ISSN:2229-6107

INTERNATIONAL JOURNAL OF PURE AND APPLIED SCIENCE & TECHNOLOGY

E-mail :
editor.ijpast@gmail.com
editor@ijpast.in

www.ijpast.in

INTERNATIONAL JOURNAL OF PURE AND APPLIED SCIENCE & TECHNOLOGY
IJPAST

ISSN 2229-6107 www.ijpast.in

Vol 10,Issue 4.Dec 2020

SUPPORT VECTOR MACHINES, A BI-OBJECTIVE HYPER-HEURISTIC, FOR THE CYBER-SAFETY OF LARGE DATA SETS

Mr. D Srikar, Mr. USV Vinod, Mr. JV Rama Kumar,

ABSTRACT

Is a major issue, and that this fact poses a significant challenge to the academic community. In order to deal with the security issues that come with large data, machine learning techniques have been proposed as a possible solution. Support vector machines (SVMs) have been one of the most popular of these methods. Obtained impressive results on a wide range of classification tasks. However, in order to set up an efficient SVM, the user must first de ne the appropriate SVM con gyration in advance—a difficult operation that calls for specialized expertise and a great deal of trial and error. Here, we provide a formalization of the SVM configuration procedure as a bi-objective optimization problem, with accuracy and model complexity as two competing goals. New problem-domain-agnostic hyper-heuristic framework for bi-objective optimization is proposed. The first ever hyper-heuristic for this issue has been created just now. To solve this problem, the authors suggest a hyper-heuristic framework that combines both high-level strategy and heuristics. The search performance is utilized by the high-level approach to determine which of many possible low-level heuristics should be used to produce a new SVM con gyration. Each of the low-level algorithms takes a unique approach to successfully searching the space of possible SVM configurations. The proposed framework adaptively combines the benefits of decomposition- and Pareto-based techniques to approximate the Pareto set of SVM con gyrations, allowing it to tackle the problem of bi-objective optimization. Two cyber security challenges, Microsoft malware big data classfication and anomalous intrusion detection, were used to assess the performance of the suggested system. In comparison to its contemporaries and other algorithms, the acquired results show that the suggested framework is very successful.

I. INTRODUCTION

Rapid progress in areas like mobile, social and the Internet of Things results in an explosion of data in digital form. Big has special meaning in this setting. Data has arisen to characterize these enormous stores Of digital information. Big data is defined as "very big and complicated datasets with both structured and Unstructured data that are created on a regular basis and need analysis in a very short period of time" [49]. Huge data, as contrast to the big database, refers to data sets that are either too large, change too rapidly, or are too complex to be processed by conventional methods. Three criteria often used to define big data are scale, diversity, and speed (aka 3Vs). The 3Vs are the attributes or dimensions of data, with volume referring to the sheer bulk of the data, variety indicating that the data came from a wide range of sources, and velocity referring to the swiftness with which new data may be created, streamed, and aggregated [49]. The growth of all three features (3Vs), not simply the volume alone, is primarily Responsible for the complexity and difficulty posed by big data [14]. Researchers, analysts, and business users may all benefit from the speedier, more informed judgments made possible by big data [38]. Researchers and practitioners from a wide range of groups, including academia, business, and government organizations, have focused on this _led because of its practical applications and problems [14].

Assistant Professor[1,2,3],
Department of Computer Science and Engineering,
Bhimavaram Institute of Engineering and Technology, Bhimavaram, Andhra Predesh, India.
E.mail id: srikar1974@gmail.com, E.mail id: satyamca41@gmail.com
E.mail id: jvramakumar@gmail.com

The 3Vs and data security were already problems, but big data introduced a new one. Evidence suggests that big data may do more than just enhance the scope of not only exacerbate existing security issues, but also provide new cyber-security dangers that must be dealt with in novel and resourceful ways. When it comes to gaining insights from big data, security is universally recognized as the top priority [47]. Threat detection, authentication, and steganography are all examples of problems in cyber security that arise with large amounts of data [45]. The biggest problem with large data security is finding malware.

Malware, an abbreviation for "harmful software," is a broad category of malicious programmers that may infect systems and leak sensitive data over networks, email, or websites [53]. Researchers and organizations have recognized the problems that may be created by this malicious software, and therefore have recognized the need to develop new strategies to avoid them. Malware is a major concern in big data, yet surprisingly little study has been conducted in this field [47]. Methods for detecting malware range from those based on signatures [22], to those that track user activity [54], and still others that use patterns [19], [53]. Existing malware detection approaches, however, are primarily suggested to deal with small-scale datasets and are unable to handle enormous data within a decent length of time. These approaches are also prohibitively expensive to implement and maintain and have low success rates [53]. Attackers may readily circumvent them. Machine learning (ML) techniques for categorizing unknown patterns and malicious software have been offered as a solution to the aforementioned problems [45], [53]. Preliminary results from ML seem good. Outcomes for categorizing and identifying previously discovered malicious software.

SVMs, or support vector machines, are a common ML approach because to their impressive performance in a wide range of practical settings [15]. SVMs' high level of performance and scalability [40] are major factors in their widespread use. Despite these benefits, an SVM's performance is very sensitive to its con gyration [9].

The soft margin parameter (or penalty) and the kernel type and its parameters are often chosen during an SVM con gyration. There are a number of different approaches to choosing SVM con gyrations that have been discussed in the academic literature. The phrasing of the SVM configuration issue and the optimization technique used allow for a categorization of these approaches [9], [12]. The performance of a created SVM configuration may be evaluated using k-fold cross-validation if just a single criterion is utilized, or using many criteria, such as model correctness and model complexity, in order to make an informed decision [46]. Grid search techniques, gradient-based approaches, and meta-heuristic methods are some of the optimization strategies that may be used. It's simple to construct grid search algorithms, and they've proven effective [13]. However, they come with a high computational cost, reducing their usefulness for large data issues. The primary drawbacks of gradient-based approaches are that they are very dependent on the beginning point and that the objective function must be differentiable [4]. To get beyond the limitations of grid search and gradient-based approaches, meta-heuristic approaches have been proposed [5, 28, and 56].

It is true that a meta-heuristic method's performance is very sensitive to the parameters and operators that are chosen, but this is because these choices are based on a number of factors. Established as a difficult and lengthy task.

Furthermore, in most research, a single kernel is used, and its parameter space is searched. In this paper, we provide a new bi-objective hyper-heuristic framework for SVM configuration optimization. Due to its flexibility and ability to produce highly competitive con gyrations, hyper heuristics are superior to other approaches. To _end a powerful SVM con gyration for big data cyber security, our proposed hyper-heuristic framework includes a number of critical components that set it apart from prior research. As a first step, the framework takes into account a bi-objective formulation of the SVM con gyration issue, where accuracy and model complexity are seen as competing goals. Second, the framework determines the kernel type, kernel settings, and soft margin parameter to use. Third, to _end an approximate Pareto set of SVM con durations, the hyper-heuristic framework combines the benefits of decomposition- and Pareto-based techniques in an adaptable fashion.

II. RELATED WORK

Here, we briefly review other studies that touch on the same topics of malware detection and meta-learning.

Hyper-heuristics for classification are also discussed.

A. MALWARE DETECTION TECHNIQUES

Ye et al. [53], in their recent study, divided the various approaches of detecting malware into three categories: those that rely on signatures, those that rely on patterns, and those that rely on the cloud. The majority of current malware detection approaches rely on signature matching [21], [22].

[22] A signature is a unique short string of bytes de need for each known piece of malicious software that may be used to identify new, previously unknown malware. Signature-based detection approaches may identify malicious software, but they need regular updates to incorporate the signature of newly discovered malware in the signature database. In addition, malware makers may quickly circumvent them via the use of encryption, polymorphism, or obfuscation [53]. In addition, creating a signature database is often a manual process performed by subject experts, which is a difficult and time-consuming effort [16]. If a piece of malware software is known to possess a certain set of patterns, then the detection technique may identify it as malicious. To differentiate malicious software, domain specialists extract the patterns. In this case, software and harmful _les [2, 10, and 35] are the offenders. Malware software analysis and pattern extraction by domain specialists, however, is mistake prone and time intensive [19]. Because malware software evolves so rapidly, it has become apparent that manual analysis and extraction are substantial obstacles to the development of pattern-based detection approaches [53]. Using a server to store detection software, cloud-based detection approaches enable malware detection to be performed in client-server architecture [41, [53], and [54].

However, the number of accessible cluster nodes and the operating duration of the detection algorithms have a significant impact on cloud-based detection [29]. Because of this, multifunctional malicious software may evade detection for longer than intended.

B. META-LEARNING APPROACHES

For optimal performance, the parameters of a classic SVM—which may be tweaked in several ways—must be optimized [9]. The use of meta-learning strategies to _end the optimal parameters and settings for support vector machines. Meta-learning is a strategy for figuring out the features of a problem and the optimal solution algorithm [52]. In specifically, it seeks to understand which aspects of a problem affect an algorithm's performance so that the best possible solution may be suggested.

To _end the parameter values of Gaussian kernel for SVM to tackle regression issues, Soars et al. [43] developed a meta-learning technique. In order to determine what the optimal value of the kernel width parameter should be, the authors used K-NN as a ranking approach. In order to come up with viable seed solutions for the genetic algorithm, Reef et al. [36] combined meta-learning with case-based reasoning. In order to address a specific instance of a problem, the suggested evolutionary algorithm is utilized to _end optimal parameter values for a certain classifier. When deciding on a kernel technique for SVM, Ali and Smith-Miles [3] used a meta-learning strategy that combines data from classical, distance, and distribution statistics. Using a combination of meta-learning and search algorithms, Gomes et al. [24] suggested a hybrid approach to choosing SVM parameters. Additionally, references like as [30] [32] and [37] demonstrate the application of metal earning strategies in the context of SVM tuning.

Although meta-learning strategies have proven useful for tuning SVMs' parameters, they still struggle with over- thing. For this reason, the extracted only instances that have been utilized for training are captured by problem features. Most previous methods were developed to fine-tune a single kernel technique and were only ever used to small-scale examples. To efficiently manage massive data issues, we present a system that employs kernel approaches and formulates the selection process as a bi-objective optimization.

C. HYPER-HEURISTICS

The goal of the emergent search technique known as hyper-heuristic is to mechanically assemble or generate a powerful problem solution [11]. The classic hyper-heuristic architecture uses a collection of design alternatives as input to determine which one should be chosen. Instead of producing a solution, a hyper-heuristic framework generates a problem solver [39]. For the one-dimensional been packing issue, Sims et al. [42] suggested a hyper-heuristic framework to construct a collection of characteristics that characterize a particular instance.

This paper makes use of a hyper-heuristic framework to determine which heuristic should be employed to address the present situation. To address this issue, Ortiz-Bayle's et al. [34] suggested a hyper-heuristic approach based on learning vector quantization neural networks. In order to choose which heuristic to use, the hyper heuristic framework was taught to analyze the instance's characteristics.

For unsupervised matching of fragmentary data, Greer [25] provided a stochastic hyper-heuristic framework. To choose which subset of features to use, the hyper heuristic framework was put into place as a feature selection technique. In order to forecast

the time and effort required to develop software, Basgalupp [7] presented a hyper-heuristic framework to generate a decision tree.
Furthermore, [6], [8], and [51] all employ hyper-heuristic frameworks to develop classyfiers.

III. PROBLEM DESCRIPTION

There are three subparts to this section. After outlining the SVM procedure, we define the con duration issue. In conclusion, we show the suggested the SVM configuration issue is formulated in terms of several objectives.

MACHINES THAT PRODUCE VECTOR SUPPORT A.

Supervised learning models, of which SVMs are an example [50], have seen extensive usage for both classification and regression purposes.

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, which are grounded in statistical learning theory, are superior to other classification algorithms in their ability to circumvent local optima. Known as a kernel-based learning method, a Support Vector Machine (SVM) looking for the best hyper plane In order to achieve linear separation, the kernel learning process transforms the input patterns into a higher-dimensional feature space. Let's pretend we have L data sets (xi, yi) j (xi, yi) 2 Rave, yes 2 R, where xi is a v-dimensional input vector and yes is a matching y-dimensional output vector. In order to build the best possible decision-making function in the feature space, the SVM method maps the input vector xi onto that space.

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

s.t.

$$y_i - f(x_i) \le \varepsilon + \xi_i$$
$$f(x_i) - y_i \le \varepsilon + \xi_i^*$$
$$\xi_i,\ \ \xi_i^* \ge 0,\ i = 1, 2, \ldots \mathcal{L}$$

where $\omega = (\omega_1, \omega_1, \omega_1, \ldots, \omega_{\aleph})^T$ Is the number of support vectors, which is determined by the insensitive loss coef client, "the margin parameter (or penalty), and C, the weight vector. Two slack variables, I and I are provided, both of which may be non-negative numbers. By recasting Equation (1) as the following dual issue, we can determine the best course of action:

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

Lagrange coef clients I and _ I stand for the two slack variables, b 2 R is the bias, and K(x, xi) is the theorem.

The Kernel Function

$$K(x, x_i) = \langle \Phi(x), \Phi(x_i) \rangle \qquad (3)$$

The mapping function from the feature space is denoted here by 8(.). The dot product of two points in the high-dimensional sample space is calculated using the kernel function. Table 1 provides a brief overview of the most common kernel functions used by SVMs [9]. The user is responsible for determining the values and d in this table, which are all kernel parameters.
The current kernel functions may be divided into two categories: local and global [9]. Although local kernel functions are adept at learning, they struggle when it comes to generalization. Global kernel functions, on the other hand, are excellent at generalization but terrible at specifics. Some kernel functions are known to be local, like the radial kernel function, while others are known to be global, like the polynomial kernel function. Determining which kernel function is appropriate for a given issue instance or decision point is the major difficulty. The kernel selection procedure is very sensitive to the input vector distribution and the nature of the link between the input and output vectors (predicted variables). In big data cyber security, in particular, the feature space distribution is not known in advance and may vary over the course of the solution process. Therefore, kernel selection may have a significant effect on SVM performance, since different kernel functions may work well for certain instances or phases of the solution process. We employ a combination of kernel functions in our study to solve this problem and overcome the limitations of utilizing a single kernel function.

B. SVM CONFIGURATION FORMULATION

Commonly, C, the kernel type, and the kernel parameters are set in a standard SVM con duration. The goal is to find SVM configurations inside the configuration space. A comprehensive set of configurations that, when evaluated on fresh data, provide the smallest potential error. This may be written as a tulle of the type SVM, _, D, C, S >, where [26] describes the issue as a black-box optimization seeking an optimum cross-validation error (I). The parameterized method is denoted as SVM, the search space of all potential SVM con durations (C, kernel type, and kernel parameters) by _, the distribution of the data set by D, the cost function by C, and the statistical data by S.

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

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

Multiple SVM configurations are shown here as sets of 2. As shown by the cost function C, each iteration of the SVM with applied to an individual issue instance 2 D is equivalent to one run of the SVM. When an SVM is put to the test over several instances, some statistical information is gathered, summarized, and presented as S (e.g., a mean value). The suggested hyper-heuristic framework's primary responsibility is to find a 2 where C (_) is maximized.

The C. MULTI-GOAL FORMULA

There are several objective functions in a multi-objective optimization problem [17], each of which must be optimized concurrently. The following is a representation of a generic multi-objective optimization problem of the minimization type:

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

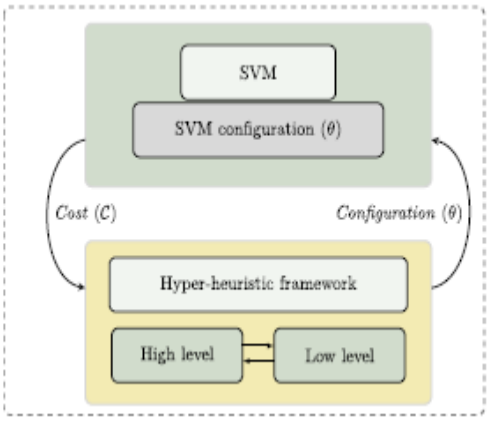
where N is the total number of decision variables (x1, x2,..., an), m is the total number of goals (if), c is the total number of constraints ((X), and x is the decision variable being optimized (L) The it decision variable has a lower limit of I and an upper bound of x (U) I. Dominance (_) is used to judge the relative merits of two solutions to a multiobjective optimization problem. If there are two solutions, a and b, and an is better than b in all respects, then an is dominant over b (a b) [17].

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

Solution that preempts all others the collection of all such Pareto-optimal solutions is referred to as the Pareto-optimal set (PS), and The Pareto front is an image in the objective space (PF). Optimization algorithms' primary focus is on finding the best PS possible. There is a tradeoff between the complexity (the NSVs) and the margin (C) that affects the accuracy of a support vector machine (SVM) [46]. Over- thing may occur with a high number of support vectors, whereas wrong sample classification can occur with a big value of C intended to improve generalization capacity. The choice of SVM con duration allows one to adjust this trade-off (C, kernel type and kernel parameters). Specifically, we see the accuracy and complexity (number of support vectors (NSV) obtained across the training examples) as competing goals in this study [46]:

The accuracy was excellent. The precision indicates how well a problem instance was classified. K-fold cross-validation is a method that may be used to determine this (CV), where the provided instance is partitioned into K equal-sized disjoint sets D1, DK. This SVM is trained K times, one for each configuration (2). Each cycle consists of K iterations, where K minus one sets are utilized for training and the remaining set is used for testing. Over the course of K training cycles, the error (err) is the typical occurrence of incorrectly classified input. Complexity. The complexity is the upper constraint on the predicted number of mistakes or the number of support vectors (NSV). The SVM configuration 2 contains the variables used for making the call (C, kernel type and kernel parameters). Each decision variable has a set of allowed values, or boundaries. The optimization problem (m D 2) may be written as [46]:

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



The procedure (shown in Figure 1) that will be used. where err is the number of incorrectly classified data sets and NSV is the total number of support vectors.

IV. METHODOLOGY

Figure 1 shows the _orchard for the suggested approach (called HH-SVM hereafter). Both the SVM and a hyper-heuristic framework make up this methodological approach. The hyper-heuristic framework's primary function is to provide con duration (C, kernel type, and kernel parameters) for the SVM to use. The cost function (average values of err and NSV) is calculated by the SVM using the resulting con duration, and then sent to the hyper heuristic framework for further processing. There are a predetermined number of times this procedure is carried out. Subsequent sections elaborate on the fundamental features of the proposed hyper-heuristic framework.

HYPER-HEURISTIC FRAMEWORK: A PROPOSAL

Using the hyper-heuristic architecture shown in Figure 2, we can pick the best configurations to test. It consists of a high-level strategy and a set of heuristics at a lower level [11]. This high-level tactic avoids the solution space in favors of the heuristic space. Every time around, the high-level strategy takes a gander at the available low-level heuristics, picks one, applies it to the current solution to generate a new solution, and then determines whether or not to accept the new solution. The low-level heuristics are a group of heuristics that work directly on the problem's solution space [39].

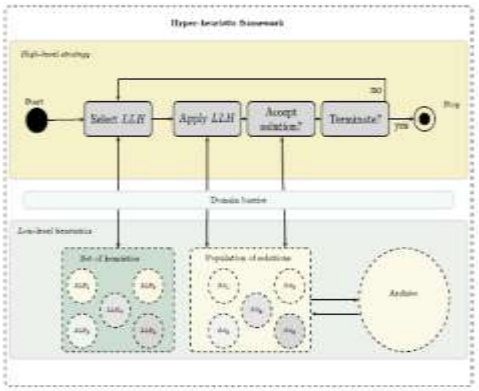


FIGURE 2. Hyper-heuristic framework.

Representation of the Solution in Figure 3.
Strength in Convergence of the Dominant Method. When applied to a set of solutions, the decomposition method enables whereas the dominant method relies on past research. Using the previous population, the archive, or both, the hyper heuristic framework creates a new population of solutions. Therefore, the search is able to strike a healthy balance between convergence and variety. It's important to remember that if you want great variety, you want to spread the solutions out as far as possible along PF, but if you want strong convergence, you want to minimize the distances between the solutions and PF.
In the parts that follow, we'll go through the fundamentals of the hyper-heuristic framework we've presented.

IMAGING THE SOLUTION

Each solution in our framework stands in for one possible SVM configuration (2). Figure 3 depicts a one-dimensional array. The margin parameter (or penalty) is denoted by C in this figure; KF is the index of the chosen kernel function; and k1, K2, kef are its parameters.

INITIALIZATION OF THE POPULATION

The PS is seeded with a random set of solutions. To generate a random number for each choice variable in a given solution (x), we use the following equation:

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

where (I)I is the index of the decision variable (d)d is the total number of decision variables (p)p is the index of the solution (josh) and (josh)j is the population size Randi For the it decision variable, I (0; 1) yields a random value in the range [0,1], lp I is the lower limit on that value, and upi is the upper bound.

DETERMINING FITNESS, PART D

According to the formula, each solution in the population is given a score that reflects how well it performs in comparison to the other solutions in the population. In this study, the MOEA/D methodology is used to the SVM con duration selection issue to optimize for multiple objectives simultaneously. First, a given multi-objective optimization issue is deconstructed into a set of single-objective sub problems, and then, these sub problems are solved jointly [57]. Following the steps outlined in [57], MOEA/D applies a secularization function to an issue in order to break it down into a set of salaried single-objective sub-problems.

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

HIGH-LEVEL STRATEGY

The primary function of a high-level strategy is to mechanize the heuristic selection process [39]. Our recommended strategic the following actions constitute a strategy.

(1) PICK

Step one, "selection," entails making a choice from among the available heuristics. For our purposes, we use Multi-Armed Bandit (MAB) [39] as a heuristic online-selection method. All heuristic results are recorded and analyzed in MAB to choose which one to utilize. An empirical reward quid and a confidence level in are linked to each heuristic. When using this heuristic throughout a search, you may expect to get, on average, a reward equal to quid. It is preferable to have a larger empirical reward. The number of times the it heuristic has been used in the past is the confidence level in. Using these two inputs, MAB determines the confidence interval for each heuristic and then choose the greatest value using the following formula (Equation (11)):

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

For our framework, we use the notation fly 1;::: ; LLH ng, where n is the total number of heuristics. Time increments are denoted by the index t, which also serves as the current repeating the search process. The confidence interval will not be too influenced by either the empirical reward or the confi dence level, and c is a scaling factor that regulates the balance between these two factors. For instance, a heuristic with a high payoff but little use should be favoured less than one with a slightly lower reward but higher usage rate.

How to compute the empirical reward qi(t):

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

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

From the beginning of the search up to the current iteration t, the RI (t) component represents the entire improvement brought by heuristic I.

2) APPLY

In the application phase, two activities take place:

Determine the best course of action. Using this process, we may choose which potential partners to include in the breeding pool. Our proposed method for solution selection draws from the best features of both the decomposition and Pareto (dominance) techniques. Each of the currently available solutions in MOEA/D stands for a different sub-problem. Combining decomposition and dominance allows us to optimize each sub-problem using data either from just its neighboring sub-problems with probability pn, from its neighboring sub-problems plus the archive with probability pan, or from the archive alone with probability pa. The Euclidean distances between any two solutions based on their weight vectors are used to establish an axed set of neighboring solutions for each sub-problem.

It makes use of heuristics. In this step, a mating pool is generated and the chosen heuristic is applied to it to develop an improved set of solutions.

Three, Take the Answer the validity of the newly developed solutions is evaluated in the acceptance phase. We begin by drawing parallels between solution x and the related sub problems Hence, if the

secularization function get(x; _) > get(y; _), then x will be used instead of y. Following this, we include no dominated approaches into the repository for future reference.
#4) ENDS In this way, the search procedure is completed. If the maximum number of iterations has been reached, this step stops the search and delivers the set of no dominated solutions. In every other case, it begins a new cycle.

LOW-LEVEL HEURISTICS

The low-level heuristics (LLHs) are a group of rules that may be applied directly to a solution and are tailored to a particular situation. One or more solutions are fed into each LLH, and they are subsequently altered in order to provide a novel answer to the problem. Within the provided framework, we use many heuristics sets in our study. It has been shown that these heuristics work well for many issues and even for various phases of the same problem. The goal in selecting these heuristics is to broaden the search's scope and make use of a variety of search techniques.
Here is a list of the heuristics [20]:

1) **Parametrised Gaussian Mutation**

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

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

2) **Differential Mutation_1**

$$x = x_1 + F \times (x_2 - x_3) \quad (15)$$

3) **Differential Mutation_2**

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

4) **Differential Mutation_3**

$$x = x_1 + F \times (x_1 - x_2) + F \times (x_3 - x_4) \quad (17)$$

Where x1, x2, x3, x4, and x5 represent distinct solutions chosen from the mating pool using the procedure outlined in IV-E.2. F is a value of the scaling factor, which are _axed to 0.9 for the purpose of this research.

5) **Arithmetic Crossover**

$$x = \lambda \times x_1 + (1 - \lambda) \times x_2 \quad (18)$$

6) **Polynomial Mutation**

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

and

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

V. EXPERIMENTAL SETUP

This section provides a brief overview of the benchmark examples that were used to evaluate the proposed framework and its parameters.

BENCHMARK INSTANCES A.

To evaluate our methodology, we applied it to two cyber security situations of varying sizes and shapes.

CLASSIFICATION OF LARGE AMOUNTS OF DATA FROM MICROSOFT MALWARE

The Microsoft Malware Big Data Classification Problem, created for BIG 2015 on Kaggle, is used for the first time in an experimental assessment. 1 Microsoft supplied 500 GB of data of known malware _les representing a combination of 9 families (classes) for use in training and testing.

TABLE 2: The default values for our framework's parameters; the training set contains a total of 10868 malwares.

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

The test collection contains 10783 different types of malware. The. Bytes files are the samples, while treble files in assembly language are the disassembled versions. (The filename) ends in '.as,' which indicates that it is an assembly language file. The end aim is to reduce the log loss function below by training the classification algorithm using the training data to properly classify each of the testing samples into one of the 9 categories (malware families).

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

Where N stands for the total number of training samples, M for the total number of classes, log for the natural logarithm, and yij for a true label with a value of 1 if I is in class j and a value of 0 otherwise. The chance that item I is in set j is denoted by the estimate pij. Visit Kaggle for more information.

INTRUSION DETECTION FOR ANOMALY

The NSL-KDD2 anomalous intrusion detection instances were employed in the second experimental assessment. The KDDCUP99 dataset was compiled by analyzing incoming network tram c, and some of those records are included in NSL-KDD as well.

Numerous academics have used NSL-KDD to create innovative IDSs for networks (NIDs). Classified as normal or anomalous, the NSL-KDD issue instance has a total of 125,973 training samples and 22,544 testing samples (i.e., a network attack).

ADJUSTING THE PARAMETERS

There are several parameters of the suggested framework that must be established in advance. We did some preliminary research to determine appropriate settings for these knobs.

We tried several settings while leaving all other variables at their default levels. In Table 2, we detail the many parameter settings we considered and the values we ultimately settled on.

VI. RESULTS AND COMPARISONS

Here, we report on the experiments we ran to test the efficacy of the paper's suggested HH-SVM system. We tested two different experiments. The _rest evaluation compared HH-SVM to each distinct low-level heuristic. The second experiment compared HH-performance SVM's to that of competing algorithms from the literature.

Results from the six low-level heuristics (LLH1 to LLH6) are compared with those from HH-SVM in Table 3.

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 (LLH1 to LLH6).

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

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

Here, we evaluate the proposed HH-SVM vs. other baseline methods (LLH). We want to analyze the outcomes of the suggested hyper-heuristic framework and its benefits. Search efficiency while using a number of LLHs. Each LLH was put through its own battery of tests to this purpose. Seven distinct algorithms—HH-SVM, LLH1, LLH2, LLH3, LLH4, LLH5, and LLH6—produced the findings. All algorithms were run in the same environment, and the same foundational elements were applied to both problems (BIG 2015 and NSL-KDD). Table 3 compares the average outcomes from 31 separate simulations. Comparing BIG 2015 findings using log loss, where lower numbers are better (20), and comparing NSL-KDD results using accuracy, where higher values are better, yields different conclusions. When comparing the performance of different algorithms, the table highlights the top performers in bold. Results show that on both BIG 2015 and NSL-KDD, HH-SVM performs better than the other algorithms (LLH1, LLH2, LLH3, LLH4, LLH5, and LLH6). Table 4 details the support vector (NSV) counts for HH-SVM and the other methods examined in both cases; smaller NSV counts are preferable. This table shows that when compared to LLH1, LLH2, LLH3, LLH4, LLH5, and LLH6, the NSV values obtained by the proposed HH-SVM architecture are lower for both BIG 2015 and NSL-KDD. These successes verify the need for the suggested hyper-heuristic architecture and the heuristics pool (LLHs).

We also ran statistical analyses using the Wilcox on test at the 0.05 significance level to double-check these findings.

Comparison of p-values between HH-SVM and LLH1 Table 5 details the results for LLH2, LLH3, LLH4, LLH5, and LLH6. When compared to the other algorithms, HH-SVM has a lower p-value in TABLE 5; hence it is clearly the better choice. When

compared to the separate low-level heuristics, HH-SVM has lower p-values.

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 log loss results of HH-SVM and other algorithms.

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

Relative to. If the p-value is more than 0.05, it means that the HH-SVM framework we propose does not significantly outperform existing methods. The table makes it easy to observe that all p-values are less than 0.05; hence HH-SVM is clearly the winner of BIG 2015 and NSL-KDD when compared to LLH1, LLH2, LLH3, LLH4, LLH5, and LLH6.

B. An Evaluation of HH-SVM and Competing Algorithms

Here we compare HH-findings SVM's to those found in the published literature. We compare the following algorithms for BIG 2015:

- XGBoost (AE)
- [RF] Random Forest
- XGBoost with Optimization (OXB) [1]

The accuracy results produced by HH-SVM are compared to those obtained by the following algorithms when applied to the NSL-KDD instance: a naive Bays tree with a Gaussian distribution [48] [27] Fuzzy Classifier (FC) The _Decision Tree (DT) [33] Table 6 and Table 7 summarizes the outcomes of HH-SVM and the other methods for the BIG 2015 and NSL-KDD issue situations, respectively. Results for BIG 2015 are shown in Table 6 as log loss values, as is customary in the literature; in Table 7, however, the methods are compared using the accuracy metric. A lower number suggests greater performance in log loss comparisons, whereas a larger value indicates better performance in accuracy comparisons. In both tables, the top-performing algorithm was highlighted in bold. TABLE 7 shows that compared to AE and RF, HH-SVM has the lowest log loss value (TABLE 6). Accuracy studies comparing HH-SVM with other popular methods.

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

VII. CONCLUSION

As part of this effort, we put forward a hyper-heuristic SVM optimization framework for tackling the challenges of cyber security in the age of big data. Specifically, we formalized the SVM con duration procedure as a dual-objective optimization issue whereby precision and model complexity is seen as antagonistic goals. The suggested hyper-heuristic framework is effective in resolving this issue of bi-objective optimization. The Pareto set of optimal configurations is approximated by combining the benefits of decomposition- and Pareto-based techniques, both of which are included into the framework. Both Microsoft's malware big data classification and anomalous intrusion detection serve as benchmarks for testing our methodology. Experimental findings show that the suggested framework may achieve comparable or even better outcomes compared to previous algorithms, proving its efficacy and potential.

REFERENCES

[1] M. Hamada, D. Ulyanovsk, S. Semenov, M. Tro_mov, and G. Jacinto, ``Novel feature extraction, selection and fusion for effective malware family classi_cation,'' in Proc. 6th ACM Conf. Data Appl. Secure. Privacy, 2016, pp. 183_194.

[2] A. V. Aho and M. J. Cora sick, ``Efficient string matching: An aid to bibliographic search,'' Common. 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. Ayah, M. Cherie, and C. Y. Seen, ``Automatic model selection for the optimization of SVM kernels," Pattern Recognition. vol. 38, no. 10, pp. 1733_1745, 2005.
[5] Y. Bao, Z. Hub, and T. Xing, ``A PSO and pattern search based mimetic 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. Ferias, ``A hyper-heuristic evolutionary algorithm for automatically designing decision-tree algorithms," in Proc. 14th Annul. Conf. Genet. Evil. Compute.,
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 Annul. ACM Sump. 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. Cargo and F. Eisenhower, Eds. Vole 609. New York, NY, USA: Humana Press, 2010, pp. 223_239.
[10] D. Brimley, C. Hart wig, Z. Liang, J. Newsome, D. Song, and H. Yin, ``Automatically identifying trigger-based behavior in malware," in Boot- net 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. Oscan, and J. R. Woodward, ``A classi_cation of hyper-heuristic approaches," in Handbook of Met heuristics (International Series in Operations Research & Management Science), vol. 146, M. Gendered and J. Y. Putin, Eds. Boston, MA, USA: Springer, 2010.
[12] A. Chalimourda, B. Schölkopf, and A. J. Smola, ``experimentally optimal _ in support vector regression for different noise models and parameter settings," Neural Newt. 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. Intel. 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 Newt. Appl., vol. 19, no. 2, pp. 171_209, Apr. 2014.
[15] N. Cristianini and J. Shaw-Taylor, an Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge, U.K.: Cambridge Univ. Press, 2000.

Author details:

MARGANI PRIYANKA
19RG1A0534
Department of Computer Science and Engineering, Malla Reddy College of Engineering For Women, Maisammaguda, Hyderabad, Telangana

KALUGATLA UDAYAKEERTHI
19RG1A0525
Department of Computer Science and Engineering, Malla Reddy College of Engineering For Women, Maisammaguda, Hyderabad, Telangana

KONDA AMULYA
19RG1A0528
Department of Computer Science and Engineering, Malla Reddy College of Engineering For Women, Maisammaguda, Hyderabad, Telangana

GADWALA SWETHA
19RG1A0517
Department of Computer Science and Engineering, Malla Reddy College of Engineering For Women, Maisammaguda, Hyderabad, Telangana

Dr VAKA MURALI MOHAN
Guide
Principal & Professor, Department of Computer Science and Engineering, Malla Reddy College of Engineering For Women, Maisammaguda, Hyderabad, Telangana