modroid dataset, which consists of 200 Benign and 200 malicious applications. For classification, ML algorithms like Support Vector Machines (SVMs), SMO, and K-Nearest Neighbour (KNN) algorithms were considered and it was observed that K nearest Neighbour algorithm performed better with an accuracy of 80.50%. Taheri et al. [10] proposed the benchmark CICInvesAndMal2019 dataset that consists of two categories, one is static features of permission and Intent, and the other is dynamic API call features with complete 5591 samples assembled from applications of the android. This dataset consists of 5065 data samples of benign apps and 426 samples of malware apps. This data contributed to collecting a total of 4115 permission features. For classification, the ML algorithm, RF, was used. They analysed differently for both categories. For the malware category using static features, the observed precision is 95.3%; for the malware category using dynamic features, accuracy is 83.3%.

Martin et al. [13] used Androidpytool for feature extraction from Android Applications. For experimentation, they used the Omniodroid dataset, which consists of 22,000 samples. For analysis, ML classification algorithms like KNN, RF, DT, and Bagging were used and 89.3% accuracy using the RF algorithm for static analysis and 89.4% accuracy using the Bagging classifier for dynamic analysis, were observed. Taheri et al. [14] analysed the work for malware detection on three datasets called Drebin (118,505 samples), Contagio (1200 samples), and Genome (28,760 samples); these three different types include API calls, Permission, and Intent features. Amid everything, API call feature gives the highest accuracy rates. An RF regression algorithm was used for feature selection. Various classification algorithms like, All Nearest Neighbours (ANN), Weighted All Nearest Neighbours (WANN), K-Medoid-based Nearest Neighbours (KMNN), and First Nearest Neighbours (FNN) were considered for accuracy analysis and it was observed that the WANN classifier performed well with an accuracy rate of 99.46%. Finally, the best reliable outcomes of the Genome dataset are more beneficial than other datasets.

Rana et al. [15] use a substring technique to attribute selection. For experimentation, the Drebin dataset was used, which consists of 11,120 samples of both static and dynamic features. Furthermore, employ four different tree-based learning machines, gradient boosted tree classifier, RF algorithm, DT algorithm, extremely randomized tree classifiers. Compared to other classifiers, RF produces the best accuracy result of 97.24%. Pehlivan et al. [16] worked with the Comodo dataset, which consists of 3784 samples of 2338 benign samples and 1446 malware samples. Four different attribute selections, Gain Ratio, ReliefF, Subset Evaulator, Consistency Subset Evaulator, were used. Towards malware analysis, five ML classifiers like Bayesian, CART, J48, RF, and SMO were considered. For experimentation, WEKA tool was used for both selection algorithms and classification algorithms and exercised to gain the best accuracy results on permission-based analysis. RF performed best with an accuracy of 94.90% by CFS subset Evaulator with 25 features. Zhang et al. [17] used different datasets AMGP (10,000 samples), Drebin dataset (21,883 samples), and In-the-wild dataset (70,000 samples) with various features including permissions, metadata, hardware and application components, and intents in the xml file. AWK and GREP tools are used for feature extraction. The Online Passive-Aggressive (PA) classifier-based model utilized to train dataset samples and also using different permission-based sub-fingerprints for malware detection, that achieves high exactitude rates 99.02% with the Benchmark dataset. Batch learning-based classifier (retrain all samples) for malware attribution to achieve 98.8% accuracy for the Drebin dataset. Varma et al. [18] used Ant Colony Optimization (ACO) for filter-based reduction of PE header malware features using Rough Sets. Cuckoo search is proven efficient for feature selection in the filter method for malware detection [19].

Large-sized attributes are the major issue in the android malware dataset, particularly with permissions. Feature minimization contributes to a great extent in reducing the computational complexity of the android malware detection system. However, as of now, not much work is done in this direction. This article proposes a Bat Optimization Algorithm for Wrapper-based Feature Selection (BOA WFS) to minimize and identify the smallest possible number of permissions that are enough for the successful classification of Android.
malware. Further, the performance of BOAWFS is also compared with CSOWFS and GWOWFS wrappers.

3 | BAT OPTIMIZATION ALGORITHM FOR WRAPPER-BASED FEATURE SELECTION (BOAWFS)

In Bat optimization [20], a group of Bats tries to find their own solution in every iteration and arrive at a globally best solution eventually after the prescribed number of repetitions. In the feature selection problem, a Bat's solution of position \( P_i \) is nothing but an array of 0's and 1's indicating if a particular attribute is included or not. All Bats will be initialized with a solution \( P_i \), fitness value \( F_i \), velocity \( C_i \), loudness \( L_i \), pulse-rate \( T_i \), frequency \( Q_i \).

The fitness plays a very crucial role in converging all Bats to a single solution, that is, the optimal solution. The ultimate target is to find the minimized attributes from the complete set so as to meet the fitness requirements, that obviously must depend on the accuracy of the wrapper classifier used and the length of solution \( l \), total number of full attributes \( u \), and the fitness tuning parameter \( \tau \). The fitness of the selected set of attributes of a Bat is calculated based on two parameters, accuracy and number of features selected as shown in Equation (1). The parameter \( \tau \in [0, 1] \) decides how much weightage is given to accuracy of a solution over the number of features selected while calculating the fitness of that solution. Higher the \( \tau \) value, the more weightage will be given to the accuracy over the number of features selected. Choosing a right \( \tau \) value is crucial for convergence of the Bats.

\[
f(P) = \tau \times accuracy + (1 - \tau) \times \frac{u - S(l)}{u}
\]

In Algorithm 1, \( a \) is the number of Bats, with each one fetching its own solution and run for \( b \) number of iterations. At the very beginning of the algorithm the solution vector \( P_i \), fitness values vector \( F_i \), velocity vector \( C_i \), loudness vector \( L_i \), pulse-rate vector \( T_i \) are initialized. The loudness value of all Bats is initialized randomly with a value between 1 and 2.

Algorithm 1 BOAWFS

Input: No. of Bats, \( a \), no. of iterations, \( b \), constants: \( \tau, \alpha, \gamma, \epsilon \), a dataset with \( x \) attributes.

Output: Minimized attributes as the best solution, \( Sol \), its size, \( |Sol| \), and its accuracy, \( Sol_{acc} \)

1. Initialize \( P_i, F_i, C_i, L_i, T_i, Q_i \) \( \forall i = \{1, 2, \ldots a\} \)
2. for \( \forall i = \{1, 2, \ldots a\} \)
3. \( f_i = AC \)
4. \( L_i = \alpha L_i \)
5. \( T_i = T_i(1 - \exp - \gamma) \)
6. \( f_{\text{max}} = \max(f_i) \)
7. \( \text{index}_{\text{max}} = \text{index}(\max f_i) \)
8. if \( (d < L_i \text{ and } AC > f_i) \)
9. \( f_i = AC \)
10. \( L_i = \alpha L_i \)
11. \( T_i = T_i(1 - \exp - \gamma) \)
12. \( f_{\text{max}} = \max(f_i) \)
13. \( \text{index}_{\text{max}} = \text{index}(\max f_i) \)
14. if \( (f_{\text{max}} > f_{\text{glob}}) \)
15. \( f_{\text{glob}} = f_{\text{max}} \)
16. \( Sol = P_i(\text{index}_{\text{max}}) \)
17. for \( \forall i = \{1, 2, \ldots a\} \)
18. \( \text{rand}_i = [0, 1] \)
19. if \( (\text{rand}_i > T_i) \)
20. for \( \forall j = \{1, 2, \ldots x\} \)
21. \( P_j^i = P_j^i + \pm L_i \) where \( L_i \) is the average Low values of all Bats
22. \( \text{sig} = [0, 1] \)
23. if \( (\text{sig} < \left(1 - \frac{1}{1 + \exp(-\text{rand}_i)})\right) \)
24. \( P_j = 1 \)
25. else
26. \( P_j = 0 \)
27. \( \text{rand}_2 = [0, 1] \)
28. if \( (\text{rand}_2 > L_i \text{ and } f_i < f_{\text{glob}}) \)
29. for \( \forall j = \{1, 2, \ldots x\} \)
30. \( Q_i = Q_{\text{min}} + (Q_{\text{max}} - Q_{\text{min}}) \text{rand}_i \)
31. \( C_i = C_{ij} + (Q_i - P_{ij}) \)
32. \( \text{sig} = [0, 1] \)
33. if \( (\text{sig} < \left(1 - \frac{1}{1 + \exp(-\text{rand}_2)})\right) \)
34. \( P_j^i = 1 \)
35. else
36. \( P_j^i = 0 \)
37. Output \( Sol \), its size, \( |Sol| \), and its accuracy, \( Sol_{acc} \)

The pulse-rate emission of all Bats is initially loaded with a value randomly between 0 and 1. Pulse rate emission is crucial; it decides if a Bat has to generate a new solution or not in an iteration. If the value of pulse-rate emission, \( T_i \), is set to zero, every Bat will generate a new solution and it leads to accelerated convergence simulating the greedy approach. On the other hand, if the pulse-rate emission \( T_i \) is set to a value of one, then, the Bats shall not generate new solutions in the initial few iterations which leads to unwanted delayed convergence. Therefore the pulse-rate emission, \( T_i \) is initialized with a random number between 0 and 1. For every iteration, in the line numbers 5–16 of Algorithm 1, the fitness of all the Bat's solutions (features selected) is calculated, and the best fitness solution is remembered. The main job of lines 17–36 of Algorithm 1 dictates how the Bats move in the search space based on loudness, frequency, and velocity. Two types of movement can be observed in the Bats search. Lines 18–26 performs a local search driven by the loudness parameter of the Bats. Lines 27–36 shows a broader search driven by the frequency and velocity of the Bats. At the beginning of the
algorithm, the fitness values are assigned to the smallest number possible, symbolically represented as $-\infty$. At every iteration, all Bats compute the fitness with the help of the embedded classifier’s accuracy, which makes the features selection process a wrapper selection. When a Bat’s solution is a better one, its loudness value $L_i$ is decreased and its pulse-rate emission $T_i$ is increased as per lines 10 and 11 of the algorithm.

In every iteration, for each Bat a new solution will be generated by moving the Bat according to the equations depicted in lines 20–26 of Algorithm 1. As per the line 19 of Algorithm 1, the value of the pulse rate $T_i$ of the $i$th Bat and a random number $\text{rand}_i$ will decide whether a new solution will be generated by $i$th Bat or not in that particular iteration. Hence it is crucial to initialize the pulse rate vector $T_i$ to a random number between 0 and 1.

Loudness $L_i$, is used to simulate the natural Bat’s loudness, the ultrasonic signal used during hunting. While working in a swarm, each Bat approaches the prey by adjusting its position comparing with the average loudness of all Bats. In an artificial Bat optimization algorithm also the position of a Bat is adjusted in a similar manner. As the solution comes closer, the loudness reduces, and hence it can be said that the step size of the Bat depends on the loudness.

4 | CUCKOO SEARCH OPTIMIZATION FOR WRAPPER-BASED FEATURE SELECTION (CSOWFS)

Yang and Deb, in 2009, designed the CSO algorithm motivated by brood parasitism of a species of bird called the Cuckoo [21]. The CSO is suitable for finding an optimal solution for NP-hard problems that involved combinatorics. Cuckoo [21]. The CSO is suitable for finding an optimal solution for NP-hard problems that involved combinatorics. Hoewever, CSO, in its raw form, is not ideal for feature selection [22]. In a wrapper-based CSO, a binary vector candidate solution, nothing but a nest, represents the inclusion (binary value of 1) and non-inclusion (binary value of 0) of an attribute among a set of $m$-dimensional features. Each attribute can be treated as an egg of a cuckoo. A CSO algorithm for feature selection consists of $n$ nests and $r$ iterations. In the initialization phase, all the nests are initialized randomly, but once the iterations start, the Cuckoo explores the search space by updating the nests using levy flight, as shown in Equation (2).

\[
E_i^{t+1} = E_i^t + \alpha \times \text{Levy}(\lambda) \quad (2)
\]

here, $\alpha$ tunes the step size. Traditionally in CSO, the values generated in a nest are continuous. However, in binary CSO, the sigmoid function is used to convert it to binary, as shown in Equations (3) and (4).

\[
M(E_i^t) = \frac{1}{1 + e^{-E_i^t(\alpha)}} \quad (3)
\]

\[
E_i^{t+1}(x + 1) = \begin{cases} 
1 & M(E_i^t) > 0.5 \\
0 & \text{otherwise}
\end{cases} \quad (4)
\]

At the end of an iteration, few nests are abandoned, and new nests are replaced using Equation (5).

\[
E_i^{t+1} = E_i^t + \delta \times (E_i^t - E^*_i) \quad (5)
\]

where $E_i$ and $E^*_i$ are two randomly picked nests, $\delta = \text{rand}(0, 1)$. The CSO Wrapper Feature Selection (CSOWFS) is shown in Algorithm 2. The first phase is the initialization of nests, in lines 1–6, $n$ nests each with $m$ features (eggs) are initialized by randomly including or not-including a particular feature from the master set. For each nest, fitness is calculated using Equation (1). The iterations start from line 7 in Algorithm 2. Lines 9–15 run for each iteration, where the Cuckoo traverses the search space using levy flight and updates all the nests. At the end of each iteration, as shown in lines 16–17, the global best solution is updated with that of the iteration’s best, based on the fitness. According to the modalities of the CSO, after each iteration, $\rho$ number of nests are abandoned, and they are replaced using Equation (5). After all the iterations are complete global minimal solution attested by the embedded wrapper classifier is returned.

**Algorithm 2 CSOWFS**

**Input:** Data sets $S_T$ for training and $S_E$ for evaluation. Number of features $m$, number of nests $n$, number of iterations $r$, and Tuning parameters $\alpha$, $\rho$, $\sigma$, $\lambda$, $\omega$. The global minimal solution $\text{glob}_{\text{solution}}$ is initialized with full feature set, $\text{Fit}_{\text{glob}}$ is the fitness of full feature dataset.

**Output:** Reduced feature set $\text{glob}_{\text{solution}}$

1. For each nest $E_i (\forall i = 1, 2, 3, \ldots, n)$
2. For each attribute/feature $j (\forall j = 1, 2, 3, \ldots, m)$
3. Initialize $E_i = \text{rand}(0, 1)$
4. $S_T$ and $S_E$ derived from $S_T$ and $S_E$ to contain the features of $E_i$
5. Train the classifier on $S_T$ and find accuracy on $S_E$
6. Evaluate the fitness, $\text{Fit}_i$ using Equation (1)
7. For each iteration $t (\forall t = 1, 2, 3, \ldots, r)$
8. For each nest $E_i (\forall i = 1, 2, 3, \ldots, n)$
9. Built new nest $E_i$ using Equation (2)
10. $S_T$ and $S_E$ derived from $S_T$ and $S_E$ to contain the features of $E_i$
11. Train the classifier on $S_T$ and find accuracy on $S_E$
12. Evaluate the fitness, $\text{Fit}_i$
13. Choose $E_k$ randomly from $n$ nests excluding $E_i$
14. If $\text{Fit}_k > \text{Fit}_i$
15. $E_k$ is replaced with $E_i$ and $\text{Fit}_k = \text{Fit}_i$
16. $\text{glob}_{\text{solution}} = \text{the nest with } \max(\text{Fit})$
17. Abandon $\rho$ nests and replace them using Equation (5)
18. Return $\text{glob}_{\text{solution}}$
5 | GREY WOLF OPTIMIZATION FOR WRAPPER-BASED FEATURE SELECTION (GWOWFS)

Yang Mirjalili et al. [23] introduced yet another nature-inspired optimization approach, the GWO. The algorithm mimics the cooperative teamwork of the Grey Wolf pack in the hunting prey. The pack consists of typically up to 12 wolves, headed by the alpha wolf. There is a social hierarchy in the pack, beta and delta wolves are the next in the hierarchy to the alpha wolf, the other wolves in the pack try to follow the top three wolves in approaching as the prey. For feature selection purposes, a binary GWO is used [24, 25]. During the wolf traversal, a new position value is normally continuous, and it shall be converted to binary with the help of Equations (2) and (3) similar to the CSO. In GWOWFS, $m$ number of wolves run for $r$ number of iterations to find an optimal feature set with the best fitness value. In the beginning, each wolf contains the set of binary values, equal to the number of features, a candidate solution, randomly chosen. As the iteration starts, in each iteration a new position of a wolf $F_i$ is obtained by cross-over, $CO_{Fi}$ of the top three wolves as given in Equation (6)–(15).

$$F_{i}^{+1} = CO_{Fi}(F_{a}, F_{b}, F_{d})$$

(6)

where

$$CO_{Fi}(F_{a}, F_{b}, F_{d}) = \begin{cases} F_{a} & \text{if } \text{rand} < 0.45 \\ F_{b} & \text{if } 0.45 \leq \text{rand} < 0.8 \\ F_{d} & \text{if } \text{rand} \geq 0.8 \end{cases}$$

(7)

$$F_{a} = \begin{cases} 1 & \text{if } F_{i} + CS_{a,F_{i}} - 0.5 \geq \text{rand} \\ 0 & \text{if } F_{i} + CS_{a,F_{i}} - 0.5 < \text{rand} \end{cases}$$

(8)

$$F_{b} = \begin{cases} 1 & \text{if } F_{i} + CS_{b,F_{i}} - 0.5 \geq \text{rand} \\ 0 & \text{if } F_{i} + CS_{b,F_{i}} - 0.5 < \text{rand} \end{cases}$$

(9)

$$F_{d} = \begin{cases} 1 & \text{if } F_{i} + CS_{d,F_{i}} - 0.5 \geq \text{rand} \\ 0 & \text{if } F_{i} + CS_{d,F_{i}} - 0.5 < \text{rand} \end{cases}$$

(10)

$$CS_{f,F_{i}} = \frac{1}{1 + e^{-\left(Ps_{S_{f,F_{i}}} \right)}}$$

(11)

$$S_{f,F_{i}} = (Z * f) - F_{i}$$

(12)

$$\overrightarrow{P} = \overrightarrow{m}(2 * g * \text{rand} - g)$$

(13)

$$\overrightarrow{Z} = \overrightarrow{m}(2 * \text{rand})$$

(14)

$$g = 2 - \frac{\text{itr}_{\text{no}} * 2}{r}$$

(15)

here $g$ is constant, $\text{itr}_{\text{no}}$ is the current iteration number, $r$ is the number of iterations, $\overrightarrow{P}$ and $\overrightarrow{Z}$ are vectors of size $m$ and the number of features. In the GWOWFS algorithm, which is listed in Algorithm 3 there is one tuning variable, $\tau$ that belongs to the fitness function of Equation (1). In lines 1–6 $n$ wolves, each representing a candidate solution with $m$ features is initialized by randomly including or not-including a particular feature from the master set. For each wolf, fitness is calculated using Equation (1). The iterations start from line 7. Lines 9–16 runs for each iteration, where the wolves pack traverse the search space and evaluate the fitness of all the wolves. At the end of the iteration the wolves are ranked and the top three wolves $F_{a}, F_{b}, F_{d}$ are identified, and according to the modalities of GWO the values $\overrightarrow{g}$, $\overrightarrow{P}$, and $\overrightarrow{Z}$ are updated. After every iteration, the minimal global solution is updated.

Algorithm 3 GWOWFS

Input: Data sets $S_{T}$ for training and $S_{E}$ for evaluation. Number of features $m$, number of wolves $n$, number of iterations $r$, and Tuning parameters $\tau$ (Equation (1)), $\text{glob} \text{solution}$

Output: Reduced feature set $\text{glob} \text{solution}$

1. For each wolf, $F_{i}$ $(\forall i = 1, 2, 3, \ldots, n)$
2. For each attribute/feature $j$ $(\forall j = 1, 2, 3, \ldots, m)$
3. Initialize $F_{j}^{i} = \text{rand}(0, 1)$
4. $S_{T}$ and $S_{E}$ derived from $S_{T}$ and $S_{E}$, to contains the features $F_{i}$
5. Train the classifier on $S_{T}$ and find accuracy on $S_{E}$
6. Evaluate the fitness, $Fit_{i}$ and rank the wolves, update $F_{a}, F_{b}, F_{d}$ wolves
7. For each iteration $(\forall t = 1, 2, 3, \ldots, r)$
8. For each wolf $F_{i}$ $(\forall i = 1, 2, 3, \ldots, n)$
9. Update the wolf position $F_{i}$ using Equation (6)
10. $S_{T}$ and $S_{E}$ derived from $S_{T}$ and $S_{E}$, to contains the features $F_{i}$
11. Train the classifier on $S_{T}$ and find accuracy on $S_{E}$
12. Evaluate the fitness, $Fit_{i}$ of updated wolf $F_{i}$ using Equation (1)
13. Rank the wolves of this iteration and update $F_{a}, F_{b}, F_{d}$ wolves
14. Renew the values $g$, $\overrightarrow{P}$, and $\overrightarrow{Z}$
15. if($Fit_{\text{gb}} > Fit_{\text{glob}}$)
16. $\text{glob} \text{solution} = F_{a}$
17. Return $\text{glob} \text{solution}$

6 | RESULTS AND DISCUSSION

CICInvesAndMal2019 [10], the Android Malware dataset with very high dimensional, 4115, Android permissions, and a binary class label, is used to evaluate BOAWFS. The dataset consists of 1187 benign and 407 malware application samples that are extracted from 42 unique malware families grouped into four categories such as Adware, Scareware, Ransomware, and SMS malware. Benign application permissions are derived from play store applications spanned from 2012 to 2019
randomly. It holds a total of 1594 samples that carry a complete 8115 intents and permission attributes. However, we consider only 4115 permission attributes for malware detection. The BOAWFS, the CSOWFS, and the G WOWFS algorithms are developed in Python language, and the performance is tested with various classifier wrappers using the scikit learn library. The trials are run on a computer with Intel Core i7 that has 8 GB of RAM. After several trials and errors, five classifiers are used as wrappers for selecting the features towards optimization of the android malware detection system. The classifiers considered for wrapper feature selection using BOAWFS, CSOWFS, and G WOWFS are RF, SVMs, KNN, DT, and Nearest Centroid (NC). The performance of BOAWFS is compared with two more nature-inspired search algorithms, the CSOWFS [21, 22], and the G WOWFS [23–25].

Several experiments are performed to evaluate and compare the performance of the three wrapper feature selection algorithms. Table 1 renders the results of the BOAWFS, CSOWFS, and the G WOWFS, each evaluated with five different classifier wrappers. For a consistent comparison in a limited time, all the three search methods are run with 50 agents and 50 iterations. All the experiments are run five times, and the average results are given. An average of five runs is taken because of a random split of training and testing data each time. Experiment 1 of Table 1 tabulates the results of the three wrapper FS search methods evaluated on the RF classifier. RF produced the best accuracy before the process of feature reduction. BOAWFS-RF performed well with 853 selected attributes, showing a 79.27% reduction, and 94.36% accuracy post-reduction, a +2.3% improvement. G WOWFS-RF is faster compared to BOAWFS-RF, but with 1064 selected features and slightly lesser accuracy. In the case of SVM wrapper, as shown in Experiment 2 of Table 1, BOAWFS-SVM outperformed CSO and GWO counterparts with 600 selected features. However, the accuracy is not satisfactory. In Experiment 3, BOAWFS-KNN topped CSO and GWO with a good accuracy of 94.78% post-reduction, but 1012 features are selected. In Experiment 4 BOAWFS-DT wrapper’s performance is satisfactory with a +3.13% improvement in accuracy and 81.24% reduction of features. Another significant performance advantage is the run time of BOAWFS-DT, which is 96% faster compared to BOAWFS-KNN, and 95.6% faster than BOAWFS-SVM, and 92.7% faster than BOAWFS-RF. In Experiment 5, BOAWFS-NC, a simple and faster classifier, is also compared. However, the accuracy after reduction is not satisfactory even though the number of features reduced is satisfactory.

To summarize the results with 50 agents and 50 iterations, with all the five classifiers that are tested, Bat Optimization outperformed CSO and GWO. Among the five classifiers, undoubtedly DT classifier did well. Figure 1 graphically show the results for a special understanding. Consistent 50 agents and 50 iterations are used for testing to narrow down to one classifier, and it is the DT.

Furthermore, few more experiments are conducted with increased agents and iterations because all the three search techniques are random selection and random walk oriented methods that typically give better results with a high number of agents and iterations. However, due to run time constraints, we cannot experiment with all the classifiers; therefore, the search space is further extended with more agents and increased iterations with DT classifier, whose results are tabulated in Table 2 and shown graphically in Figure 2. In

| Experiment No. | Wrapper Classifier Used with BOAWFS | Accuracy Rate before Attribute Selection | Accuracy Rate after Attribute Selection | % Gain in the Accuracy | Features Selected (out of 4115) | % Reduction of Features | Total Run Time in Sec (50 Agents, 50 Iterations) |
|----------------|------------------------------------|------------------------------------------|----------------------------------------|------------------------|--------------------------------|-------------------------|---------------------------------------------|
| 1              | BOAWFS-RF                          | 92.06%                                   | 94.36%                                 | +2.30%                 | 853                            | 79.27%                  | 2085                                        |
|                | CSOWFS-RF                          | 92.27%                                   | −0.42%                                 |                        | 1064                           | 74.14%                  | 1771                                        |
|                | G WOWFS-RF                         | 91.64%                                   | −0.63%                                 |                        | 1061                           | 74.21%                  | 4056                                        |
| 2              | BOAWFS-SVM                         | 90.60%                                   | 91.44%                                 | +0.84%                 | 600                            | 85.42%                  | 3449                                        |
|                | CSOWFS-SVM                         | 91.23%                                   | +0.63%                                 |                        | 1678                           | 59.22%                  | 8730                                        |
|                | G WOWFS-SVM                        | 89.97%                                   | −0.63%                                 |                        | 1061                           | 74.39%                  | 4406                                        |
| 3              | BOAWFS-KNN                         | 91.23%                                   | 94.78%                                 | +3.54%                 | 1012                           | 75.41%                  | 4042                                        |
|                | CSOWFS-KNN                         | 92.90%                                   | +1.06%                                 |                        | 1691                           | 58.90%                  | 7743                                        |
|                | G WOWFS-KNN                        | 92.06%                                   | +0.83%                                 |                        | 1026                           | 75.06%                  | 3311                                        |
| 4              | BOAWFS-DT                          | 90.60%                                   | 93.73%                                 | +3.13%                 | 772                            | 81.24%                  | 151                                         |
|                | CSOWFS-DT                          | 92.27%                                   | +1.67%                                 |                        | 1575                           | 61.72%                  | 966                                         |
|                | G WOWFS-DT                         | 91.65%                                   | +1.05%                                 |                        | 1075                           | 73.87%                  | 158                                         |
| 5              | BOAWFS-NC                          | 84.55%                                   | 88.10%                                 | +3.45%                 | 801                            | 80.53%                  | 129                                         |
|                | CSOWFS-NC                          | 86.84%                                   | +2.29%                                 |                        | 1518                           | 63.11%                  | 1036                                        |
|                | G WOWFS-NC                         | 87.47%                                   | +2.92%                                 |                        | 1036                           | 74.82%                  | 122                                         |
Experiment 6, the iterations are 50, and BOAWFS-DT is run with 100 Bats, CSOWFS-DT is run with 100 nests, and GWOWFS-DT is run with 100 wolves. Again BOAWFS has given the best results with an 82.16% reduction in features. Experiment 7, the iterations are further increased to 100, and the agents are further increased to 200. The output is quite evident that increased iterations and agents shall produce even better results. BOAWFS-DT selected 518 features, an 87.41% reduction, with an accuracy post-reduction of 93.73%, a +1.67% improvement, which is an outstanding achievement. The noise and redundant data are eliminated by the process of increased agents and iterations, and therefore, the accuracy is also enhanced. It is worth looking at the Experiment 8, where the performance of a simpler and faster NC classifier wrapper, the BOAWFS-NC produced the least number of features, 375, a 90.88% reduction, with an improvement of +6.47% in post-reduction accuracy, but the accuracy just did not cross 90%.

In summary, by increasing the agents and iterations, the search space is better explored, eliminating the noisy features. BOAWFS-DT is the winner with 518 selected features and 93.73% average accuracy. The best run accuracy was recorded to be 95.92% after feature reduction. This article discloses the selected features so that in the future, any researchers can test and further try to improve the performances. Table 3 list the 518 feature numbers produced by Experiment 7, of Table 2, for BOAWFS-DT, and Table 4 lists the 375 features built by BOAWFS-NC from Experiment. Further, in the literature, feature selection on the Android permission features of the CICInvesAndMal2019 [10] dataset is not found to compare the performance of feature selection; however, the classification accuracies of similar works are compared in Table 5.

Figures 3–7 are the ROC curve, a visual performance indicator of the five classifiers under consideration. The ROC curves are generated from a single run as opposed to the

| Experiment No. | Wrapper Classifier Used with BOAWFS | Agents and Iterations | Accuracy Rate Before Attribute selection | Accuracy Rate After Attribute Selection | % Gain in the Accuracy | Features Selected (out of 4115) | % Reduction of Features | Total Run Time in Sec |
|----------------|------------------------------------|-----------------------|------------------------------------------|----------------------------------------|------------------------|--------------------------------|------------------------|----------------------|
| 6              | BOAWFS-DT                          | 100 & 50              | 91.85%                                   | 93.11%                                 | +1.25%                 | 734                            | 82.16%                 | 322                  |
|                | CSOWFS-DT                          |                       |                                          | 92.48%                                 | +0.63%                 | 1353                           | 69.55%                 | 2048                 |
|                | GWOWFS-DT                          |                       |                                          | 92.69%                                 | +0.84%                 | 1028                           | 75.02%                 | 334                  |
| 7              | BOAWFS-RF                          | 200 & 100             | 92.06%                                   | 93.73%                                 | +1.67%                 | 518                            | 87.41%                 | 1025                 |
|                | CSOWFS-RF                          |                       |                                          | 93.73%                                 | +1.67%                 | 1214                           | 70.49%                 | 5965                 |
|                | GWOWFS-RF                          |                       |                                          | 92.90%                                 | +0.84%                 | 986                            | 76.26%                 | 948                  |
| 8              | BOAWFS-NC                          | 200 & 100             | 82.67%                                   | 89.14%                                 | +6.47%                 | 375                            | 90.88%                 | 861                  |
|                | CSOWFS-NC                          |                       |                                          | 88.94%                                 | +6.27%                 | 1198                           | 70.88%                 | 8956                 |
|                | GWOWFS-NC                          |                       |                                          | 87.47%                                 | 4.80%                  | 1025                           | 76.06%                 | 809                  |

In summary, by increasing the agents and iterations, the search space is better explored, eliminating the noisy features. BOAWFS-DT is the winner with 518 selected features and 93.73% average accuracy. The best run accuracy was recorded to be 95.92% after feature reduction. This article discloses the selected features so that in the future, any researchers can test and further try to improve the performances. Table 3 list the 518 feature numbers produced by Experiment 7, of Table 2, for BOAWFS-DT, and Table 4 lists the 375 features built by BOAWFS-NC from Experiment. Further, in the literature, feature selection on the Android permission features of the CICInvesAndMal2019 [10] dataset is not found to compare the performance of feature selection; however, the classification accuracies of similar works are compared in Table 5.

Figures 3–7 are the ROC curve, a visual performance indicator of the five classifiers under consideration. The ROC curves are generated from a single run as opposed to the
accuracies of Tables 1 and 2, which are average of five runs. It can be clearly noticed that the ROC curve (solid plots) for the classifier on a reduced feature set has a greater area under the curve. This clearly demonstrates the performance of the Bat Optimization Wrapper Feature Selection.

7 | CONCLUSION

Analysis of permission attributes of Android Mobile applications through the ML approach plays a crucial role in mobile malware detection. The Android permissions are evolving rapidly with very high dimensionality. Efficient feature reduction of the high-dimensional Android permissions dataset is a potential area of research. The very recent benchmark dataset for Android application malware identification using the permission database is the CICInvesAndMal2019. This dataset has 4115 permissions, which is considered very high-dimensional, is a bottle neck for time, space, and computational complexity of malware detection system. This article explored three widespread nature-inspired searching tools when embedded with wrapper classifiers produce efficient results. Bat Optimization, Cuckoo Search, and Grey Wolf Optimization are used in a wrapper selection mode tested on five different classifiers, RF, SVM, KNN, DT, and NC. The BOAWFS, CSOWFS, and 

**FIGURE 2** List of attributes selected by BOAWFS-DT (Experiment 7, Table 2)

**TABLE 3** List of 518 attributes selected by BOAWFS-DT (Experiment 7, Table 2)

**TABLE 4** List of 375 attributes selected by BOAWFS-NC (Experiment 8, Table 2)
GVOWFS are applied on the 4115 permission attributes of the CICInvesAndMal2019 benchmark dataset. Firstly, with 50 agents and 50 iterations, performance evaluation is done to identify the most promising search algorithm and classifier combination. BOAWFS-DT is found to be the best. Later, with a higher number of agents and iterations, the search space of BOAWFS-DT is explored, and the 4115 features are reduced to 518 features and 93.73% average accuracy. The best run accuracy was recorded to be 95.92% after feature reduction. The Bat Optimization clearly won over the Cuckoo search and Grey

| Paper, Year | Accuracy | Classifier   |
|-------------|----------|--------------|
| [26], 2019  | 95%      | Random forest|
| [10], 2019  | 95.3%    | Random forest|
| [27], 2016  | 95.63%   | Functional tree|
| [28], 2017  | 91.97%   | SVM          |
| [29], 2020  | 93.8%    | J48          |
| This paper  | 95.92%   | DT (CART)    |

**FIGURE 3** ROC curve for RF classifier

**FIGURE 4** ROC curve for SVM classifier

**FIGURE 5** ROC curve for DT classifier

**FIGURE 6** ROC curve for KNN classifier

**FIGURE 7** ROC curve for NC classifier
wolf wrapper feature selection with respect to the high-dimen-
sional Android malware permission dataset.

Mobile cloud computing is becoming increasingly popular. It allows mobile users to run resource-intensive mobile appli-
cations in the cloud environment rather than in mobiles [30, 31]. Using cloud computing for mobile application running raises concern about user data security [32]. On the other hand, the malicious users also pose a critical threat to such cloud services. The proposed BOAWFS-DT can be used efficiently in a cloud environment to classify malicious apps as well as ma-
licious users, and out future work shall be carried over in this
direction. Cloud applications overwhelm the dataset sizes, and it becomes computationally infeasible to process on an in-
dividual system. Distributed processing platforms like the
Hadoop environment shall be considered in the future work to
handle big data sizes of malware applications.

ORCID
Ravi Kiran Varma P  https://orcid.org/0000-0002-4989-1305
S Kumar Reddy Mallidi  https://orcid.org/0000-0001-5908-8316

REFERENCES

1. O’dea, S.: Market share of mobile operating systems worldwide 2012–
2019. Statista [Online]. (2020). https://www.statista.com/statistics/
272698/
2. Qamar, A., Karim, A., Chang, V.: Mobile malware attacks: review, taxon-
omy & future directions. Future Gener. Comput. Syst. 97, 887–909 (2019)
3. Biswas, S., Haipeng, W., Rashid, J.: Android permissions management at
app installing. Int. J. Secur. Appl. 10, 223–232 (2016)
4. Prajakt, D.S., Gadicha, A.B.: Analysis of malware detection tech-
niques in android. Int. J. Comput. Sci. Mob. Comput. 3(3), 176–182
(2014)
5. Angelob, G.D., Ficco, M., Palmieri, F.: Malware detection in mobile
environments based on autoencoders and API-images. J. Parallel Distrib.
Comput. 137, 26–33 (2020)
6. Ahmad, M., et al.: Stadatt: addressing the problem of dynamic code
updates in the security analysis of android applications. J. Syst. Software.
159(110386), 1–13 (2020)
7. Schmidt, A.D., et al.: Static analysis of executables for collaborative
malware detection on android. Proc. 2009 IEEE Int. Conf. Communica-
tions, Dresden, Germany, pp. 1–5 (2009)
8. Zhang, Y., et al.: Feature selection algorithm based on barebones parti-
cle swarm optimization. Neurocomputing, 148, 150–157 (2015)
9. Varma, P.R.K., Kumari, V.V., Kumar, S.S.: Feature selection using relative
fuzzy entropy and ant colony optimization applied to real-time intrusion
detection system. Procedia Comput. Sci. 85, 503–510 (2016)
10. Taheri, L., Kadir, A.F.A., Lashkari, A.H.: Extensible android malware
detection and family classification using network-flows and API-calls. In:
Proc. 2019 International Carnahan Conference on Security Technology
(ICCST), Chennai, pp. 1–8 (2019)
11. Aung, Z., Zaw, W.: Permission-based android malware detection. Int. J.
Sci. Tech. Res. 2(3), 228–234 (2013)
12. Kakavand, M., Dabghah, M., Dehghanban, A.: Application of ma-
chine learning algorithms for android malware detection. Proc. Int.
Conf. t32-36
13. Martin, A., Lara-Cabrera, R., Camacho, D.: Android malware detection
through hybrid features fusion and ensemble classifiers: the AndroPyTool
framework and the OmniDroid dataset. Inform. Fusion. 52, 128–142 (2019)
14. Taheri, R., et al.: Similarity-based android malware detection using
hamming distance of static binary features. Future Gener. Comput. Syst.
105, 230–247 (2020)
15. Rana, M.S., Rahman, S.S.M.M., Sung, A.H.: Evaluation of tree based
machine learning classifiers for android malware detection. Lect. Notes
Comput. Sci. 11056, 377–385 (2018)
16. Pehlivan, U., et al.: The analysis of feature selection methods and clas-
sification algorithms in permission based android malware detection.
In: Proc. 2014 IEEE Symposium on Computational Intelligence in Cyber
Security (CICS), Orlando, pp. 1–8 (2014)
17. Zhang, L., Thing, V.I.L., Cheng, Y.: A scalable and extensible framework
for android malware detection and family attribution. Comput. Secur. 80,
120–133 (2019)
18. Varma, P.R.K., Akhila, K., Mallidi, S.K.R.: Feature reduction and opti-
mization of malware detection system using ant colony optimization and
rough sets. Int. J. Inf. Secur. Privacy. 14(3), 95–114 (2020)
19. Varma, P.R.K., et al.: Feature selection and performance improvement of
malware detection system using cuckoo search optimization and rough
sets. Int. J. Adv. Comput. Sci. Appl. 11(5), 708–714 (2020)
20. Nakamura, R.Y.M., et al.: BBA: a binary bat algorithm for feature se-
lection. Proc. 2012 25th SIBGRAPI Conf. Graphics, Patterns and Images,
Ouro Preto, pp. 291–297 (2012)
21. Yang, X.S., Deb, S.: Engineering optimisation by cuckoo search. Int. J.
Math. Modell. Numer. Optim. 1(4), 330–345 (2010)
22. Aziz, M., Hassani, A.: Modified cuckoo search algorithm with rough
sets for feature selection. Neural Comput. Appl. 29, 925–934 (2018)
23. Mirjalili, S., Mirjalili, S.M., Lewis, A.: Grey wolf optimizer. Adv Eng
Software. 69, 46–61 (2014)
24. Emary, E., Zawbaa, H., Hassani, A.: Binary grey wolf optimization
approaches for feature selection. Neurocomputing. 172, 371–381
(2016)
25. Al-Tashi, Q., et al.: Binary optimization using hybrid Grey wolf optimi-
ization for feature selection. IEEE Access. 7, 39496–39508 (2019)
26. Chen, R., Li, Y., Fang, W.: Android malware identification based on traffic
analysis. Lect. Notes Comput. Sci. 11632, 293–303 (2019)
27. Sun, L., et al.: SigPID: significant permission identification for android
malware detection. In: Proc. 11th International Conference on Malicious
and Unwanted Software (MALWARE), Fajardo, pp. 1–8 (2016)
28. Li, J., et al.: Significant permission identification for machine learning based
android malware detection. IEEE Trans. Ind. Inf. 14(7), 3216–3225 (2017)
29. Jiang, X., et al.: Android malware detection using fine-grained features.
Sci. Program. 1–13 (2020)
30. Khan, A.U.R., et al.: A survey of mobile cloud computing application
models. IEEE Commun. Surv. Tutorials. 16(1), 393–413 (2014)
31. Alanezi, F.T., Salama, D., Hosny, K.: An efficient framework for mobile
cloud computing. In: Proc. 32nd IBIMA Conference, Seville, pp. 5783–
5796 (2018)
32. Sahu, D., et al.: Cloud computing in mobile applications. Int. J. Sci. Res.
Publ. 2(8), 1–9 (2012)

How to cite this article: P RKV, Mallidi SKR, K SJ, D PL. Bat optimization algorithm for wrapper-based
feature selection and performance improvement of android malware detection. IET Netw.
2021;10:131–140. https://doi.org/10.1049/ntw.2.12022