Received December 15, 2017, accepted January 16, 2018, date of publication March 6, 2018, date of current version March 15, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2801792

A Bi-objective Hyper-Heuristic Support Vector Machines for Big Data Cyber-Security

NASSER R. SABAR[1], XUN YI[2], AND ANDY SONG[2]

[1]Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC 3086, Australia

[2]School of Computer Science and Information Technology, RMIT University, Melbourne, VIC 3000, Australia

Corresponding author: Nasser R. Sabar (n.sabar@latrobe.edu.au)

ABSTRACT Cyber security in the context of big data is known to be a critical problem and presents a great challenge to the research community. Machine learning algorithms have been suggested as candidates for handling big data security problems. Among these algorithms, support vector machines (SVMs) have achieved remarkable success on various classification problems. However, to establish an effective SVM, the user needs to define the proper SVM configuration in advance, which is a challenging task that requires expert knowledge and a large amount of manual effort for trial and error. In this paper, we formulate the SVM configuration process as a bi-objective optimization problem in which accuracy and model complexity are considered as two conflicting objectives. We propose a novel hyper-heuristic framework for bi-objective optimization that is independent of the problem domain. This is the first time that a hyper-heuristic has been developed for this problem. The proposed hyper-heuristic framework consists of a high-level strategy and low-level heuristics. The high-level strategy uses the search performance to control the selection of which low-level heuristic should be used to generate a new SVM configuration. The low-level heuristics each use different rules to effectively explore the SVM configuration search space. To address bi-objective optimization, the proposed framework adaptively integrates the strengths of decomposition- and Pareto-based approaches to approximate the Pareto set of SVM configurations. The effectiveness of the proposed framework has been evaluated on two cyber security problems: Microsoft malware big data classification and anomaly intrusion detection. The obtained results demonstrate that the proposed framework is very effective, if not superior, compared with its counterparts and other algorithms.

INDEX TERMS Hyper-heuristics, big data, cyber security, optimisation.

I. INTRODUCTION

The rapid advancements in technologies and networkings such as mobile, social and Internet of Things create massive amounts of digital information. In this context, the term *big data* has been emerged to describe this massive amounts of digital information. Big data refers to large and complex datasets containing both structured and unstructured data generated on a daily basis, and need to be analysed in short periods of time [49]. The term big data is different from the big database, where big data indicates the data is too big, too fast, or too hard for existing tools to handle. Big data is commonly described by three characteristics: volume, variety and velocity (aka 3Vs). The 3Vs define properties or dimensions of data where volume refers to an extreme size of data, variety indicates the data was generated from divers sources and velocity refers to the speed of data creation, streaming and aggregation [49]. The complexity and challenge of big data are mainly due to the expansion of all three characteristics (3Vs)- rather than just the volume alone [14]. Learning from big data allows researchers, analysts, and organisations users to make better and faster decisions to enhance their operations and quality of life [38]. Given its practical applications and challenges, this field has attracted the attention of researchers and practitioners from various communities, including academia, industry and government agencies [14].

However, big data created a new issue related not only to the 3Vs characteristics, but also to data security. It has been indicated that big data does not only increase the scale of the challenges related to security, but also create new and different cyber-security threats that need to be addressed in an effective and intelligent ways. Indeed, security is known as the prime concern for any organisation when learning from big data [47]. Examples of big data cyber-security challenges are malwares detection, authentications and steganoanalysis [45]. Among these challenges, malware detection is the most critical challenge in big data cyber-security. The term malware (short for ***mal***icious soft***ware***) refers to various malicious computer programs such as ransomwares, viruses and scarewares that can infect computers and release important information via networks, email or websites [53]. Researchers and organisations acknowledged the issues that can be caused by these dangerous software (malicious computer programs) and therefore new methods should be developed to prevent them. Yet, despite the fact that malware is a crucial issue in big data, very little researches have been done in this area [47]. Examples of malware detection methods include signature-based detection methods [22], behaviors monitoring detection methods [54] and patterns-based detection methods [19], [53]. However, most of existing malware detection methods are mainly proposed to deal with small-scale datasets and unable to handle big data within a moderate amount of time. In addition, these methods can be easily evaded by attackers, very costly to maintain and they have very law success rates [53].

To address the above issues, machine learning (ML) algorithms have been proposed for classifying unknown patterns and malicious software [45], [53]. ML have showing promising results to classify and identify unknown malware software. Support vector machines (SVMs) are among the most popular ML algorithms and have shown remarkable success in various real-world applications [15]. The popularity of SVMs is due to their strong performance and scalability [40]. However, despite these advantages, the performance of an SVM is strongly affected by its selected configuration [9]. A typical SVM configuration includes the selection of the soft margin parameter (or penalty) and the kernel type as well as its parameters. In the literature, various methodologies have been developed for selecting SVM configurations. These methodologies can be classified based on the formulation of the SVM configuration problem and the optimisation method used [9], [12]. An SVM configuration formulation can rely on either a single criterion, in which case k-fold cross-validation is used to assess the performance of the generated configuration, or multiple criteria, in which case more than one criterion must be used to evaluate the generated configuration, such as the model accuracy and model complexity [46]. The available optimisation methods include grid search methods, gradient-based methods and meta-heuristic methods. Grid search methods are easy to implement and have shown good results [13]. However, they are computationally expensive, which limits their applicability to big data problems. Gradient-based methods are very efficient, but their main shortcomings are that they require the objective function to be differentiable and that they strongly depend on the initial point [4]. Meta-heuristic methods have been suggested to overcome the drawbacks of grid search methods and gradient-based methods [5], [28], [56]. However, the performance of a meta-heuristic method strongly depends on the selected parameters and operators, the selection of which is known to be a very difficult and time-consuming process. In addition, only one kernel is used in most works, and the search is performed over the parameter space of that kernel.

This work presents a novel bi-objective hyper-heuristic framework for SVM configuration optimisation. Hyper-heuristics are more effective than other methods because they are independent of the particular task at hand and can often obtain highly competitive configurations. Our proposed hyper-heuristic framework integrates several key components that differentiate it from existing works to find an effective SVM configuration for big data cyber security. First, the framework considers a bi-objective formulation of the SVM configuration problem, in which the accuracy and model complexity are treated as two conflicting objectives. Second, the framework controls the selection of both the kernel type and kernel parameters as well as the soft margin parameter. Third, the hyper-heuristic framework combines the strengths of decomposition- and Pareto-based approaches in an adaptive manner to find an approximate Pareto set of SVM configurations.

The performance of the proposed framework is validated and compared with that of state-of-the-art algorithms on two cyber security problems: Microsoft malware big data classification and anomaly intrusion detection. The empirical results fully demonstrate the effectiveness of the proposed framework on both problems.

The remainder of this paper is organised as follows. In the next section (Section II), we present a brief overview of related work. The definition and formulation of SVMs are presented in Section III. In Section IV, we describe the proposed hyper-heuristic framework and its main components. In Section V, we discuss the experimental setup, including the benchmark instances and the parameter settings of the proposed framework. In Section VI, we provide the computational results of our framework and compare the framework with other algorithms. Finally, the conclusion of this paper is presented in Section VII.

II. RELATED WORK

In this section, we briefly discuss some related works on malware detection methods and meta-learning methods. It also includes review on hyper-heuristics for classification problems.

A. MALWARE DETECTION METHODS

Recent survey by Ye *et al.* [53] classified malware detection methods into three types: signature-based detection methods, patterns-based detection methods and cloud-based detection methods. Most of existing detection methods use signature to detect malware software [21], [22]. Signature is a unique short string of bytes defined for each known malware software so it can be used to detect future unknown software [22]. Although signature-based detection methods are able to detect malware software, they require constant updating to include the signature of new malware software into the signature database. In addition, they can be easily evaded by malware developers by using encryption, polymorphism or obfuscation [53]. Furthermore, signature database is usually created via manual process by domain experts which is known as tedious task and time-consuming [16].

Patterns-based detection methods check whether a given malware software contains a set of patterns or not. The patterns are extracted by domain experts to distinguish malware software and non-benign files [2], [10], [35]. However, the analysis of malware software and the extraction of patterns by domain experts is subject to error-prone and requires a huge amount of time [19]. This indicates that manual analysis and extraction are major issues in developing patterns-based detection methods because malware software grows very fast [53].

Cloud-based detection methods use a server to store detection software so malware detection can be done in a client-server manner using cloud-based architecture [41], [53], [54]. However, cloud-based detection methods are highly affected by the available number of cluster nodes and the running time of the detection methods [29]. This can slow down the detection processes and thus multiable malware software can not be easily detected.

Generally speaking, due to the economic benefits, malware software getting increasingly complex and malware developers employ automated malware development tool-kits to write and modify malware codes to evade detection methods [44]. In addition, existing methods are not scalable enough to deal with big data and less responsive to new threats due to the quickly changing nature of malware software. Machine learning (ML) algorithms have been suggested to be used as malware detection methods to automatically detect malware software [53]. However, designing an effective detection method using machine learning algorithm is a challenging task due to the large number of possible design options and the lack of intelligent way for how to choose and/or combine existing options. This work addresses these challenges through proposing a hyper-heuristic framework to search the space of the design options and their values, and iteratively combine and adapt different options for different problem instances.

B. META-LEARNING APPROACHES

A traditional SVM has several tunable parameters that need to be optimised in order to obtain high quality results [9]. Meta-learning approaches have been widely used to find the best combination of parameters and their values for SVM. Meta-learning is an approach that aims at understanding the problem characteristics and the best algorithm that fit to it [52]. In particular, it tries to discover or learn which problem features contribute to algorithm performance and then recommend the appropriate algorithm for that problem. Soares *et al.* [43] proposed a meta-learning approach to find the parameter values of Gaussian kernel for SVM to solve regression problems. The authors used K-NN as a ranking method to select the best value for the kernel width parameter. Reif *et al.* [36] hybridised meta-learning and case-based reasoning to generate the initial starting solutions for genetic algorithm. The proposed genetic algorithm is used to find appropriate parameter values for a given classifier to solve a given problem instance. Ali and Smith-Miles [3] employed a meta-learning approach that uses classical, distance and distribution statistical information to recommend kernel method for SVM. Gomes *et al.* [24] proposed a hybrid method that combines meta-learning and search algorithms to select SVM parameter values. Other examples that use meta-learning approaches to tune SVMs are [30]–[32] and [37].

Although meta-learning approaches have shown to be effective in tuning SVMs parameter values, they still face the problem of over-fitting. This is because the extracted problem features only capture the instances that have been used during the training process. In addition, most of existing approaches are used to tune single kernel method and were tested on small scale instances. Our proposed framework uses kernel methods and the selection process is formulated as a bi-objective optimisation to effectively deal with big data problems.

C. HYPER-HEURISTICS

Hyper-heuristic is an emergent search method that seeks to automate the process of combining or generating an effective problem solver [11]. A traditional hyper-heuristic framework takes all possible designing options as an input and then decides which one should be used. The output of a hyper-heuristic framework is a problem solver rather than a solution [39]. Sim *et al.* [42] proposed a hyper-heuristic framework to generate a set of attributes that characterise a given instance for one dimensional bin packing problem. The authors used hyper-heuristic framework to predict which heuristic should be used to solve the current problem instance. Ortiz-Bayliss *et al.* [34] proposed a learning vector quantization neural network based hyper-heuristic framework for solving constraint satisfaction problems. The hyper-heuristic framework was trained to decide which heuristic to select based on the given properties of the instance at hand. Greer [25] presented a stochastic hyper-heuristic framework for unsupervised matching of partial information. The hyper-heuristic framework was implemented as a feature selection method to determine which sub-set of features should be selected. Basgalupp [7] proposed a hyper-heuristic framework to evolve decision-tree for software effort prediction. Other examples that use hyper-heuristic frameworks to evolve classifiers are [6], [8], and [51].

III. PROBLEM DESCRIPTION

This section is divided into three subsections. We first describe the SVM process, followed by the formulation of the configuration problem. Finally, we present the proposed multi-objective formulation of the SVM configuration problem.

A. SUPPORT VECTOR MACHINES

SVMs are a class of supervised learning models that have been widely used for classification and regression [50].

TABLE 1. Kernel functions.

Name	Formula
Radial	$K(x,x_i) = \exp(-\alpha\|x,x_i\|^2)$
Polynomial	$K(x,x_i) = (\alpha(x.x_i)+\beta)^d$
Sigmoidal	$K(x,x_i) = tanh(\alpha(x.x_i)+\beta)$
ANOVA	$K(x,x_i) = \left(\sum_i \exp((\alpha(x-x_i))^2\right)^d$
Inverse multi-quadratic	$K(x,x_i) = 1/\sqrt{\|x,x_i\|^2+\beta}$

SVMs are based on statistical learning theory and are better able to avoid local optima than other classification algorithms. An SVM is a kernel-based learning algorithms that seeks the optimal hyperplane. The kernel learning process maps the input patterns into a higher-dimensional feature space in which linear separation is feasible. Suppose that we have $\mathcal{L}$ sample sets $\{(x_i,y_i) \mid x_i \in R^v, y_i \in R\}$, where x_i is an input vector of dimensionality v and y_i is the output vector corresponding to x_i. The basic idea of the SVM approach is to map the input vector x_i into an $\mathbb{N}$-dimensional feature space and then construct the optimal decision-making function in the feature space as follows [50]:

$$min(\frac{1}{2} \parallel \omega \parallel + C\sum_{i=1}^{\mathcal{L}}(\xi_i + \xi_i^*)) \tag{1}$$

s.t.

$$\begin{aligned} &y_i - f(x_i) \leq \varepsilon + \xi_i \\ &f(x_i) - y_i \leq \varepsilon + \xi_i^* \\ &\xi_i, \quad \xi_i^* \geq 0, \; i = 1, 2, \ldots \mathcal{L} \end{aligned}$$

where $\omega = (\omega_1, \omega_1, \omega_1, \ldots, \omega_{\mathbb{N}})^T$ is the weight vector; C is the margin parameter (or penalty); ε is the insensitive loss coefficient, which controls the number of support vectors; and ξ_i and ξ_i^* are two slack variables, which take non-negative values. Equation (1) can be transformed into a dual problem, in which the optimal decision can be obtained by solving the following:

$$f(x) = \sum_{i=1}^{\mathcal{L}}(\alpha_i - \alpha_i^*)K(x, x_i) + b \tag{2}$$

where α_i and α_i^* are Lagrange coefficients representing the two slack variables, $b \in$ R is the bias, and $K(x, x_i)$ is the the kernel function

$$K(x, x_i) = \langle \Phi(x), \Phi(x_i) \rangle \tag{3}$$

Here, $\Phi(.)$ represents the mapping function to the feature space. The kernel function is used to compute the dot product of two sample points in the high-dimensional space. Table 1 summarises the kernel functions that have been widely used in SVMs [9]. In this table, α, β and d are kernel parameters that need to be set by the user.

The existing kernel functions can be classified as either local or global kernel functions [9]. Local kernel functions have a good learning ability but do not have a good generalisation ability. By contrast, global kernel functions have a good generalisation ability but a poor learning ability. For example, the radial kernel function is known to be a local function, whereas the polynomial kernel function is a global kernel function. The main challenge lies in determining which kernel function should be used for the current problem instance or the current decision point. This is because the kernel selection process strongly depends on the distribution of the input vectors and the relationship between the input vector and the output vector (predicted variables). However, the feature space distribution is not known in advance and may change during the course of the solution process, especially in big data cyber security. Consequently, different kernel functions may work well for different instances or in different stages of the solution process, and kernel selection may thus have a crucial impact on SVM performance. To address this issue, in this work, we use multiple kernel functions to improve the accuracy of our algorithm and avoid the shortcomings of using a single kernel function.

B. SVM CONFIGURATION FORMULATION

A traditional SVM configuration specifies the appropriate values for C, the kernel type and the kernel parameters. The aim is to find SVM configurations from the space of all possible configurations that minimise the expected error when tested on completely new data. This can be represented as a black-box optimisation problem that seeks an optimal cross-validation error ($\mathcal{I}$) and can be expressed as a tuple of the form <SVM, Θ, $\mathcal{D}$, $\mathcal{C}$, $\mathcal{S}$ >, where [26]

- SVM is the parametrised algorithm,
- Θ is the search space of the possible SVM configurations (C, kernel type and kernel parameters),
- $\mathcal{D}$ is the distribution of the set of instances,
- $\mathcal{C}$ is the cost function, and
- $\mathcal{S}$ is the statistical information.

$$\theta^* \in \arg\min_{\theta \in \Theta} \mathcal{I}(\Theta) \tag{4}$$

The goal is to optimise the cost function $\mathcal{C}$: $\Theta \times \mathcal{D} \longmapsto \mathcal{R}$ of the SVM over a set of problem instances $\pi \in \mathcal{D}$ to find

$$\theta^* \in \arg\min_{\theta \in \Theta} \frac{1}{|\mathcal{D}|} \sum_{\pi \in \mathcal{D}} \mathcal{C}(\theta, \pi) \tag{5}$$

Each $\theta \in \Theta$ represents one possible configuration of the SVM. The cost function $\mathcal{C}$ represents a single execution of the SVM using θ to solve a problem instance $\pi \in \mathcal{D}$. The statistical information $\mathcal{S}$ (e.g., a mean value) summarises the output of $\mathcal{C}$ obtained when testing the SVM across a set of instances. The main role of the proposed hyper-heuristic framework is to find a $\theta \in \Theta$ such that $\mathcal{C}(\theta)$ is optimised.

C. MULTI-OBJECTIVE FORMULATION

A multi-objective optimisation problem involves more than one objective function that all need to be optimised simultaneously [17]. A general multi-objective optimisation problem of the minimisation type can be represented as follows:

$$\begin{aligned} &\min\ \mathcal{F}(\mathcal{X}) = [f_1(x), f_2(x), \ldots, f_m(x)] \\ &\text{s.t. } \zeta(\mathcal{X}) = [\zeta_1(x), \zeta_2(x), \ldots, \zeta_c(x)] \geq 0 \\ &\quad x_i^{(L)} \leq x_i \leq x_i^{(U)} \end{aligned} \tag{6}$$

where $X = (x_1, x_2, \ldots, x_N)$ is a set of N decision variables, m is the number of objectives f_i, c is the number of constraints $\zeta(\mathcal{X})$, $x_i^{(L)}$ is the lower bound on the i^{th} decision variable, and $x_i^{(U)}$ is the upper bound on the i^{th} decision variable. In a multi-objective optimisation problem, two solutions are compared using the concept of *dominance* ($\prec$). Given two solutions a and b, a is said to dominate b ($a \prec b$) if a is superior or equal to b in all objectives and strictly superior to b in at least one objective [17]:

$$\mathcal{F}(a) \prec \mathcal{F}(b) \text{ iff } \begin{cases} f_i(a) \leq f_i(b), & \forall i = 1, \ldots, m \\ \exists i \in 1, \ldots, m, & f_i(a) < f_i(b) \end{cases} \tag{7}$$

A solution a is Pareto optimal if there is no other solution that dominates it. Accordingly, the set of all Pareto-optimal solutions is called the Pareto-optimal set (PS), and its image in the objective space is called the Pareto front (PF). The main goal of optimisation algorithms is to find the optimal PS.

For an SVM, the accuracy can be seen as a trade-off between the complexity (number of support vectors (NSV)) and the margin (C) [46]. A large number of support vectors may lead to over-fitting, whereas a large value of C to increase the generalisation ability may result in incorrect classification of some samples. This trade-off can be controlled through the selection of the SVM configuration (C, kernel type and kernel parameters). To this end, in this work, we consider the accuracy and complexity (number of support vectors (NSV)) achieved over the training instances as two conflicting objectives [46]:

- **Accuracy**. The accuracy represents the classification performance on a given problem instance. It can be calculated via so-called K-fold cross-validation (CV), in which the given instance is split into K disjoint sets $D_1, \ldots, D_K$ of the same size. For each configuration ($\theta \in \Theta$), the SVM is trained K times. In each iteration, $K - 1$ sets are used for training, and the other set is used for performance testing. The error (err) represents the average number of misclassified data sets over K training iterations.
- **Complexity**. The complexity represents the number of support vectors (NSV) or the upper bound on the expected number of errors.

The SVM configuration $\theta \in \Theta$ comprises the decision variables (C, kernel type and kernel parameters). The bounds on each decision variable represent its range of possible values. The two objectives to be optimised ($m = 2$) can be formulated as follows [46]:

$$\begin{aligned} &\min \mathcal{F}(\mathcal{X}) = [f_1(x), f_2(x)] \\ &\text{where } f_1(x) = err \\ &\qquad f_2(x) = NSV \end{aligned} \tag{8}$$

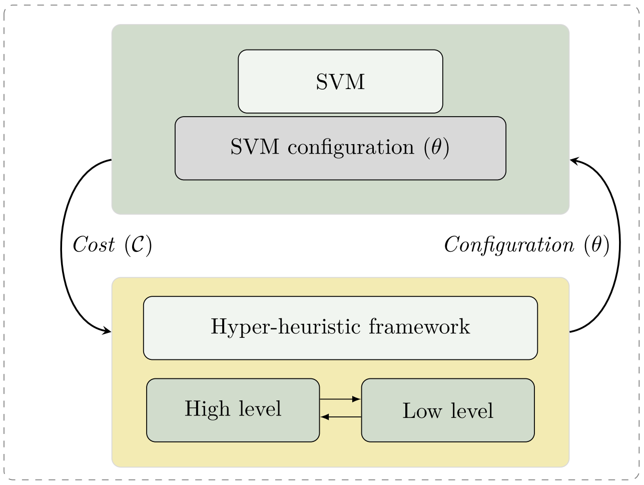


FIGURE 1. The proposed methodology.

where err represents the number of misclassified data sets and NSV denotes the number of support vectors.

IV. METHODOLOGY

The flowchart of the proposed methodology (abbreviated as HH-SVM) is depicted in Figure 1. The methodology has two parts: the SVM and the hyper-heuristic framework. The main role of the hyper-heuristic framework is to generate a configuration (C, kernel type and kernel parameters) and send it to the SVM. The SVM uses the generated configuration to solve a given problem instance and then sends the cost function (mean values of err and NSV) to the hyper-heuristic framework. This process is repeated for a certain number of iterations. In the following subsections, we discuss the proposed hyper-heuristic framework along with its main components.

A. THE PROPOSED HYPER-HEURISTIC FRAMEWORK

The proposed hyper-heuristic framework for configuration selection is shown in Figure 2. It has two levels: the *high-level strategy* and the *low-level heuristics* [11]. The high-level strategy operates on the heuristic space instead of the solution space. In each iteration, the high-level strategy selects a heuristic from the existing pool of low-level heuristics, applies it to the current solution to produce a new solution and then decides whether to accept the new solution. The low-level heuristics constitute a set of problem-specific heuristics that operate directly on the solution space of a given problem [39].

To address the bi-objective optimisation problem, we propose a population-based hyper-heuristic framework that operates on a population of solutions and uses an archive to save the non-dominated solutions. The proposed framework combines the strengths of decomposition- and Pareto (dominance)- based approaches to effectively approximate the Pareto set of SVM configurations. Our idea is to combine the diversity ability of the decomposition approach with the

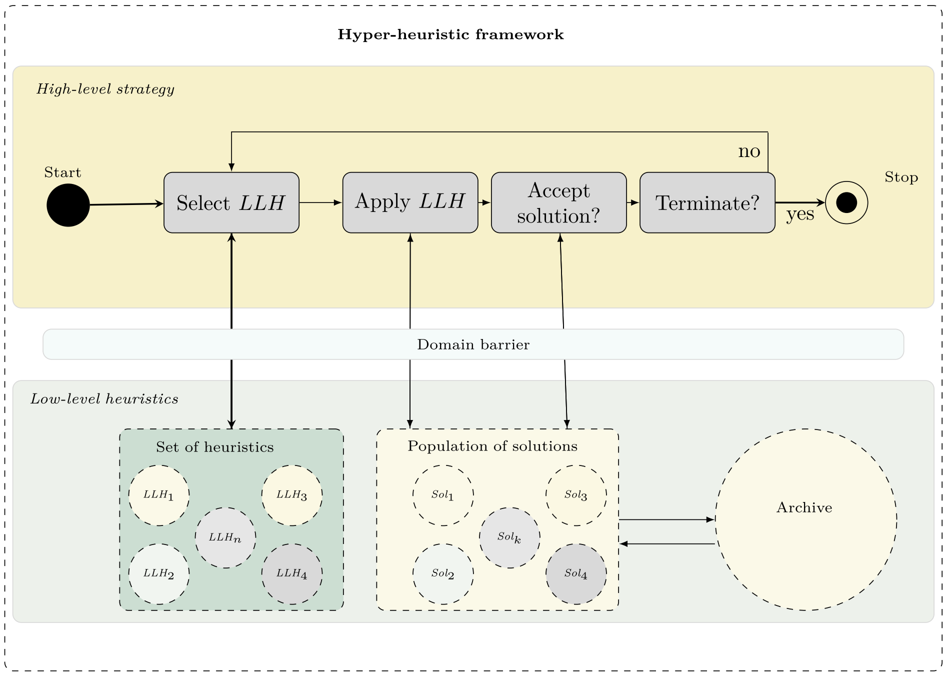
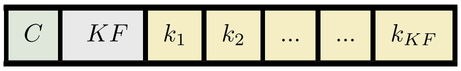


FIGURE 2. Hyper-heuristic framework.



C	KF	k_1	k_2	...	...	k_{KF}

FIGURE 3. Solution representation.

convergence power of the dominance approach. The decomposition approach operates on the population of solutions, whereas the dominance approach uses the archive. The hyper-heuristic framework generates a new population of solutions using either the old population, the archive, or both the old population and the archive. This allows the search to achieve a proper balance between convergence and diversity. It should be noted that seeking good convergence involves minimising the distances between the solutions and PF, whereas seeking high diversity involves maximising the distribution of the solutions along PF.

The main components of the proposed hyper-heuristic framework are discussed in the following subsections.

B. SOLUTION REPRESENTATION

In our framework, each solution represents one configuration ($\theta \in \Theta$) of the SVM, which is represented in the form of a one-dimensional array, as shown in Figure 3. In this figure, C is the margin parameter (or penalty), KF is the index of the selected kernel function, and k_1, k_2, ..., k_{KF} are the parameters of that kernel function.

C. POPULATION INITIALISATION

The population of solutions (PS) is randomly initialised. We use the following equation to assign a random value to each decision variable in a given solution (x):

$$x_i^p = l_i^p + Rand_i^p(0, 1) \times (u_i^p - l_i^p),$$
$$p = 1, 2, \ldots, |PS|, \quad i = 1, 2, \ldots, d \tag{9}$$

where i is the index of the decision variable, d is the total number of decision variables, p is the index of the solution, $|PS|$ is the population size, $Rand_i^p(0, 1)$ returns a random value in the range [0,1] for the i^{th} decision variable, l_i^p is the lower bound on the value of that decision variable, and u_i^p is the upper bound.

D. FITNESS CALCULATION

The fitness calculation assigns a value to each solution in the population that indicates how good this solution is compared with those in the current population. In this work, we use the MOEA/D approach to solve the multi-objective optimisation problem for selecting the SVM configuration. In this approach, a given multi-objective optimisation problem is first decomposed into a number of single-objective sub-problems, and then, all sub-problems are solved in a collaborative manner [57]. MOEA/D uses a scalarisation function to decompose a given problem into a number of scalarised single-objective sub-problems as follows [57]:

$$g^{te}(x, \lambda) = \max_{i \in m}(\lambda_i |z_i^* - f_i(x)|) \tag{10}$$

where g^{te} is the *Tchebycheff* decomposition approach, x is a given solution (SVM configuration), m is the number of objectives (in this work, $m = 2$), and $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m)$ is a weighting vector such that $\lambda_i \geq 0$, $\forall\, i \in m$. f_i is the fitness value for the i^{th} objective calculated using Equation (8).

$z^* = (z_1^*, z_2^*, \ldots, z_m^*)$ is the idea or the reference point, i.e., $z_i^* = \min\{f_i(x) \mid x \in \Omega\}$ for each $i = 1, 2, \ldots, m$.

E. HIGH-LEVEL STRATEGY

The main role of the high-level strategy is to automate the heuristic selection process [39]. Our proposed high-level strategy consists of the following steps.

1) SELECT

The selection step involves selecting one heuristic from the existing pool of heuristics using a selection mechanism. In this work, we use Multi-Armed Bandit (MAB) [39] as an on-line heuristic selection mechanism . In MAB, the past performance of each heuristic is saved; then, these performances are used to decide which heuristic should be selected. Each heuristic is associated with two variables: the empirical reward q_i and the confidence level n_i. The empirical reward q_i represents the average reward obtained during the search process using this heuristic. A higher value of the empirical reward is better. The confidence level n_i is the number of times that the i^{th} heuristic has previously been applied. Based on these two variables, MAB calculates the confidence interval for each heuristic and then selects the highest value using the following formula (Equation (11)):

$$\arg \max_{i=LLH_1 \ldots LLH_n} \left(q_{i(t)} + c\sqrt{\frac{2log \sum_{i=LLH_1}^{LLH_n} n_{i(t)}}{n_{i(t)}}} \right) \tag{11}$$

The pool of heuristics in our framework is denoted by $\{LLH\_1, \ldots, LLH\_n\}$, where n is the total number of heuristics. The index t is the time step or the number of the current iteration of the search. c is a scaling factor that adjusts the balance between the influence of the empirical reward and the confidence level to ensure that the confidence interval will not be excessively biased by either of these indicators. For example, a highly rewarded but infrequently used heuristic should most likely be less preferred than a frequently used heuristic whose reward value is only slightly lower.

The empirical reward $q_{i(t)}$ is calculated as follows:

$$q_{i(t)} = \frac{n_{i(t-1)} \times q_{i(t-1)} + r_{i(t)}}{n_{i(t)}} \tag{12}$$

where $r_{i(t)}$ is the accumulative reward value of heuristic LLH_i up to time t. The computation of $r_{i(t)}$ uses the equation below, Equation (13).

$$r_{i(t)} = \sum \Delta\, g^{te}(x, \lambda) \tag{13}$$

The component $r_{i(t)}$ is the sum total of the fitness improvement introduced by heuristic i from the beginning of the search up through the current iteration t.

2) APPLY

Two tasks are performed in the application step:

- **Solution selection**. This task determines which solutions should be selected to form the mating pool. In this work, we propose to utilise the advantages of both the decomposition- and Pareto (dominance)-based approaches during the solution selection process. In MOEA/D, each solution in the current population represents a sub-problem. To combine decomposition and dominance, we optimise each sub-problem using information from only its neighbouring sub-problems with probability p_n, from both the neighbouring sub-problems and the archive with probability p_{na}, or from only the archive with probability p_a. A fixed set of neighbouring solutions for each sub-problem is determined using the Euclidean distances between any two solutions based on their weight vectors.
- **Heuristic application**. In this task, the selected heuristic is applied to the created mating pool to evolve a new set of solutions.

3) ACCEPT SOLUTION

The acceptance step checks whether the newly generated solutions should be accepted. In this work, we first compare each solution x with its neighbouring sub-problems y. x will replace y if it is superior in terms of the scalarisation function, $g^{te}(x, \lambda) < g^{te}(y, \lambda)$. Next, we update the archive using non-dominated solutions.

4) TERMINATE

This step terminates the search process. This step checks whether the total number of iterations has been reached and, if so, terminates the search process and returns the set of non-dominated solutions. Otherwise, it starts a new iteration.

F. LOW-LEVEL HEURISTICS

The low-level heuristics (LLHs) are a set of problem-specific rules that operate directly on a given solution. Each LLH takes one or more solutions as input and then modifies them to generate a new solution. In this work, we utilise various sets of heuristics within the proposed framework. These heuristics have been demonstrated to be suitable for different problems and even for different stages of the same problem. These heuristics are chosen so as to incorporate various characteristics into the search and to include different search behaviours. The heuristics are as follows [20]:

1) **Parametrised Gaussian Mutation**

$$x = x + \mathcal{N}(Mean, \sigma 2) \tag{14}$$

where $Mean = 0$ and $\sigma 2 = 0.5$ is the standard deviation.

2) **Differential Mutation_1**

$$x = x_1 + F \times (x_2 - x_3) \tag{15}$$

3) **Differential Mutation_2**

$$x = x_1 + F \times (x_2 - x_3) + F \times (x_4 - x_5) \tag{16}$$

4) **Differential Mutation_3**

$$x = x_1 + F \times (x_1 - x_2) + F \times (x_3 - x_4) \tag{17}$$

where x_1, x_2, x_3, x_4 and x_5 are five different solutions selected from the mating pool in accordance with the solution selection process discussed in IV-E.2. F is a scaling factor, whose value is fixed to 0.9 in this study.

5) **Arithmetic Crossover**

$$x = \lambda \times x_1 + (1 - \lambda) \times x_2 \tag{18}$$

where λ is a randomly generated number, whose value is within the range $\lambda \in [0, 1]$. x_1 is the current sub-problem, and x_2 is the best solution in its neighbourhood.

6) **Polynomial Mutation**

$$x = \begin{cases} x_1 + \sigma \times (b - a), & \text{if } Rand \le 0.5 \\ x_1, & \text{otherwise} \end{cases} \tag{19}$$

and

$$\sigma = \begin{cases} (2 \times Rand)^{\frac{1}{(\eta+1)}} - 1, & \text{if } Rand \le 0.8 \\ 1 - (2 - 2 \times Rand)^{\frac{1}{(\eta+1)}}, & \text{otherwise} \end{cases}$$

where η is set to 0.3 and a and b are the lower and upper bounds, respectively, on the value of the i^{th} decision variable.

G. ARCHIVE

The archive saves the set of non-dominated solutions and is updated in each iteration. In this work, the newly generated solutions are first added to the archive. Then, following the concept of NSGA-II [18], we use the non-dominated sorting procedure to divide the archive into several levels of non-domination such that solutions in the first level have the highest priority to be selected, those in the second level have the second highest priority, etc. To ensure that the selected solutions are distributed along the Pareto front (PF), we may also select some solutions at the lowest level, depending on the crowding distance measure.

V. EXPERIMENTAL SETUP

This section summarises the benchmark instances that were used to assess the proposed framework and the parameter settings.

A. BENCHMARK INSTANCES

In this work, we analysed our proposed framework on two different cyber security problems with a broad range of different structures and sizes.

1) MICROSOFT MALWARE BIG DATA CLASSIFICATION

A first experimental evaluation uses Microsoft malware big data classification problem which was introduced for BIG 2015, hosted at Kaggle.[1] Microsoft provided a total of 500 GB of data of known malware files representing a mix of 9 families (classes) for 2 purposes: training and testing. A total of 10868 malwares are included in the training set, and 10783 malwares are included in the testing set. Each sample is a binary file with the extension ''.bytes'', and the corresponding disassembled file in the assembly language (text) has the extension ''.asm''. The ultimate goal is to train the classification algorithm using the training samples to effectively classify each of the testing samples into one of the 9 categories (malware families) such that the logloss function below is minimised:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}) \tag{20}$$

where N represents the number of training samples, M is the number of classes, log is the natural logarithm, and y_{ij} is a true label that takes a value of 1 if i is in class j and 0 otherwise. p_{ij} is the estimated probability that i belongs to class j. Further description can be found on the Kaggle web site.

2) ANOMALY INTRUSION DETECTION

In the second experimental evaluation, we used the NSL-KDD[2] anomaly intrusion detection instances. NSL-KDD includes selected records from the KDDCUP99 dataset collected by monitoring incoming network traffic. NSL-KDD has been used by many researchers to develop network-based intrusion detection systems (NIDs). The NSL-KDD problem instance consists of 125,973 training samples and 22,544 testing samples, each classified as either normal or anomalous (i.e., a network attack).

B. PARAMETER SETTINGS

The proposed framework has a few parameters that need to be determined in advance. To this end, we conducted a preliminary investigation to set the values of these parameters. We tested different values for each parameter while keeping the other parameters fixed. Table 2 shows the parameter settings investigated in our work as well as the final selected values.

TABLE 2. The parameter settings of our framework.

Parameter	Investigated range	Final value
Maximum number of generations	5-150	100
Population size, PS	5-30	20
Archive size	5-20	10
p_n	0.1-0.8	0.5
p_{na}	0.1-0.8	0.3
p_a	0.1-0.8	0.2

VI. RESULTS AND COMPARISONS

In this section, we present the results of the experiments that we conducted to evaluate the proposed HH-SVM framework described in this paper. We conducted two experimental tests. In the first test, HH-SVM was compared with each low-level heuristic individually. In the second test, the results of HH-SVM were compared with those of other algorithms proposed in the literature.

[1]https://www.kaggle.com/c/malware-classification/data

[2]http://nsl.cs.unb.ca/NSL-KDD/

TABLE 3. Comparison of the HH-SVM results against the results of all low-level heuristics (LLH_1 to LLH_6) individually.

Algorithm / Instance	BIG 2015	NSL-KDD
HH-SVM	**0.0031**	**85.69**
LLH_1	0.0332	77.24
LLH_2	0.0223	66.45
LLH_3	0.0214	80.01
LLH_4	0.0208	79.22
LLH_5	0.0227	80.37
LLH_6	0.0216	76.93

TABLE 4. The NSV values obtained by HH-SVM and the individual low-level heuristics (LLH_1 to LLH_6).

Algorithm / Instance	BIG 2015	NSL-KDD
HH-SVM	**20**	**8**
LLH_1	33	12
LLH_2	34	17
LLH_3	34	20
LLH_4	42	16
LLH_5	41	22
LLH_6	38	21

TABLE 5. The *p*-values of HH-SVM compared with the individual low-level heuristics.

HH-SVM vs.	BIG 2015	NSL-KDD
	p-value	*p*-value
LLH_1	0.001	0.000
LLH_2	0.000	0.010
LLH_3	0.020	0.011
LLH_4	0.000	0.000
LLH_5	0.012	0.000
LLH_6	0.022	0.000

TABLE 6. Comparison of the *logloss* results of HH-SVM and other algorithms.

Algorithm	BIG 2015
HH-SVM	**0.0031**
AE	0.0748
RF	0.0988557
OXB	0.0063

A. HH-SVM COMPARED WITH INDIVIDUAL LOW-LEVEL HEURISTICS

This section compares the proposed HH-SVM with each low-level heuristic (LLH). Our aim is to assess the benefits of the proposed hyper-heuristic framework and the effects of using multiple LLHs on the search performance. To this end, we tested each LLH separately. The outcomes were the results of seven different algorithms, denoted by HH-SVM, LLH_1, LLH_2, LLH_3, LLH_4, LLH_5, and LLH_6. All algorithms were executed under identical conditions, and the same base components were utilised on both problem instances (BIG 2015 and NSL-KDD). The average results over 31 independent runs are compared in Table 3. The BIG 2015 results are compared in terms of *logloss*, for which lower values are better (20), whereas the NSL-KDD results are compared based on *accuracy*, for which higher values are better. In the table, the best results achieved among all algorithms are indicated in bold font. From the results, we can see that HH-SVM outperforms all other algorithms (LLH_1, LLH_2, LLH_3, LLH_4, LLH_5, and LLH_6) on both BIG 2015 and NSL-KDD. Table 4 reports the numbers of support vectors (NSV) for HH-SVM and the compared algorithms on both instances, for which lower values are better. As seen from this table, the proposed HH-SVM framework produced lower NSV values for both BIG 2015 and NSL-KDD compared with LLH_1, LLH_2, LLH_3, LLH_4, LLH_5, and LLH_6. These positive results justify the use of the proposed hyper-heuristic framework and the use of the pool of heuristics (LLHs).

To further verify these results, we conducted statistical tests using the Wilcoxon test with a significance level of 0.05. The *p*-values for the HH-SVM results versus those of LLH_1, LLH_2, LLH_3, LLH_4, LLH_5, and LLH_6 are reported in Table 5. In this table, a *p*-value of less than 0.05 indicates that HH-SVM is statistically superior to the algorithm considered for comparison. A value greater than 0.05 indicates that the performance of our proposed HH-SVM framework is not significantly superior. From the table, we can clearly see that all *p*-values are less than 0.05, indicating that HH-SVM is statistically superior to LLH_1, LLH_2, LLH_3, LLH_4, LLH_5, and LLH_6 across both BIG 2015 and NSL-KDD.

B. HH-SVM COMPARED WITH OTHER ALGORITHMS

In this section, the results of HH-SVM are compared with those reported in the literature. For BIG 2015, we consider the following algorithms in the comparison:

- XGBoost (AE) [55]
- Random Forest (RF) [23]
- Optimised XGBoost (OXB) [1]

For the NSL-KDD instance, the *accuracy* results obtained by HH-SVM are compared against those of the following algorithms:

- Gaussian Naive Bayes Tree (GNBT) [48]
- Fuzzy Classifier (FC) [27]
- Decision Tree (DT) [33]

The results of HH-SVM and the other algorithms for the BIG 2015 and NSL-KDD problem instances are summarised in Table 6 and Table 7, respectively. Similar to the literature, the results for BIG 2015 in Table 6 are given in the form of the *logloss* values achieved by the various algorithms, whereas in Table 7, all algorithms are compared in terms of the *accuracy* measure. In the *logloss* comparisons, a lower value indicates better performance, whereas in the *accuracy* comparisons, a higher value indicates better performance. The best result obtained among the compared algorithms is indicated in bold in both tables. As shown in Table 6, HH-SVM has a lower *logloss* value than those of AE, RF and

TABLE 7. Comparison of the *accuracy* results of HH-SVM and other algorithms.

Algorithm	NSL-KDD
HH-SVM	**85.69**
GNBT	82.02
FC	82.74
DT	80.14

OXB for the BIG 2015 instance, whereas in Table 7, the *accuracy* value of HH-SVM is higher than those of GNBT, FC and DT for the NSL-KDD instance. The results demonstrate that HH-SVM is an effective methodology for addressing cyber security problems. The good performance of HH-SVM can be attributed to its ability to design and optimise different SVMs for different problem instances and for different stages of the solution process.

VII. CONCLUSION

In this work, we proposed a hyper-heuristic SVM optimisation framework for big data cyber security problems. We formulated the SVM configuration process as a bi-objective optimisation problem in which accuracy and model complexity are treated as two conflicting objectives. This bi-objective optimisation problem can be solved using the proposed hyper-heuristic framework. The framework integrates the strengths of decomposition- and Pareto-based approaches to approximate the Pareto set of configurations. Our framework has been tested on two benchmark cyber security problem instances: Microsoft malware big data classification and anomaly intrusion detection. The experimental results demonstrate the effectiveness and potential of the proposed framework in achieving competitive, if not superior, results compared with other algorithms.

REFERENCES

[1] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proc. 6th ACM Conf. Data Appl. Secur. Privacy*, 2016, pp. 183–194.

[2] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, Jun. 1975.

[3] S. Ali and K. A. Smith-Miles, "A meta-learning approach to automatic kernel selection for support vector machines," *Neurocomputing*, vol. 70, nos. 1–3, pp. 173–186, 2006.

[4] N.-E. Ayat, M. Cheriet, and C. Y. Suen, "Automatic model selection for the optimization of SVM kernels," *Pattern Recognit.*, vol. 38, no. 10, pp. 1733–1745, 2005.

[5] Y. Bao, Z. Hu, and T. Xiong, "A PSO and pattern search based memetic algorithm for SVMs parameters optimization," *Neurocomputing*, vol. 117, pp. 98–106, Oct. 2013.

[6] R. C. Barros, M. P. Basgalupp, A. C. P. L. F. de Carvalho, and A. A. Freitas, "A hyper-heuristic evolutionary algorithm for automatically designing decision-tree algorithms," in *Proc. 14th Annu. Conf. Genet. Evol. Comput.*, 2012, pp. 1237–1244.

[7] M. P. Basgalupp, R. C. Barros, T. S. da Silva, and A. C. P. L. F. de Carvalho, "Software effort prediction: A hyper-heuristic decision-tree based approach," in *Proc. 28th Annu. ACM Symp. Appl. Comput.*, 2013, pp. 1109–1116.

[8] M. P. Basgalupp, R. C. Barros, and V. Podgorelec, "Evolving decision-tree induction algorithms with a multi-objective hyper-heuristic," in *Proc. 30th Annu. ACM Symp. Appl. Comput.*, 2015, pp. 110–117.

[9] A. Ben-Hur and J. Weston, "A user's guide to support vector machines," in *Data Mining Techniques for the Life Sciences. Methods in Molecular Biology* (Methods and Protocols), O. Carugo and F. Eisenhaber, Eds. vol 609. New York, NY, USA: Humana Press, 2010, pp. 223–239.

[10] D. Brumley, C. Hartwig, Z. Liang, J. Newsome, D. Song, and H. Yin, "Automatically identifying trigger-based behavior in malware," in *Botnet Detection (Advances in Information Security*, W. Lee, C. Wang, and D. Dagon, Eds. Boston, MA, USA: Springer, 2008.

[11] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of Metaheuristics* (International Series in Operations Research & Management Science), vol. 146, M. Gendreau and J. Y. Potvin, Eds. Boston, MA, USA: Springer, 2010.

[12] A. Chalimourda, B. Schölkopf, and A. J. Smola, "Experimentally optimal ν in support vector regression for different noise models and parameter settings," *Neural Netw.*, vol. 17, no. 1, pp. 127–141, 2004.

[13] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, 2011.

[14] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 171–209, Apr. 2014.

[15] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge, U.K.: Cambridge Univ. Press, 2000.

[16] M. Damshenas, A. Dehghantanha, and R. Mahmoud, "A survey on malware propagation, analysis, and detection," *Int. J. Cyber-Secur. Digit. Forensics*, vol. 2, no. 4, pp. 10–29, 2013.

[17] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, vol. 16. Hoboken, NJ, USA: Wiley, 2001.

[18] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[19] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, 2012, Art. no. 6.

[20] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, vol. 53. Heidelberg, Germany: Springer, 2003.

[21] E. Filiol, "Malware pattern scanning schemes secure against black-box analysis," *J. Comput. Virol.*, vol. 2, no. 1, pp. 35–50, 2006.

[22] E. Filiol, G. Jacob, and M. Le Liard, "Evaluation methodology and theoretical model for antiviral behavioural detection strategies," *J. Comput. Virol.*, vol. 3, no. 1, pp. 23–37, 2007.

[23] *Malware Classification: Distributed Data Mining With Spark*. [online] Available: http://msan-vs-malware.com/

[24] T. A. F. Gomes, R. B. C. Prudêncio, C. Soares, A. L. D. Rossi, and A. Carvalhoc, "Combining meta-learning and search techniques to select parameters for support vector machines," *Neurocomputing*, vol. 75, no. 1, pp. 3–13, 2012.

[25] K. Greer, "A stochastic hyperheuristic for unsupervised matching of partial information," *Adv. Artif. Intell.*, vol. 2012, 2012, Art. no. 790485, doi: 10.1155/2012/790485.

[26] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *J. Artif. Intell. Res.*, vol. 36, no. 1, pp. 267–306, 2009.

[27] P. Krömer, J. Platoš, V. Snášel, and A. Abraham, "Fuzzy classification by evolutionary algorithms," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2011, pp. 313–318.

[28] A. C. Lorena and A. C. P. L. F. de Carvalho, "Evolutionary tuning of SVM parameter values in multiclass problems," *Neurocomputing*, vol. 71, nos. 16–18, pp. 3326–3334, 2008.

[29] M. M. Masud *et al.*, "Cloud-based malware detection for evolving data streams," *ACM Trans. Manage. Inf. Syst.*, vol. 2, no. 3, 2011, Art. no. 16.

[30] P. B. C. Miranda, R. B. C. Prudêncio, A. C. P. L. F. Carvalho, and C. Soares, "Combining meta-learning with multi-objective particle swarm algorithms for SVM parameter selection: An experimental analysis," in *Proc. Brazilian Symp. Neural Netw. (SBRN)*, Oct. 2012, pp. 1–6.

[31] P. B. C. Miranda, R. B. C. Prudêncio, A. C. P. L. F. de Carvalho, and C. Soares, "Multi-objective optimization and Meta-learning for SVM parameter selection," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jun. 2012, pp. 1–8.

[32] P. B. C. Miranda, R. B. C. Prudêncio, A. P. L. F. De Carvalho, and C. Soares, "A hybrid meta-learning architecture for multi-objective optimization of SVM parameters," *Neurocomputing*, vol. 143, pp. 27–43, Nov. 2014.

[33] M. Mohammadi, B. Raahemi, A. Akbari, and B. Nassersharif, "New class-dependent feature transformation for intrusion detection systems," *Secur. Commun. Netw.*, vol. 5, no. 12, pp. 1296–1311, 2012.

[34] J. C. Ortiz-Bayliss, H. Terashima-Marín, and S. E. Conant-Pablos, "Learning vector quantization for variable ordering in constraint satisfaction problems," *Pattern Recognit. Lett.*, vol. 34, no. 4, pp. 423–432, 2013.

[35] C. R. Panigrahi, M. Tiwari, B. Pati, and R. Prasath, "Malware detection in big data using fast pattern matching: A hadoop based comparison on GPU," in *Mining Intelligence and Knowledge Exploration* (Lecture Notes in Computer Science), vol. 8891, R. Prasath, P. O'Reilly, and T. Kathirvalavakumar, Eds. Cham, Switzerland: Springer, 2014.

[36] M. Reif, F. Shafait, and A. Dengel, "Meta-learning for evolutionary parameter optimization of classifiers," *Mach. Learn.*, vol. 87, no. 3, pp. 357–380, Jun. 2012.

[37] A. Rosales-Pérez, J. A. Gonzalez, C. A. C. Coello, H. J. Escalante, and C. A. Reyes-Garcia, "Surrogate-assisted multi-objective model selection for support vector machines," *Neurocomputing*, vol. 150, pp. 163–172, Feb. 2015.

[38] N. R. Sabar, J. Abawajy, and J. Yearwood, "Heterogeneous cooperative co-evolution memetic differential evolution algorithm for big data optimization problems," *IEEE Trans. Evol. Comput.*, vol. 21, no. 2, pp. 315–327, Apr. 2017.

[39] N. R. Sabar, M. Ayob, G. Kendall, and R. Qu, "A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems," *IEEE Trans. Cybern.*, vol. 45, no. 2, pp. 217–228, Feb. 2015.

[40] B. Schölkopf and A. J. Smola, *Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT press, 2002.

[41] S. Shaw, M. K. Gupta, and S. Chakraborty, "Cloud based malware detection," in *Proc. 5th Int. Conf. Frontiers Intell. Comput., Theory Appl. (FICTA)*, vol. 1. 2017, pp. 485–495.

[42] K. Sim, E. Hart, and B. Paechter, "A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution," in *Parallel Problem Solving from Nature—PPSN XII* (Lecture Notes in Computer Science), vol. 7492, C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, Eds. Berlin, Germany: Springer, 2012.

[43] C. Soares, P. B. Brazdil, and P. Kuba, "A meta-learning method to select the kernel width in support vector regression," *Mach. Learn.*, vol. 54, no. 3, pp. 195–209, 2004.

[44] D. Song *et al.*, "BitBlaze: A new approach to computer security via binary analysis," in *Information Systems Security*. 2008, pp. 1–25.

[45] S. Suthaharan, "Big data classification: Problems and challenges in network intrusion prediction with machine learning," *SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 4, pp. 70–73, Apr. 2014.

[46] T. Suttorp and C. Igel, "Multi-objective optimization of support vector machines," in *Multi-Objective Machine Learning*, 2006, pp. 199–220.

[47] C. Tankard, "Big data security," *Netw. Secur.*, vol. 2012, no. 7, pp. 5–8, 2012.

[48] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl. (CISDA)*, Jul. 2009, pp. 1–6.

[49] C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos, "Big data analytics: A survey," *J. Big Data*, vol. 2, no. 1, p. 21, 2015.

[50] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer, 2013.

[51] A. Vella, D. Corne, and C. Murphy, "Hyper-heuristic decision tree induction," in *Proc. World Congr. Nature Biol. Inspired Comput. (NaBIC)*, Dec. 2009, pp. 409–414.

[52] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artif. Intell. Rev.*, vol. 18, no. 2, pp. 77–95, 2002.

[53] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, 2017, Art. no. 41.

[54] Y. Ye *et al.*, "Combining file content and file relations for cloud based malware detection," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 222–230.

[55] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 3854–3861.

[56] J. Zhang *et al.*, "Evolutionary computation meets machine learning: A survey," *IEEE Comput. Intell. Mag.*, vol. 6, no. 4, pp. 68–75, Apr. 2011.

[57] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, Dec. 2007.

NASSER R. SABAR is a Lecturer with the Computer Science and IT Department, La Trobe University, Australia. He has published over 55 papers in international journals and peer-reviewed conferences. His current research interests include design and development of hyper-heuristic frameworks, deep learning, machine learning, evolutionary computation and hybrid algorithms with a specific interest in big data optimisation problems, cloud computing, dynamic optimisation and data-mining problems.

XUN YI is currently a Professor with the School of Computer Science and IT, RMIT University, Australia. His research interests include applied cryptography, computer and network security, mobile and wireless communication security, and privacy-preserving data mining. He has published over 150 research papers in international journals, such as the IEEE Transaction Knowledge and Data Engineering, the IEEE Transaction Wireless Communication, the IEEE Transaction Dependable and Secure Computing, the IEEE Transaction Circuit and Systems, and conference proceedings. He has ever undertaken program committee members for over 20 international conferences. Since 2014, he has been an Associate Editor for the IEEE Transaction Dependable and Secure Computing.

ANDY SONG is a Senior Lecturer with the Computer Science and IT Department, School of Science, RMIT University, Australia. His research interests include machine learning especially evolutionary computing-based learning on solving complex real-world problems, including texture analysis, motion detection, activity recognition, event detection, and optimization. Recently, he has been active in establishing cutting-edge techniques, which integrate machine intelligence, mobile and crowd sensing, to benefit transportation, logistics and warehouse industry. He collaborates with a range of industry partners.