Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 8 (2013) pp. 076-085


# A Comparison about Evolutionary Algorithms for Optimum-Path Forest Clustering Optimization

**Kelton A. P. da Costa, Clayton R. Pereira, Luis A. M. Pereira, Rodrigo M. Nakamura and João Paulo Papa**

Unesp - Univ Estadual Paulista, Department of Computing,
Av. Eng. Luiz Edmundo Carrijo Coube, 14-01, Bauru, Brazil
{*kelton.costa,clayton,rodrigo.mizobe,luis.pereira,papa*}*@fc.unesp.br*

***Abstract***: **TIn this paper we deal with the problem of boosting the Optimum-Path Forest (OPF) clustering approach using evolutionary-based optimization techniques. As the OPF classifier performs an exhaustive search to find out the size of sample's neighborhood that allows it to reach the minimum graph cut as a quality measure, we compared several optimization techniques that can obtain close graph cut values to the ones obtained by brute force. Experiments in two public datasets in the context of unsupervised network intrusion detection have showed the evolutionary optimization techniques can find suitable values for the neighborhood faster than the exhaustive search. Additionally, we have showed that it is not necessary to employ many agents for such task, since the neighborhood size is defined by discrete values, with constrain the set of possible solution to a few ones.**

***Keywords***: Optimum-Path Forest, Evolutionary Algorithms, Intrusion Detection, Computer Networks.

## I. Introduction

Data clustering is one of the most difficult tasks in pattern recognition, since none information about data is available. Sometimes, when you have a small portion of labeled data, it can be used by semi-supervised learning algorithms to improve the performance.

Although several research has been done considering supervised methods, it is not so easy to find free-labeled datasets, being most part of the problems still unlabeled. Among them, we can cite medical image processing, remote sensing and intrusion detection in computer networks. Since the latter context represents a serious issue in a network environment, system administrators have the challenge to avoid unauthorized access of confidential and privileged information. In order to tackle this problem, intrusion detection systems (IDS) have been developed to scan the network activity and also to detect such intrusion attacks.

Generally, network intrusion detection relies on machine learning algorithms to recognize an attack signature given by a human expert or detect deviations from the normal observed data. As such, some very interesting works have faced this problem in the last years. For example, Zhong *et al*. [1] compared $k$-means, Mixture-of-Spherical Gaussians, Self-Organizing Maps and Neural-Gas in network intrusion detection. Portnoy *et al*. [2] employed traditional clustering techniques for anomaly detection, and Ye and Li [3] proposed an incremental algorithm for the same task. Guan and Ghorbani [4] presented a new clustering technique to handle intrusion detection called Y-means, and Eskin [5] addressed anomaly detection by learning the samples' probability distribution. More recently, Wu and Banzhaf [6] have presented an interesting review about supervised and unsupervised methods applied for intrusion detection in computer networks. Chaki and Chaki [7] addressed the same problem for mobile networks, Sen [8] presented a distributed intrusion detection architecture for wireless-based Ad-hoc networks. Cai et al. [9] proposes a new anomaly detection method in order to deal with the low detection rate and high false alarms using the Ball Vector Machine (BVM) and Extreme Learning Machine (ELM). Tartakovsky et al. [10] proposed a novel score-based multi-cyclic detection algorithm, which is based on the so-called Shiryaev-Roberts procedure. Such approach has showed better results than other anomaly detection schemes.

Although we have several unsupervised pattern recognition techniques, Rocha et al. [11] have proposed a new clustering algorithm known as Optimum-Path Forest (OPF), which is a framework of classifiers based on graph partition methods. The OPF, which has also a supervised version [12], has demonstrated to improve traditional Mean shift [13] in terms of robustness to data clustering and number of irrelevant clusters. This method works by modeling the dataset as a $k$-nn graph, and the main idea is to find some key nodes (prototypes) and to partition this graph into optimum-path trees (OPTs) through a competition between them according to some path-cost function. Thus, each OPT defines a cluster which represents a group of similar samples that are strongest connected. Additionally, it is important to highlight the robustness of OPF clustering has been already evaluated in the context intrusion detection in computer networks [14].

However, a main drawback of OPF clustering is to find proper values for the $k$ neighborhood to build the $k$-nn graph. The naive version employs an exhaustive search in the interval $[1, k_{max}]$ to find the $k$ value that minimizes the graph cut, which may be prohibitive for large datasets. Generally, in the context of intrusion detection in computer networks, it is not difficult to find datasets with millions of samples. Costa

et al. [15] have proposed a nature-inspired algorithm to find suitable values of $k$ using the Harmony Search (HS) [16], and have validated it in the context of intrusion detection in computer networks. In this paper, we have compared the approach proposed in [15] together with a wide range of evolutionary optimization techniques, such as Particle Swarm Optimization (PSO) [17, 18], Harmony Search, Bat Algorithm (BA) [19], Cuckoo Search (CS) [20, 21], Firefly Algorithm (FFA) [22], Charged System Search (CSS) [23], Gravitational Search Algorithm (GSA) [24]

The remainder of the paper is organized as follows. The OPF theory is stated in Section II, and in Section III, we present the evolutionary Optimization Techniques we have employed in this work. Materials and methods and experimental evaluation results are discussed in Sections IV and IV-B, respectively. Finally, conclusions are stated in Section VI.

## II. Optimum-Path Forest Clustering

The OPF framework is a recent highlight to the development of pattern recognition techniques based on graph partitions. The nodes are data samples, which are represented by their corresponding feature vectors, and are connected according to some predefined adjacency relation. Given some key nodes (prototypes), they will compete among themselves aiming to conquer the remaining nodes. Thus, at the final of the process one has an optimum path forest, which is a collection of optimum-path trees rooted at each prototype.

Therefore, to design an OPF-based classifier, one needs to establish tree parameters: (i) adjacency relation, (ii) path-cost function and (iii) methodology to estimate prototypes. Depending on the way one chooses them, a different classifier can be obtained. Thus, the OPF is a flexible way to create pattern recognition techniques based on the computation of an optimum-path forest over a graph induced by data samples.

### A. Background Theory

Let $\mathcal{Z}$ be a dataset such that for every sample $s \in \mathcal{Z}$ there is a feature vector $\vec{v}(s)$. Let $d(s,t)$ be the distance between $s$ and $t$ in the feature space (e.g., $d(s,t) = \|\vec{v}(t) - \vec{v}(s)\|$). A graph $(\mathcal{Z}, \mathcal{A})$ is defined such that the arcs $(s,t) \in \mathcal{A}$ connect $k$-nearest neighbors in the feature space. The arcs are weighted by $d(s,t)$ and the nodes $s \in \mathcal{Z}$ are weighted by a density value $\rho(s)$:

$$\rho(s) = \frac{1}{\sqrt{2\pi\sigma^2}|\mathcal{A}(s)|} \sum_{\forall t \in \mathcal{A}(s)} \exp\left(\frac{-d^2(s,t)}{2\sigma^2}\right), \quad (1)$$

where $|\mathcal{A}(s)| = k$, $\sigma = \frac{d_f}{3}$, and $d_f$ is the maximum arc weight in $(\mathcal{Z}, \mathcal{A})$. This parameter considers all nodes for density computation, since a Gaussian function covers most samples within $d(s,t) \in [0, 3\sigma]$.

The traditional method to estimate a probability density function (pdf) is by Parzen-window. Equation 1 can provide a Parzen-window estimation based on isotropic Gaussian kernel when we define the arcs by $(s,t) \in \mathcal{A}$ if $d(s,t) \leq d_f$. This choice, however, presents problems with the differences in scale and sample concentration. Solutions for this problem led to adaptive choices of $d_f$ depending on the region of the feature space [25]. By taking into account the $k$-nearest neighbors, we are handling different concentrations and reducing the scale problem finding the best value of $k$ within $[1, k_{\max}]$, for $1 \leq k_{\max} \leq |\mathcal{Z}|$.

The solution proposed by Rocha et al. [11] to find $k$ considers the minimum graph cut provided by the clustering results for $k \in [1, k_{\max}]$, according to a measure $C(k)$ suggested by Shi and Malik [26]:

$$C(k) = \sum_{i=1}^{c} \frac{W_i'}{W_i + W_i'}, \quad (2)$$

$$W_i = \sum_{\forall (s,t) \in \mathcal{A} | \lambda(s) = \lambda(t) = i} \frac{1}{d(s,t)}, \quad (3)$$

$$W_i' = \sum_{\forall (s,t) \in \mathcal{A} | \lambda(s) = i, \lambda(t) \neq i} \frac{1}{d(s,t)}, \quad (4)$$

where $\lambda(t)$ is the label of sample $t$, $W_i'$ uses all arc weights between cluster $i$ and other clusters, and $W_i$ uses all arc weights within cluster $i = 1, 2, \ldots, c$.

We say that a path $\pi_t$ is a sequence of adjacent samples starting from a root $R(t)$ and ending at a sample $t$, being $\pi_t = \langle t \rangle$ a trivial path and $\pi_s \cdot \langle s,t \rangle$ the concatenation of $\pi_s$ and arc $(s,t)$. We assign to each path $\pi_t$ a cost $f(\pi_t)$ given by a path-value function $f$. A path $\pi_t$ is considered optimum if $f(\pi_t) \leq f(\tau_t)$ for any other path $\tau_t$.

Among all possible paths $\pi_t$ with roots on the maxima of the pdf, we wish to find a path whose the lowest density value along it is maximum. Each maximum should then define an influence zone (cluster) by selecting the strongest connected samples , than to any other maximum. Formally, we wish to maximize $f(\pi_t)$ for all $t \in \mathcal{Z}$ where:

$$f(\langle t \rangle) = \begin{cases} \rho(t) & \text{if } t \in \mathcal{R} \\ \rho(t) - \delta & \text{otherwise} \end{cases}$$

$$f(\langle \pi_s \cdot \langle s,t \rangle \rangle) = \min\{f(\pi_s), \rho(t)\}, \quad (5)$$

for $\delta = \min_{\forall (s,t) \in \mathcal{A} | \rho(t) \neq \rho(s)} |\rho(t) - \rho(s)|$ and $\mathcal{R}$ being a root set with one element for each maximum of the pdf. Higher values of delta reduce the number of maxima. We are setting $\delta = 1.0$ and scaling real numbers $\rho(t) \in [1, KMax]$ in this work. The OPF algorithm maximizes $f(\pi_t)$ such that the optimum paths form an optimum-path forest — a predecessor map $P$ with no cycles that assigns to each sample $t \notin \mathcal{R}$ its predecessor $P(t)$ in the optimum path from $\mathcal{R}$ or a marker $nil$ when $t \in \mathcal{R}$. Algorithm 1 implements this procedure.

Algorithm 1 identifies one root in each maximum of the pdf ($P(s) = nil$ in Line 4 implies $s \in \mathcal{R}$), assigns to each root a distinct label in Line 5, and computes the influence zone (cluster) of each root as an optimum-path tree in $P$, such that the nodes of the tree receive the same label of its root in a map $L$ (Line 9). It also outputs the optimum path-value map $V$ and forest $P$ in Line 11. It is more robust than the Mean shift because it does not depend on pdf gradients, it uses a $k$-nn graph and assigns a single label per maximum, even when the maximum is a connected component in $(\mathcal{Z}, \mathcal{A})$. Therefore, it is more general because the choice of $f(\langle t \rangle)$ can reduce irrelevant maxima (clusters).

**Algorithm 1** – OPF CLUSTERING

INPUT: Graph $(\mathcal{Z}, \mathcal{A})$ and function $\rho$.
OUTPUT: Optimum-path forest $P$, cost map $C$ and label map $L$.
AUXILIARY: Priority queue $Q$, variables $tmp$ and $l \leftarrow 1$.

*For all* $s \in \mathcal{Z}$, *set* $P(s) \leftarrow nil$, $C(s) \leftarrow \rho(s) - \delta$, *insert* $s$ *in* $Q$.
*While* $Q$ *is not empty, do*
  *Remove from* $Q$ *a sample* $s$ *such that* $C(s)$ *is maximum.*
  *If* $P(s) = nil$, *then*
    *Set* $L(s) \leftarrow l$, $l \leftarrow l + 1$, *and* $C(s) \leftarrow \rho(s)$.
  *For each* $t \in \mathcal{A}(s)$ *and* $C(t) < C(s)$, *do*
    *Compute* $tmp \leftarrow \min\{C(s), \rho(t)\}$.
    *If* $tmp > C(t)$ *then*
      *Set* $L(t) \leftarrow L(s)$, $P(t) \leftarrow s$, $C(t) \leftarrow tmp$.
      *Update position of* $t$ *in* $Q$.
*Return a classifier* $[P, C, L]$.

A sample $t \in \mathcal{Z}$ can be classified in one of the clusters by identifying which root would offer it an optimum path as though it were part of the forest. By considering the $k$-nearest neighbors of $t$ in $\mathcal{Z}$, we can use Equation 1 to compute $\rho(t)$, evaluate the optimum paths $\pi_s \cdot \langle s, t \rangle$, and select the one that satisfies

$$C(t) = \max_{\forall (s,t) \in \mathcal{A}} \{\min\{C(s), \rho(t)\}\} \tag{6}$$

Let the node $s^* \in \mathcal{Z}$ be the one that satisfies Equation 6. The classification simply assigns $L(s^*)$ as the cluster of $t$.

## III. Evolutionary Optimization Background

In this section we present the evolutionary optimization techniques we have employed in this work.

### A. Bat Algorithm

Bats have an amazing capability to detect prey, avoid obstacles, and locate their roosting crevices even in the complete darkness by emitting a very loud sound pulse and listening for the echo that bounces back from the surrounding objects. In addition, bats seem to be able to discriminate targets by the variations of the Doppler effect induced by the wing-flutter rates of the target insects [27]. Inspired by those phenomena, Yang [28] proposed a meta-heuristic algorithm, the Bat Algorithm (BA), to behave as a population of bats looking for prey/foods using their capability of echolocation.
Firstly, each bat $i$ is modeled with a position $\vec{x}_i \in \Re^n$, a velocity $\vec{v}_i \in \Re^n$ and a frequency $f_i$. For each time step $t$, as $t = 1, ..., M$, being $M$ the maximum number of iterations, the movement of a virtual bat $i$ at dimension $j$ is given by updating its velocity and position using Equations 7 to 9, as follows:

$$f_i = f_{min} + (f_{max} - f_{min})\beta, \tag{7}$$

$$v_i^j(t) = v_i^j(t-1) + (\hat{x} - x_i^j(t-1))f_i, \tag{8}$$

$$x_i^j(t) = x_i^j(t-1) + v_i^j(t), \tag{9}$$

where $\beta$ denotes a randomly generated number $\in [0, 1]$. The result of $f_i$ (Equation 7) is used to control the pace and range of the movement of the bats. The variable $\hat{x}$ represents the current best global solution, and $f_{min}$ and $f_{max}$ stand for the minimum and maximum frequency values, which are specified by the user.
Note that Bat Algorithm works similarly to the standard Particle Swarm Optimization (see Section III-G) as $f_i$ essentially controls the pace and range of the movement of the bats. To mitigate possible local solutions around the known global maximum, the Bat Algorithm employs a random walk. Basically, one solution is selected among the current best solutions, and then the random walk is applied to locally generate a new solution for each bat:

$$x_{new} = x_{old} + \epsilon \overline{A}(t), \tag{10}$$

in which $\overline{A}(t)$ is the average loudness of all the bats at the time $t$, and $\epsilon \in [-1, 1]$ is a random number. For each iteration of the algorithm, the loudness $A_i$ and the emission pulse rate $r_i$ have to be updated, as follows:

$$A_i(t+1) = \alpha A_i(t) \tag{11}$$

and

$$r_i(t+1) = r_i(0)[1 - e^{-\gamma t}], \tag{12}$$

where $\alpha$ and $\gamma$ are constants. At the first step of the algorithm, the emission rate $r_i(0)$ and the loudness $A_i(0)$ for each bat should be different and randomly chosen. For instance, $A_i(0)$ could be $\in [1, 2]$ and $r_i(0)$ could be $\in [0, 1]$. However, the loudness and the emission rates will be updated only if the new solutions are improved, which means that these bats are moving towards the optimal solution.

### B. Cuckoo Search

The parasite behavior of some species of Cuckoo are extremely intriguing. These birds can lay down their eggs in a host nests, and mimic external characteristics of host eggs such as color and spots. In case of this strategy is unsuccessful, the host can throw the cuckoo's egg away, or simply abandon its nest, making a new one in another place. Based on this context, Yang and Deb [29] have developed a novel evolutionary optimization algorithm named as Cuckoo Search (CS), and they have summarized CS using three rules, as follows:

1. Each cuckoo choose a nest randomly to lays eggs.
2. The number of available host nests is fixed, and nests with high quality of eggs will carry over to the next generations.
3. In case of a host bird discovered the cuckoo egg, it can throw the egg away or abandon the nest, and build a completely new nest.

Algorithmically, each host nest $n$ is defined as an agent which can contain a simple egg $x$ (unique dimension problem) or more than one, when the problem concerns to multiple dimensions. CS starts by placing the nest population randomly in the search space. In each algorithm iteration, the nests are updated using random walk via Lévy flights:

$$x_i^j(t) = x_i^j(t-1) + \alpha \oplus Levy(\lambda) \tag{13}$$

and

$$Levy \sim u = s^{-\lambda}, (1 < \lambda \leq 3), \tag{14}$$

where $s$ is step size, and $\alpha > 0$ is the step size scaling factor/parameter. Here the entrywise product $\oplus$ is similar to those used in PSO, and $x_i^j$ stands for the $j^{\text{th}}$ egg (feature) at nest $i$ (solution), $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, d$. The Lévy flights employ a random step length which is drawn from a Lévy distribution. Therefore, the CS algorithm is more efficient in exploring the search space as its step length is much longer in the long run [29]. Finally, the nests which have eggs with the lowest quality, are replaced to new ones according to a probability $p_a \in [0, 1]$.

### C. Charged System Search

The governing Coulomb's law is a physics law used to describe the interactions between electrically charged particles. Let a charge be a solid sphere with radius $r$ and uniform density volume. The attraction force $F_{ij}$ between two spheres $i$ and $j$ with total charges $q_i$ and $q_j$ is defined by:

$$F_{ij} = \frac{k_e q_i q_j}{d_{ij}^2}, \tag{15}$$

where $k_e$ is a constant called the Coulomb constant and $d_{ij}$ is the distance between the charges.
Based on aforementioned definition, Kaveh and Talatahari [23] have proposed a new metaheuristic algorithm called Charged System Search (CSS). In this algorithm, each Charged Particle (CP) on system is affected by the electrical fields of the others, generating a resultant force over each CP, which is determfurtherined by using the electrostatics laws. The CP interaction movement is determined using Newtonian mechanics laws. Therefore, Kaveh and Talatahari [23] have sumarized CSS over the following definitions:

- *Definition 1*: The magnitude of charge $q_i$, with $i = 1, 2, ..., n$, is defined considering the quality of its solution, i.e. objective function value $fit(i)$:

$$q_i = \frac{fit(i) - fitworst}{fitbest - fitworst}, \tag{16}$$

  where $fitbest$ and $fitworst$ denote, respectively, the so far best and the worst fitness of all particles. The distance $d_{ij}$ between two CPs is given by the following equation:

$$d_{ij} = \frac{\|\vec{x}_i - \vec{x}_j\|}{\|\frac{\vec{x}_i - \vec{x}_j}{2} - \vec{x}_{best}\| + \epsilon}, \tag{17}$$

  in which $\vec{x}_i$, $\vec{x}_j$ and $\vec{x}_{best}$ denote the positions of the $i$th, $j$th and the best current CP respectively, and $\epsilon$ is a small positive number to avoid singularities.

- *Definition 2*: The initial position $x_{ij}(0)$ and velocity $v_{ij}(0)$, for each $j$th variable of the $i$th CP, with $j = 1, 2, \ldots, m$, is given by:

$$x_{ij}(0) = x_{i,min} + \theta(x_{i,max} - x_{i,min}) \tag{18}$$

  and

$$v_{ij}(0) = 0, \tag{19}$$

  where $x_{i,max}$ and $x_{i,min}$ represents the upper and low bounds respectively, and $\theta \sim U(0, 1)$.

- *Definition 3*: For maximization problem, the probability of each CP moves toward others CPs is given as follow:

$$p_{ij} = \begin{cases} 1 & \text{if } \frac{fit(j) - fitworst}{fit(i) - fit(j)} > \theta \vee fit(i) > fit(j) \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

- *Definition 4*: The value of the resultant force acting on a CP $j$ is defined as:

$$r = 0.1\text{max}(x_{i,max} - x_{i,min}) \tag{21}$$

$$F_j = q_j \sum_{j,i \neq j} \left( \frac{q_i}{r^3} \cdot d_{ij} \cdot c_1 + \frac{q_i}{d_{ij}^2} \cdot c_2 \right) p_{ij}(\vec{x_i} - \vec{x_j}), \tag{22}$$

  where $c_1 = 1$ and $c_2 = 0$ if $d_{ij} < r$, otherwise $c_1 = 0$ and $c_2 = 1$.

- *Definition 5*: The new position and velocity of each CP is given by

$$\vec{x}_j(t) = \theta_{j1} \cdot k_a \cdot F_j + \theta_{j2} \cdot k_v \cdot \vec{v}_j(t-1) + \vec{x}_j(t-1) \tag{23}$$

  and

$$\vec{v}_j(t) = \vec{x}_j(t) - \vec{x}_j(t-1), \tag{24}$$

  where $k_a = 0.5(1 + \frac{t}{T})$ and $k_v = 0.5(1 - \frac{t}{T})$ are the acceleration and the velocity coefficients respectively, being $t$ the actual iterations and $T$ the maximum number of iterations.

- *Definition 6*: A number of the best so far solutions is saved using a Charged Memory (CM). The worst solutions are excluded from CM, and better new ones are included to the CM.

### D. Firefly Algorithm

The Firefly Algorithm was proposed by Yang [22] and it is derived from the flash attractiveness of fireflies for mating partners (communication) and attracting potential preys. The brightness of a firefly is determined by some objective function and the perceived light intensity $I$ depends on the distance $d$ from its source, as follows:

$$I = I_0 e^{-\iota d} \tag{25}$$

where $I_0$ is the original light intensity and $\iota$ stands for the light absorption coefficient.
As a firefly's attractiveness is proportional to the light intensity seen by adjacent fireflies, we can now define the attractiveness $\beta$ of a firefly by

$$\beta = \beta_0 e^{-\iota d^2} \tag{26}$$

where $\beta_0$ is the attractiveness at $d = 0$.
A firefly $i$ is attracted to another firefly $k$ with a better fitness value, and moves according to:

$$x_i^j(t+1) = x_i^j(t) + \beta_0 e^{-\iota d_{i,k}^2}(x_k^j - x_i^j) + \phi \left( \sigma_i - \frac{1}{2} \right), \tag{27}$$

where the second term states the attraction between both fireflies, $d_{i,k}^2$ stands for the distance between fireflies $i$ and $k$, $\phi$ is a randomization factor and $\sigma_i \sim U(0, 1)$.

### E. Gravitational Search Algorithm

Rashedi et al. [24] proposed an optimization algorithm based on the gravity, which is one of the fundamental interactions of nature. Their approach, called Gravitational Search Algorithm, models each possible solution as a particle in the universe, which interacts with other ones according to the Newton's law of universal gravitation [30].

Let $p_i$ be a particle in a universe, and $x_i \in \Re^n$ and $v_i \in \Re^n$ its position and velocity, respectively. One can define, at a specific time $t$, the force acting on particle $i$ from particle $k$ in the $j$th dimension as following:

$$F_{ik}^j(t) = G(t)\frac{M_i(t)M_k(t)}{R_{ik}(t)+\tau}(x_k^j(t) - x_i^j(t)), \tag{28}$$

where $R_{ik}(t)$ is the Euclidean distance between particles $i$ and $k$, $M_i$ stands for the mass of particle $i$ and $\tau$ is a small constant to avoid division by zero. $G$ is a gravitational potential, which is given by

$$G(t) = G(t_0)i(\frac{t_0}{t})^\zeta,\ \zeta < 1, \tag{29}$$

in which $\zeta$ is a control parameter [31], $G(t)$ is the value of gravitational potential at time $t$, and $G(t_0)$ is the value of the gravitational potential at the time of the "creation of the universe" that is being considered [31].

To give a stochastic behaviour to Gravitational Search Algorithm, Rashedi et al. [24] assume the total force that acts on particle $i$ in a dimension $j$ as a randomly weighted sum of the forces exerted from other agents:

$$F_i^j(t) = \sum_{k=1, j\neq i}^{m} \sigma_j F_{ik}^j(t), \tag{30}$$

in which $\sigma_i \sim U(0,1)$ and $m$ denotes the number of particles (size of the universe).

The acceleration of a particle $i$ at time $t$ and dimension $j$ is given by

$$a_i^j(t) = \frac{F_i^j(t)}{M_i(t)}, \tag{31}$$

in which the mass $M_i$ is calculated as follows:

$$M_i(t) = \frac{q_i(t)}{\sum_{k=1}^m q_k(t)}, \tag{32}$$

with

$$q_i(t) = \frac{f_i(t) - w(t)}{b(t) - w(t)}. \tag{33}$$

The terms $w(t)$ and $b(t)$ denote, respectively, the particles with the worst and best fitness value. The term $f_i(t)$ stands for the fitness value of particle $i$.

Finally, to avoid local optimal solutions, only the best $b$ masses, i.e., the ones with highest fitness values, will attract others. Let $\mathcal{B}$ be the set of these masses. The value of $b$ is set to $b_0$ at the beginning of the algorithm and decreases with time. Hence, Equation 30 is rewritten as:

$$F_i^j(t) = \sum_{b\in\mathcal{B}, b\neq i} \sigma_b F_{ib}^j(t). \tag{34}$$

The velocity and position updating equations are given by:

$$v_i^j(t+1) = \sigma_i v_i^j(t) + a_i^j(t) \tag{35}$$

and

$$x_i^j(t+1) = x_i^j(t) + v_i^j(t+1), \tag{36}$$

where in which $\sigma_i \sim U(0,1)$.

### F. Harmony Search

Harmony Search (HS) is an evolutionary algorithm inspired in the improvisation process of music players [16]. The main idea is to use the same process adopted by musicians to create new songs to obtain a near-optimal solution for some optimization process. Basically, any possible solution is modeled as a harmony and each parameter to be optimized can be seen as a musical note. The best harmony (solution) is chosen as the one that maximizes some optimization criteria. The algorithm is composed of few steps, as described below:

In order to describe how HS works, an optimization problem is specified in Step 1 as follows:

$$\min\ f(x) \quad \text{subject to } x^j \in X_j, \forall j = 1, 2, \ldots, N, \tag{37}$$

where $f(x)$ is the objective function, $x^j$ and $X_j$, mean, respectively, the design variable $j$ and its set of possible values, and $N$ is the number of design variables. In this work, we have set $X_j \in \{1, k_{max}\}$. As we have only one variable to be optimized, i.e., $k$, we have that $j = 1$.

The HS parameters required to solve the optimization problem (Equation 37) are also specified in this step. They are: the harmony memory size (HMS), the harmony memory considering rate (HMCR), the pitch adjusting rate (PAR), and the stopping criterion. HMCR and PAR are parameters used to improve the solution vector, i.e., they can help the algorithm to find globally and locally improved solutions in the harmony search process (Step 3).

In Step 2, the HM matrix (Equation 38) is initialized with randomly generated solution vectors with their respective values for the objective function:

$$HM = \left[\begin{array}{cccc|c} x_1^1 & x_1^2 & \ldots & x_1^N & f(x_1) \\ x_2^1 & x_2^2 & \ldots & x_2^N & f(x_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{HMS}^1 & x_{HMS}^2 & \ldots & x_{HMS}^N & f(x_{HMS}) \end{array}\right], \tag{38}$$

where $x_i^j$ denotes the decision variable $j$ from harmony $i$.

In Step 3, a new harmony vector $\widehat{x}$=($\widehat{x}^1$,$\widehat{x}^2$ ,..., $\widehat{x}^N$ ) is generated from the HM based on memory considerations, pitch adjustments, and randomization (music improvisation). It is also possible to choose the new value using the HMCR parameter, which varies between 0 and 1 as follows:

$$\widehat{x}^j \leftarrow \begin{cases} \widehat{x}^j \in \left\{x_1^j, x_2^j, \ldots, x_{HMS}^j\right\} & \text{with probability HMCR,} \\ \widehat{x}^j \in X_j & \text{with probability (1-HMCR).} \end{cases} \tag{39}$$

The HMCR is the probability of choosing one value from the historic values stored in the HM, and (1- HMCR) is the probability of randomly choosing one feasible value not limited to those stored in the HM.

Further, every component $j$ of the new harmony vector $\widehat{x}$ is examined to determine whether it should be pitch-adjusted:

$$\text{Pitching adjusting decision for } \widehat{x}^j \leftarrow \begin{cases} \textbf{Yes} \text{ with probability PAR,} \\ \textbf{No} \text{ with probability (1-PAR).} \end{cases} \quad (40)$$

The pitch adjustment for each instrument is often used to improve solutions and to escape from local optima. This mechanism concerns shifting the neighboring values of some decision variable in the harmony. If the pitch adjustment decision for the decision variable $\widehat{x}^j$ is Yes, $\widehat{x}^j$ is replaced as follows:

$$\widehat{x}^j \leftarrow \widehat{x}^j + rb, \quad (41)$$

where $b$ is an arbitrary distance bandwidth for the continuous design variable, and $r$ is a uniform distribution between 0 and 1.

In Step 4, if the new harmony vector is better than the worst harmony in the HM, the latter is replaced by this new harmony.

In Step 5, the HS algorithm finishes when it satisfies the stopping criterion. Otherwise, Steps 3 and 4 are repeated in order to improvise a new harmony again.

### G. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an algorithm modeled on swarm intelligence that finds a solution in a search space based on the social behavior dynamics [17]. Each possible solution of the problem is modeled as a particle in the swarm that imitates its neighborhood based on a objective function. Some definitions consider Particle Swarm Optimization as a stochastic and population-based search algorithm, in which the social behavior learning allows each possible solution to move onto this search space by combining some aspect of the history of its own current and best locations with those of one or more members of the swarm, with some random perturbations. This process simulates the social interaction between humans looking for the same objective or a flock of birds looking for food, for instance.

The entire swarm is modeled as a multidimensional space $\Re^n$, in which each particle $p_i = (x_i, v_i) \in \Re^n$ has two main features: (i) position ($\vec{x}_i$) and (ii) velocity ($\vec{v}_i$). The local (best current position $\hat{x}_i$) and global solution $\vec{g}$ are also known for each particle. After defining the swarm size $m$, i.e., the number of particles, each one is initialized with random values for both velocity and position. Each individual is then evaluated with respect to some fitness function and its local maximum is updated. At the end, the global maximum is updated with the particle that achieved the best position in the swarm. This process is repeated until some convergence criterion is reached. The updated velocity and position equations of the particle $p_i$ in the simplest form that governs the Particle Swarm Optimization at time step $t$ are, respectively, given by

$$v_i^j(t+1) = wv_i^j(t) + c_1 r_1(\hat{x}_i(t) - x_i^j(t)) + c_2 r_2(\vec{g} - x_i^j(t)) \quad (42)$$

and

$$x_i^j(t+1) = x_i^j(t) + v_i^j(t+1), \quad (43)$$

where $w$ is the inertia weight that controls the interaction between particles, and $r_1, r_2 \in [0, 1]$ are random variables that give the stochastic idea to Particle Swarm Optimization. The variables $c_1$ and $c_2$ are used to guide the particles onto good directions.

## IV. Materials and Methods

In this section, we described the methodology used in the experiments, as well as the datasets employed to compared the effectiveness and efficiency of the evolutionary-based techniques for finding proper values of $k$.

### A. Datasets

In this work, we have employed two public datasets, as described below:

- KddCup[1]: this dataset is composed of millions samples and is divided in 23 classes, being 22 of them related to network attacks, and the remaining one stands for normal access. The number of features is 41. In this paper, we used a reduced dataset, which is composed of 9% of the original dataset size (44,091 samples).

- NSL-Kdd[2]: is a dataset specially designed to remove redundancy of the well known KddCup'99 dataset. More details about it can be found in [32]. Notice we have used 35% of the original dataset (44,090 samples).

### B. Experimental Evaluation

As explained in Section II-A, OPF clustering employs the $k \in [1, k_{\max}]$ value to compute the density of each graph node (Equation 1), and further such value is used to find the $k$-nearest neighbors to begin the competition process between prototypes, since their discrete influence region is bounded by their neighborhood (inner loop in Lines 6-10 according to Algorithm 1).

In this work, we compared several evolutionary optimization techniques to find out $k$, since the current implementation of OPF [33] employs an exhaustive search for that, which may be impracticable for large datasets. Since the clustering quality can be measured by the minimum graph cut (Equation 2), we use it as the fitness function, i.e., the evolutionary-based optimization techniques will find proper $k$ values that minimize the graph cut. As we are working with discrete values for $k$, we employed for all techniques an array of size equal to $k_{\max}$ to save the occurrences of $k$. The idea is to avoid that agents with the same solution be evaluated twice, which may increase the computational load. Thus, techniques with good exploration ability, i.e., capability to attract the other agents, can take advantage of this strategy to become faster.

In order to avoid meta-optimization, we have chosen the meta-heuristic parameters empirically, based in our previous experience. The maximum number of the iterations were set as 20. We have also investigated the influence of techniques

[1] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

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

regarding the number of agents. As such, the number of agents has been set within the range $[5, 25]$ in steps of five. Table IV-B presents the parameters used for all nature-inspired optimization techniques employed in our experiments.

| Technique | Parameters |
|---|---|
| Bat Algorithm | $\alpha = \gamma = 0.9$ |
| Cuckoo Search | $\alpha = 0.01, p_a = 0.35$ |
| Charged System Search | CMCR= 0.9, PAR= 0.4 |
| Firefly Algorithm | $\phi = 0.8, \iota = \beta_0 = 1$ |
| Gravitational Search Algorithm | $G(t_0) = 100, \zeta = 20$ |
| Harmony Search | HMCR= 0.9, PAR= 0.3 |
| Particle Swarm Optimization | $c_1 = c_2 = 2, w \in [0.4, 0.9]$ |

*Table 1*: Meta-heuristic algorithm parameters setting. Recall that $w$ value for PSO was linearly decreased from 0.9 to 0.4. The same was applied with the number of agents for GSA, which decreased from the maximum number of agents to 1.

## V. Experiments

In this section, we present the experiments conducted to assess the robustness of the evolutionary optimization techniques over KddCup and NSL-Kdd datasets. Firstly, we have performed an exhaustive search with the traditional approach available in the LibOPF [33] aiming to find out the optimal $k$ value within the range $[1, k_{max}]$, being $k_{max} = 100$. We consider such value reasonable taking into account the size of both datasets. In regard to KddCup dataset, the minimum graph cut found was 3.900435 in 390.65 seconds, and regarding to NSL-Kdd the minimum graph cut found was 0.000013 in 346.13 seconds. We used these results as our baseline in order to compare with the optimization techniques. Thus, the idea is to verify which techniques are able to achieve close graph cut values to the ones obtained by exhaustive search. The remaining discussion presents the results obtained by the optimization techniques.

As the number of agents increases, a more exploration force the metaheuristic algorithms have. However, a high number of agents may increase the computational load. In our case, we are interested in techniques able to provide a good trade-off between graph cut values and a low computational load. Figures 1 and 2 display the average graph cut values over 10 runnings for KddCup and NSL-Kdd datasets. As we can note, PSO and BA were the best approaches, achieving the best graph cut values for all number of employed agents. As such, we can imply that both techniques have good exploitation capabilities, even with a small number of agents.

CSS has been the second best approach, being a good explorer only when the number of agents has been more than 10. Although CS presented the same behavior as CSS for KddCup dataset, it was less consistent regarding NSL-Kdd. Thus, we can infer that CS was not suitable to scape from local traps in some cases. GSA, HS and FFA were the worst performers, and they did not present good consistent results as the number of agents vaires. Although GSA achieved good results with 10, 15 and 25 agents for KddCup dataset; and 15 and 25 for NSL-Kdd, it did not achieve good results with 20 agents for both datasets. HS and FFA presented also irregular performances, being them the worst techniques with a slow exploitation rate.

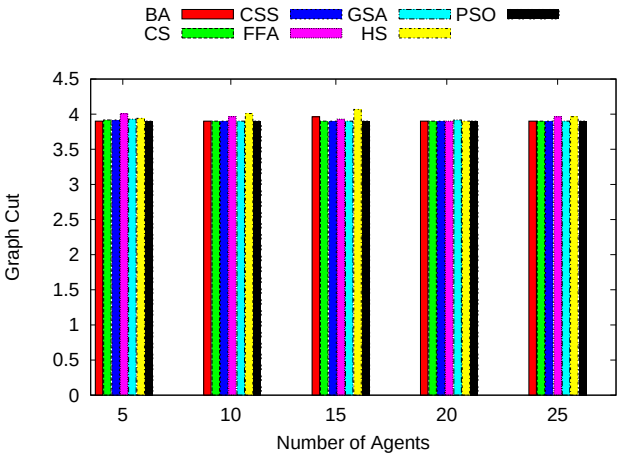


**Figure. 1**: Graph cut values varying the number of agents for the KddCup dataset.

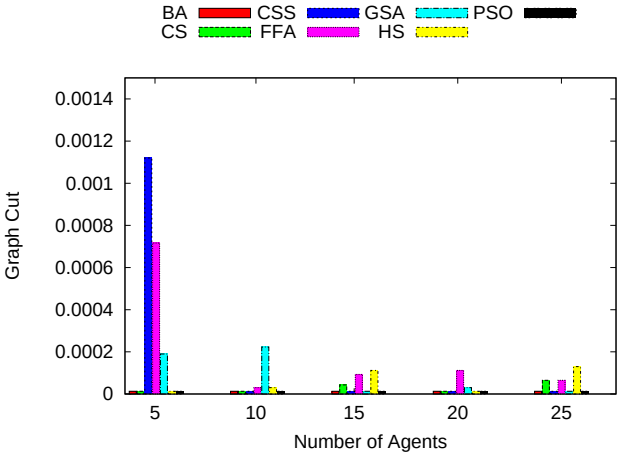


**Figure. 2**: Graph cut values varying the number of agents for the NSL-Kdd dataset.

From Figures 3 and 4 we can observe the mean execution times for all techniques over both datasets. As expected, the best performers, PSO and BA, were the slowest techniques as the number of agents increases. This happens due to their good capability for exploring the search space. Unlike, techniques with low explore capability, such as FFA and HS, tend to move their agents slowly, which may make an agent stay longer in a position and be not evaluated while its respective $k$ value does not change. Thus, these techniques may present a lower computational load regarding to PSO and BA, for instance.

## VI. Conclusions

In this paper we deal with the problem of finding suitable values for the OPF clustering $k$-neighborhood that lead to near-optimal graph cut vales. We have compared several nature-inspired optimization techniques for such task, and we have validated it in the context of intrusion detection in computer networks.

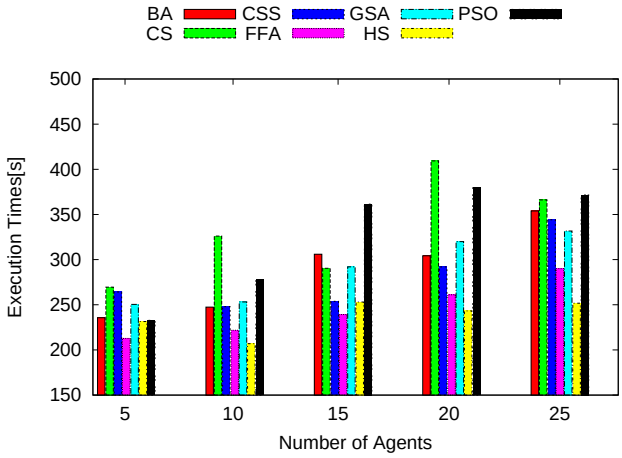


**Figure. 3**: Execution times in seconds varying the number of agents for KddCup dataset.

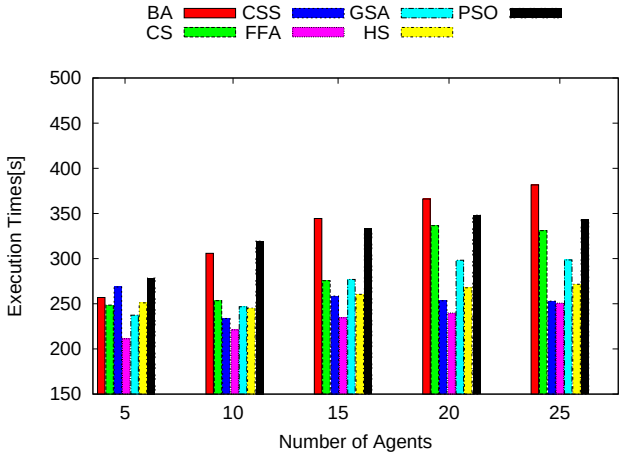


**Figure. 4**: Execution times in seconds varying the number of agents for NSL-Kdd dataset.

The experimental results in two public datasets have showen that all techniques achieved similar graph cut values for KddCup dataset. In addition, the number of agents did not improve the graph cut results. In regard to NSL-Kdd dataset, the minimum graph cut values were obtained with 5 agents by BA, CS, HS and PSO approaches. Therefore, we can note that the number of agents did not influence the effectiveness, making the evolutionary techniques slower when the number of agents increases. However, if we consider only 5 agents, we can find suitable values for $k$ faster than an exhaustive search. In regard to future works, we intend to evaluate faster evolutionary-based optimization techniques for finding suitable $k$ values.

## Acknowledgment

The authors are grateful to FAPESP (São Paulo Research Foundation) grants #2009/16206-1, #2012/14494-2, #2011/14094-1 and #2011/14058-5, and also CNPq (National Counsil of Technological and Scientific Development) grant #303182/2011-3.

## References

[1] Shi Zhong, Taghi M. Khoshgoftaar, and Naeem Seliya, "Clustering-based network intrusion detection," *International Journal of Reliability, Quality and Safety Engineering*, vol. 14, no. 2, pp. 169–187, 2007.

[2] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo, "Intrusion detection with unlabeled data using clustering," in *Proceedings of ACM CSS Workshop on Data Mining Applied to Security*, 2001, pp. 5–8.

[3] Nong Ye and Xiangyang Li, "A scalable clustering technique for intrusion signature recognition," in *Proceedings of 2nd IEEE SMC Information Assurance Workshop*, 2001, pp. 1–4.

[4] Yu Guan and Ali A. Ghorbani, "Y-means: A clustering method for intrusion detection," in *Proceedings of Canadian Conference on Electrical and Computer Engineering*, 2003, pp. 1083–1086.

[5] Eleazar Eskin, "Anomaly detection over noisy data using learned probability distributions," in *Proceedings of the International Conference on Machine Learning*. 2000, pp. 255–262, Morgan Kaufmann.

[6] S.X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, 2010.

[7] R. Chaki and N. Chaki, "IDSX: A cluster based collaborative intrusion detection algorithm for mobile ad-hoc network," in *Proceedings of the 6th International Conference on Computer Information Systems and Industrial Management Applications*, 2007, pp. 179–184.

[8] J. Sen, "An intrusion detection architecture for clustered wireless ad hoc networks," in *Proceedings of the Second International Conference on Computational Intelligence, Communication Systems and Networks*, july 2010, pp. 202–207.

[9] C. Cai, H. Pan, and J. Cheng, "Fusion of bvm and elm for anomaly detection in computer networks," in *International Conference on Computer Science Service System (CSSS)*, 2012, pp. 1957–1960.

[10] A.G. Tartakovsky, A.S. Polunchenko, and G. Sokolov, "Efficient computer network anomaly detection by changepoint detection methods," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 1, pp. 4–11, 2013.

[11] L. M. Rocha, F. A. M. Cappabianco, and A. X. Falcão, "Data clustering as an optimum-path forest problem with applications in image analysis," *International Journal of Imaging Systems and Technology*, vol. 19, no. 2, pp. 50–68, 2009.

[12] J. P. Papa, A. X. Falcão, and C. T. N. Suzuki, "Supervised pattern classification based on optimum-path forest," *International Journal of Imaging Systems and Technology*, vol. 19, no. 2, pp. 120–131, 2009.

[13] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 790–799, Aug 1995.

[14] K. Costa, C. Pereira, R. Nakamura, and J. Papa, "Intrusion detection in computer networks using optimum-path forest clustering," in *Proceedings of the IEEE 37th Conference on Local Computer Networks*, 2012, pp. 128–131.

[15] K. Costa, C. Pereira, R. Nakamura, L. Pereira, and J. Papa, "Boosting optimum-path forest clustering through harmony search and its applications for intrusion detection in computer networks," in *Fourth International Conference on Computational Aspects of Social Networks*, 2012, pp. 181–185.

[16] Z. W. Geem, *Music-Inspired Harmony Search Algorithm: Theory and Applications*, Springer Publishing Company, Incorporated, 1st edition, 2009.

[17] J. Kennedy and R.C. Eberhart, *Swarm Intelligence*, M. Kaufman, 2001.

[18] Tang Rui, Simon Fong, Xin-She Yang, and Suash Deb, "Nature-inspired clustering algorithms for web intelligence data," in *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, 2012, pp. 147–153.

[19] X.-S. Yang., "Bat algorithm for multi-objective optimisation," *International Journal of Bio-Inspired Computation*, vol. 3, no. 5, pp. 267–274, 2011.

[20] Xin-She Yang and Suash Deb, "Engineering optimisation by cuckoo search," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 1, pp. 330–343, 2010.

[21] Rui Tang, S. Fong, Xin-She Yang, and Suash Deb, "Integrating nature-inspired optimization algorithms to k-means clustering," in *Seventh International Conference on Digital Information Management (ICDIM)*, 2012, pp. 116–123.

[22] Xin-She Yang, "Firefly algorithm, stochastic test functions and design optimisation," *International Journal Bio-Inspired Computing*, vol. 2, no. 2, pp. 78–84, 2010.

[23] A. Kaveh and S. Talatahari, "A novel heuristic optimization method: charged system search," *Acta Mechanica*, vol. 213, no. 3, pp. 267–289, 2010.

[24] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, "GSA: A gravitational search algorithm," *Information Sciences*, vol. 179, no. 13, pp. 2232–2248, 2009.

[25] D. Comaniciu, "An algorithm for data-driven bandwidth selection," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 25, no. 2, pp. 281–288, 2003.

[26] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, Aug 2000.

[27] J. D. Altringham, T. McOwat, and L. Hammond, *Bats: biology and behaviour*, Oxford University Press, USA, 1998.

[28] Xin She Yang, "A new metaheuristic bat-inspired algorithm," in *Proceedings of the Nature Inspired Cooperative Strategies for Optimization*. 2010, vol. 284 of *Studies in Computational Intelligence*, pp. 65–74, Springer.

[29] Xin-She Yang and Suash Deb, "Cuckoo search via lvy flights," in *Proceedings of the NaBIC 2009 - World Congress on Nature & Biologically Inspired Computing*, 2009, pp. 210–214.

[30] David Halliday, Robert Resnick, and Jearl Walker, *Extended , Fundamentals of Physics, 6th Edition*, Wiley, 2000.

[31] R. Mansouri, F. Nasseri, and M. Khorrami, "Effective time variation of g in a model universe with variable space dimension," *Physics Letters*, vol. 259, pp. 194–200, 1999.

[32] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*. 2009, pp. 53–58, IEEE Press.

[33] J. P. Papa, C. T. N. S., and A. X. Falcão, *LibOPF: A library for the design of optimum-path forest classifiers*, 2009, Software version 2.0 available at `http://www.ic.unicamp.br/~afalcao/LibOPF`.

# Author Biographies

**First Author** Kelton Pontara Augusto da Costa received his B.Sc. in Systems Analysis from Sagrado Coração University, SP, Brazil. In 2004, he received his M.Sc. in Computer Science from the Centro Universitário Eurípides, SP, Brazil. In 2009, he received his Ph.D. in Electrical Engineering from the São Paulo State University, SP, Brazil. During 2010-2011, he worked as a post-doctorate researcher at the Institute of Computing of the University of Campinas, SP, Brazil. He currently works as a post-doctorate researcher in the Department of Computer Science of the UNESP - Univ. Estadual Paulista, SP, Brazil. He is a professor in the Department of Computing, College of Technology of the São Paulo State since 2008 and his research interests include machine learning, security in computer networks and detecting anomalies in computer networks.

**Second Author** Clayton R. Pereira received his B.Sc. in Information Systems from Facol - Faculdade Orígenes Lessa, SP, Brasil in 2008. In 2010, he received a specialization in Information Technology Management from Faculdade Anhanguera, SP, Brasil. In 2012, he received his M.Sc in Computer Science from UNESP - Univ. Estadual Paulista, SP, Brasil. Currently, he is pursuing the Ph.D in Computer Science from the Federal University of São Carlos, SP, Brasil.

**Third Author** Luís A. M. Pereira received his B.Sc. in Information Systems from UNESP - Univ Estadual Paulista, SP,

Brazil (2011). Currently, he is pursuing the M.Sc in computer science at the same institute. His research interests include machine learning, pattern recognition and image processing.

**Fourth Author** Rodrigo Y. M. Nakamura received his B.Sc. in Computer Science from UNESP - Univ Estadual Paulista, SP, Brazil (2011). Currently, he is pursuing the M.Sc in computer science at the same institute. His research interests include machine learning and pattern recognition.

**Fifth Author** João P. Papa received his B.Sc. in Information Systems from UNESP - Univ Estadual Paulista, SP, Brazil. In 2005, he received his M.Sc. in Computer Science from the Federal University of São Carlos, SP, Brazil. In 2008, he received his Ph.D. in Computer Science from the University of Campinas, SP, Brazil. During 2008-2009, he had worked as post-doctorate researcher at the same institute. He has been Professor at the Computer Science Department, UNESP - Univ Estadual Paulista, since 2009, and his research interests include machine learning, pattern recognition and image processing.