Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT

PERGAMON

Pattern Recognition 37 (2004) 927–942

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY
www.elsevier.com/locate/patcog

A genetic clustering method for intrusion detection

Yongguo Liu[a,b,*], Kefei Chen[a], Xiaofeng Liao[b], Wei Zhang[b,c]

[a] *Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200030, China*
[b] *Department of Computer Science and Engineering, Chongqing University, Chongqing 400044, China*
[c] *Department of Computer and Modern Education Technology, Chongqing Education College, Chongqing 400067, China*

Received 27 February 2003; received in revised form 22 September 2003; accepted 22 September 2003

Abstract

Traditional intrusion detection methods lack extensibility in face of changing network configurations as well as adaptability in face of unknown attack types. Meanwhile, current machine-learning algorithms need labeled data for training first, so they are computational expensive and sometimes misled by artificial data. In this paper, a new detection algorithm, the Intrusion Detection Based on Genetic Clustering (IDBGC) algorithm, is proposed. It can automatically establish clusters and detect intruders by labeling normal and abnormal groups. Computer simulations show that this algorithm is effective for intrusion detection.

Keywords: Intrusion detection; Clustering; Genetic algorithms; Simulated annealing

1. Introduction

As the increase of the significance of computer networks in modern society, its security becomes one of the hottest issues to be solved. Therefore, it is extremely imperative to find an effective way to protect this valuable network infrastructure. In general, an intrusion is defined as "any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource" [1]. How to repel intrusions is one of key tasks in the field of the computer network security. Network intrusion prevention techniques, such as user authentication, avoiding programming errors and information protection, have been extensively applied to protect computer systems [2]. However, intrusion prevention alone is not sufficient for this purpose because as computer networks become ever more complex, intrusion methods become ever more intelligent [3]. The policies that balance convenience versus strict control of a system and information access also make it impossible for computer systems to be completely secure. Therefore, intrusion detection is naturally used as another means to protect vulnerable network systems.

The basic assumptions of intrusion detection are users and program activities are observable, and more importantly, normal activities and intrusion activities have distinct behaviors. The goal of intrusion detection is to monitor network activities automatically, detect malicious attacks and to establish a proper architecture of the computer network security with cooperation of other intrusion prevention approaches such as the firewall. Once an intrusion is detected, system administrators are informed in no time and can take correct action. In order to detect intruders, a number of detection methods have been proposed [4–6]. These methods extract features from network data, and detect intrusions using a preset hand-coded algorithm set by human experts. Such methods cannot adapt to new types of intrusions effectively, and the preset algorithm has to be manually revised for every type of intrusions invented. Other approaches make use of machine-learning algorithms trained by labeled network data for this task [2,7–9]. When unknown types of attacks need to be detected, these techniques possess the advantage of being able to automatically retrain detection

* Corresponding author. Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200030, China. Tel.: +86-21-62932135.
E-mail address: liu-yg@cs.sjtu.edu.cn (Y. Liu).

0031-3203/$30.00
doi:10.1016/j.patcog.2003.09.011

models on different input data. But labeled data are not readily available in most cases. Generally, a very large volume of network data is encountered, thus it is expensive to classify them manually. Of course, labeled data can be obtained under an artificial circumstance. However, confined to the set of simulated intrusions and unknown to types of attacks unshown in the training data, the intrusion detection system will not effectively detect new attacks eventually. In order to solve this problem, a new detection algorithm, the Intrusion Detection Based on Genetic Clustering (IDBGC) algorithm, is proposed in this article. This algorithm consists of two stages, namely, nearest neighbor clustering and genetic optimization. Based on aforementioned assumptions, i.e. intrusion activities will appear as outliers for normal activities and can be grouped into different clusters from the normal cluster. The IDBGC algorithm can automatically establish clusters and detect intruders by labeling normal groups and abnormal groups, respectively. By means of experimental results, its effectiveness is proven.

The remaining part of this paper is organized as follows. In Section 2, the related work in clustering analysis is discussed, especially for those clustering methods based on genetic algorithms. In Section 3, the IDBGC algorithm and its stages are proposed. Experimental results are described in Section 4. Finally, some conclusions are drawn in Section 5.

2. The related work in clustering analysis

Clustering analysis is a formal study of algorithms and methods for grouping, or classifying objects without category labels. A cluster is a set of entities that are alike, and entities from different clusters are not alike. Many clustering techniques have been proposed [10,11]. They can be classified into two categories: hierarchical and partitional. Among them, the K-means algorithm is an important one as it is a typical iterative hill-climbing algorithm. This algorithm is proven to fail to converge to a local minimum under certain conditions [12]. In Ref. [13], a branch-and-bound algorithm is proposed to find the globally optimum clustering, but it requires a long computation time. Simulated annealing algorithms are applied to clustering and the results show that a globally optimum solution can be obtained under certain conditions [14,15]. In Refs. [16–25], evolution strategies and genetic algorithms are explored for solving the clustering problem. They are described in detail as follows.

In Ref. [16], the applicability of evolution strategies to optimize the centroid type of clustering objective functions, within group sum of squared error measure, is discussed. The solution x is represented by a vector of real numbers $x_1x_2\cdots x_n$, and $n = C \times d$ where d denotes the number of dimensions and C denotes the number of clusters. As the number of dimensions and the number of clusters increase, the convergence rate decreases abruptly because of the exponential increase of the search space.

In Ref. [17], a partition is encoded as a chromosome of length n which corresponds to the number of objects. The ith element of the chromosome represents the cluster number to which the corresponding object belongs. The performance comparison between the proposed algorithm and the K-means algorithm is provided. In many real life situations, the proposed approach suffers from the problem of the limited applicability in face of huge data sets.

In Ref. [18], an improved genetic algorithm for clustering objects in images based on their visual features is proposed. The Boolean Matching Code and the Single Gene Crossover are defined for the genetic clustering. The Boolean Matching Code, an alternative to Linear Code, denotes each variable with a string of n bits having a bit for each object in the following format:

$$S_{\mathrm{BMC}} = [b_{11},\ldots,b_{1n}|b_{21},\ldots,b_{2n}|\cdots|b_{m1},\ldots,b_{mn}],$$

$$b_{ij} = \begin{cases} 1 \rightarrow x_j \in c_i, \\ 0 \rightarrow x_j \notin c_i, \end{cases}$$

where x_j denotes an object of a set of n objects $S = \{x_1, x_2, \ldots, x_n\}$ and m denotes the size of clusters $\{c_1, c_2, \ldots, c_m\}$. For instance, there are three clusters in data sets: $c_1 = \{x_1, x_6\}$, $c_2 = \{x_3, x_4, x_7\}$, $c_3 = \{x_2, x_5, x_8\}$. Therefore, the chromosome can be coded by BMC as $S_{\mathrm{BMC}} = [10000100, 00110010, 01001001]$. In this way, for the experimental data used in this article, intrusion categories are more than 20 and the size of each instance set is more than 1000. As a result, the chromosome is too long to be put into the computer memory.

In Ref. [19], a genetic algorithm approach to clustering analysis, COWCLUS, is proposed. This algorithm is an independent-cluster-seeking algorithm that applies the Calinski and Harabasz Variance Ratio Criterion as the fitness function to explain the concepts of within-cluster cohesion and between-cluster isolation. COWCLUS is a genetic/hill-climbing hybrid: first the nondeterministic genetic algorithm is used to explore many good partitions, then deterministic hill-climbing methods are used to improve these partitions and to produce the final best partition. Any partition of n distinct objects into m clusters is called a candidate that is represented by a sequence (called a chromosome) of n integers, each between 1 and m, where the jth component (called a gene) of the sequence indicates to which cluster the jth points is assigned. In this way, the search space is not acceptable for intrusion detection discussed in this paper as above mentioned.

In Ref. [20], a hybrid genetic algorithm (GKA) using the distance-based-mutation operator and the K-means operator is proposed to solve the clustering problem. In GKA, the chromosome, whose length is equal to the number of objects, an allele corresponds to a pattern and its value represents the cluster number to which the corresponding pattern belongs. In this way, the search space of the GKA will exponentially increase with the size of objects and the size of patterns. For intrusion detection, this again will not be feasible because

the size of instances is very large and the size of patterns is much larger than 2.

In Ref. [21], three models, SImultaneously Clustering Method (SICM), STepwise Clustering Method (STCM) and Cluster Seed Points Method (CSPM), are presented according to different coding/decoding techniques to solve the clustering problem. SICM assumes that there must be at least two objects to form a cluster, by which the maximum number of clusters can be deduced $K = \lfloor N/2 \rfloor$. Take 17 objects for example, there are at most 8 partition sets ($8 = \lfloor 17/2 \rfloor$). K is then coded in binary code and so 8 partition sets require three genes for representation. The chromosome is represented as $S = [x_1 x_2 x_3 \cdots x_n]$, where x_i is denoted by the cluster coded in binary code that it belongs to. For the above example, the chromosome is composed of 51 genes. STCM successively solves the optimal binary clustering of a cluster until the objective value cannot be further improved under the depth first principal. The length of the original chromosome is equal to the size of objects. In each binary division, genetic algorithms are employed to solve the 0–1 mathematical programming by maximizing the value of the fitness function to attain the optimal binary clustering. CSPM first employs genetic algorithms to select the most suitable cluster seeds from all objects, then assigns the rest of the objects to each cluster according to their similarity to the cluster seed or to the degree of improvement of the objective function. The number of cluster seeds represents the number of clusters. The length of the chromosome is also equal to the size of objects. The experimental results indicate that the computation capability of the aforementioned three models is reduced substantially when the size of objects $n > 200$. The reason is that the coding technique leads to a very large search space for these three algorithms to converge efficiently at the optimal point.

In Ref. [22], an algorithm, GA-clustering, is proposed which uses the clustering center to minimize the fitness function. The chromosome is composed of m D-dimensional instances that act as the clustering center in real number code. Therefore the length of the chromosome is $m * D$. Ref. [23] solves the clustering problem in the same way. It is assumed that the centroid of data points belonging to a cluster represents the center of that cluster. Each chromosome is denoted by a sequence of $m * D$ floating point numbers where the first D positions denote the D dimensions of the first cluster center, the next D positions represent those of the second cluster center, and so on. As a result, the length of the chromosome is the same as that in Ref. [22]. By simulated experiments, the algorithms described in Ref. [22] are superior to the K-means algorithm while the algorithm proposed in Ref. [23] is superior to both the K-means algorithm and that proposed in Ref. [17]. However, the length of the chromosome is still too long for intrusion detection.

Most clustering techniques mentioned in Refs. [16–23] require the designer to specify the number of clusters. However, the designer has no idea, in general, about this beforehand and so the algorithms are not able to effectively achieve the optimal performance by setting the number of clusters without a priori information. This problem becomes especially significant for intrusion detection in face of unknown or new attack types. Most normal instances come into one cluster, and other intrusions come into $K - 1$ clusters. But K is unknown for the designer beforehand. So, aforementioned approaches that require the designer to specify the number of clusters subjectively are unfeasible for this issue.

To determine the number of clusters in the data set, the GCUK-clustering algorithm is proposed [24]. By setting a range of the number of clusters $[K_{\min}, K_{\max}]$, the chromosome is initialized by the centers of K_i clusters generated in the range $[K_{\min}, K_{\max}]$ at random, and the left allele is represented as the do not care symbol '#'. The representation mode of the chromosome is similar to that of Refs. [22,23]. Then GCUK-clustering can evolve the proper number of clusters and the computed centers are close to the actual ones according to simulated experiments. But this algorithm lacks an effective method to find the value of $K_{\max}$ which determines the result of the clustering algorithm. This issue is not addressed in the article and so $K_{\max}$ is still specified by the designer subjectively in Ref. [24].

In Ref. [25], a genetic clustering algorithm is applied to estimate the proper number of clusters and group data points with compact spherical clusters. Here, the clustering centers are represented as genes in the chromosome to reduce the computation complexity. This algorithm is only suitable for clusters with compact spherical shape, not for nonspherical clusters.

In Ref. [26], under-partitioned state, optimal-partitioned state and over-partitioned state are introduced to determine the proper number of clusters in the partitioning process. As the clustering state enters different partitioned states, the change of the number of clusters will result in the fluctuation of the value of the validity index. The goal of this algorithm is to obtain the optimal cluster number with the smallest value of the validity index. But this algorithm is not suitable for data sets whose shape is not compact and convex.

3. The IDBGC algorithm

Clustering in D-dimensional Euclidean space R^D is a process of partitioning a given set of n instances into K groups based on some similarity or dissimilarity metrics. Let the set of n points $\{x_1, x_2, \ldots, x_n\}$ be represented by the set C and K clusters be represented by $c_1, c_2, \ldots, c_K$. Then

$$c_i \neq \phi, \quad i = 1, \ldots, K,$$

$$c_i \cap c_j = \phi, \; i, j = 1, \ldots, K \quad \text{and} \quad i \neq j \quad \text{and}$$

$$\bigcup_{i=1}^{K} c_i = C.$$

Before designing the IDBGC algorithm, we assume that the size of normal activities vastly outnumbers that of intrusion activities in the network data set in this paper. In general, normal activities are the mainstream in the normal network environment and intrusion activities are rare. In this way, the IDBGC algorithm will be mainly applied to detect intrusions in the normal network environment. According to the basic idea that intrusions are essentially different from normal activities and will appear as outliers, intrusions and normal activities are distinct and not overlapped. In this article, the nearest neighbor method is used to group similar instances to set up the original set of clusters at the first stage.

The IDBGC algorithm consists of two stages. They are stated as follows:

1. The stage of nearest neighbor clustering: To establish the set of original clusters using the nearest neighbor method by grouping very similar instances into a cluster and filter noisy data objects based on some similarity or dissimilarity metrics.
2. The stage of genetic optimization: To combine original clusters by genetic algorithms and obtain the near optimal result, then label the cluster including most activities as the normal according to above assumptions.

3.1. The limitation of the clustering center

In Section 2, many clustering approaches based on the concept of the clustering center are described. In this paper, two examples are given to describe the limitation of this concept to intrusion detection. Take Ref. [26] for instance, three clustering concepts are presented, under-partitioned state, optimal-partitioned state and over-partitioned state. The clustering partition function including the mean intra-cluster distance (MICD) and the inter-cluster minimum distance (ICMD) is defined according to the partition process shown as Fig. 1. As the partition state enters the under-partitioned state, at least one cluster maintains a large MICD. According to the partitioning process, the following conclusions are drawn in Ref. [26]: as the partition state moves to the optimal-partitioned and over-partitioned ones, the large MICD abruptly decrease; on the other hand, ICMD becomes large when the data set is under-partitioned or optimally partitioned; as the state enters the over-partitioned one, ICMD becomes very small because at least one of the compact clusters is subdivided. But for the cluster whose shape may not be convex and spherical, MICD and ICMD will change in different way. Here, two examples are given to discuss this problem.

Figs. 2 and 3 show these two examples. They are composed of three clusters C_0, C_1 and C_2 whose data objects are of uniform and compact distribution. In Fig. 2, two ring clusters with equal radius distribute symmetrically to the center of coordinate axis. The other cluster is composed of two semi-rings distributing symmetrically to the center of coordinate axis. The radii of two semi-rings are equal and twice as those of two ringed clusters. In Fig. 2(a), c_0, c_1 and c_2 denote the center of clusters C_0, C_1 and C_2, respectively. According to Ref. [26], the distance set in Fig. 2(a) is defined as $d'_{\mathrm{ICMD}} = \{d_{c_0c_1}, d_{c_0c_2}, d_{c_1c_2}\}$, i.e. $d'_{\mathrm{ICMD}} = \{\sqrt{D_2^2 + R_1^2}, \sqrt{D_1^2 + R_2^2}, \sqrt{D_2^2 + R_1^2} + \sqrt{D_1^2 + R_2^2}\}$. Since $R_1 = R_2$ and $D_1 = D_2$, then $d'_{\mathrm{ICMD}} = \{\sqrt{R_1^2 + D_1^2}, \sqrt{R_1^2 + D_1^2}, 2\sqrt{R_1^2 + D_1^2}\}$. As the clustering state enters the over-partitioned one, the cluster C_0 is divided by the X-axis into two semi-rings and two new clustering centers c_3^* and c_4^* are then set up. At this time, the distance set $d''_{\mathrm{ICMD}} = \{d_{c_1c_2}, d_{c_1c_3^*}, d_{c_1c_4^*}, d_{c_2c_3^*}, d_{c_2c_4^*}, d_{c_3^*c_4^*}\}$, i.e. $d''_{\mathrm{ICMD}} = \{2\sqrt{R_1^2 + D_1^2}, \sqrt{(D_1 + R_1)^2 + R_1^2}, \sqrt{(3R_1)^2 + R_1^2}, \sqrt{(3R_1)^2 + R_1^2}, \sqrt{(D_1 + R_1)^2 + R_1^2}, 2\sqrt{R_1^2 + (2R_1)^2}\}$. If $D_1 \leqslant 2R_1$, ICMD will increase rather than decrease. In Fig. 3, C_0 and C_2 are two concentric rings. C_1 and C_2 have equal radii which are one half as that of C_0. In Fig. 3(a), the distance set is defined as $d'_{\mathrm{MICD}} = \{2R_1, R_1, R_1\}$ under the optimal-partitioned state. As the state enters the under-partitioned one, the distance set $d''_{\mathrm{MICD}} = \{1.5R_1, R_1\}$, which shows the conclusion that at least one cluster maintains large MICD is not true. Above phenomena show that the clustering partition functions in Ref. [26] need to confine itself to the case where clusters are of compact and spherical shape. Therefore, the validity index in Ref. [26] may not be suitable for data sets whose clusters are not compact and convex. Because of the uncertainty for intrusions, a new validity index suitable is needed.

3.2. The stage of nearest neighbor clustering

The task of this stage is to remove noisy data and to build the set of original clusters by the nearest neighbor approach.

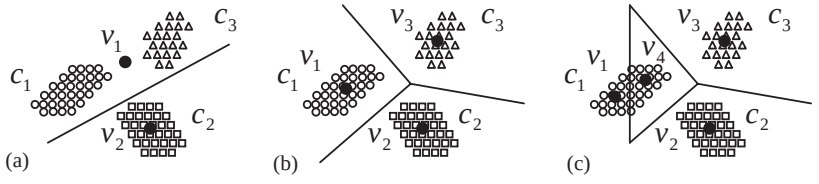

Fig. 1. Partition of the data set in Ref. [26]. (a) Under-partitioned state, (b) optimal-partitioned state and (c) over-partitioned state.

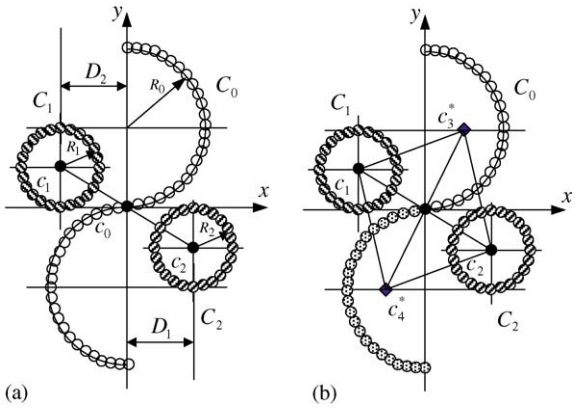

Fig. 2. Partition of example 1. (a) Optimal-partitioned state and (b) over-partitioned state.

In this paper, the Euclidean metric is used as the mainly similarity metric.

3.2.1. Data normalization processing

We encounter a problem when processing instances whose different features are on different scales. This will cause bias toward some features over other ones. For instance, consider two 3-feature instances whose each feature comes from different distributions, $A = (34, 2899, 0.5)$, $B = (45, 4000, 0.8)$. Under the Euclidean metric, the squared distance between instances is $D = (34 - 45)^2 + (2899 - 4000)^2 + (0.5 - 0.8)^2$, which is dominated by the second field obviously. To solve this problem, these raw data sets need to be normalized as follows:

$$I_j^e = \frac{1}{n}\sum_{i=1}^{n} I_{ij},$$

$$I_j^\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(I_{ij} - I_j^e)^2},$$

then

$$I_j' = \frac{I_j - I_j^e}{I_j^\sigma},$$

where I_j^e denotes the average feature instance of I_j, I_j^σ denotes the standard deviation feature instance of I_j, and I_j' denotes the normalized instance. In the following section, I_j will denote the normalized instance. Here, all features are equally weighted in order to enhance the algorithm's generality. Meanwhile, for some features with discrete values, we add a constant value to the squared distance between the instances for every discrete feature where they have two distinct values so as to treat each different value as being orthologous in the feature space.

3.2.2. The algorithm of nearest neighbor clustering

The algorithm is described in the following:

Step 1: For the instance I_j, $j = 1, \ldots, n$, where n denotes the size of instances, compute the similarity between I_j and its nearest neighbor I_t, $t = 1, \ldots n$, and $j \neq t$, i.e. $d_j = \min\|I_j - I_t\|$, where $\|I_j - I_t\| = (\sum_{q=1}^{D}(I_{jq} - I_{tq})^2)^{1/2}$.

Step 2: Compute the average value of the nearest neighbor similarity, $d = w * 1/n \sum_{j=1}^{n} d_j$, where w denotes the weighted factor applied to adjust the similarity within one cluster and the dissimilarity among different clusters.

Step 3: Set up the adjacency matrix of all instances $M_{n\times n}$ as follows:

$$M(j,t) = \begin{cases} 0, & j = t, \\ 1, & d_j \leqslant d, \\ 0, & d_j > d. \end{cases} \quad \text{where } j, t = 1, \ldots, n.$$

Step 4: Remove noisy data objects. After Step 3, those objects that are not adjacent to any others will be labeled as intrusions. The reason is that these objects are far away from normal objects and will enhance the computation complexity at the second stage and result in a local minimum under certain conditions. For example, one abnormal object extremely far away from others will prevent the detection algorithm from finding other intrusions. After being filtered, these objects will be ignored in the stage of genetic optimization.

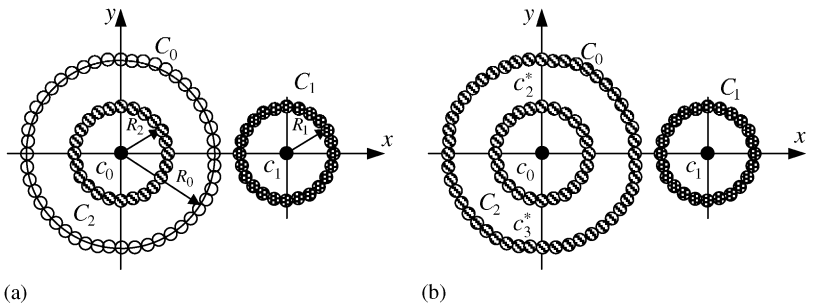

Fig. 3. Partition of example 2. (a) Optimal-partitioned state and (b) under-partitioned state.

Step 5: Build the set of original clusters. Group adjacent instances into clusters, respectively, and label them as $c_1, c_2, \ldots, c_K$. In this way, there are K clusters, where K denotes the number of clusters after removing instances labeled as intrusions.

Step 6: Label the cluster with the most instances as c_m, $m \leqslant K$, which denotes that this cluster is a set of normal activities because above assumptions ensure that the cluster representing normal instances must be qualitatively different from that of the abnormal and include the most instances.

Step 7: For cluster c_j, $j=1,\ldots,K$, compute the similarity between cluster c_j and its nearest neighbors c_t, $t=1,\ldots,K$, and $j \neq t$, i.e. $d_k^* = \min \|c_j - c_t\|$.

In general, K is far smaller than n, i.e. $K \ll n$. The main purpose of the first stage is to reduce the size of data sets to a moderate one suitable for genetic algorithms at the second stage and to reduce the computing time as much as possible. For the issue stated in Section 3.1, the case mentioned in Figs. 1(a) and (c) will not happen after above steps because only two similar instances will be group into a cluster and they will also not be divided. Meanwhile, for cases given in Figs. 2 and 3, the original clusters obtained are the optimal partition after all the above-mentioned steps are performed.

3.3. The stage of genetic optimization

After above steps, the set of initial clusters $C = \{c_1, c_2, \ldots, c_K\}$ can be obtained. This stage is a genetic optimization process at which initial clusters are used to set up the initial chromosome. Genetic Algorithms (GAs) are randomized search and optimization techniques guided by the principles of evolution and natural genetics. They have a large amount of implicit parallelism and provide the near optimal solution of the objective or fitness function in complex, large, and multi-modal landscapes [27]. In GAs, the parameters of the search space are encoded in the form of strings (or chromosomes). The fitness function that represents the goodness degree of the solution encoded in the chromosome is associated with each string. Biologically inspired operators like selection, crossover, and mutation are used over a number of generations for generating potentially better solutions. GAs consist of an initialization step and an evolution step with three phases in each generation, which will be described in the following.

3.3.1. Chromosome representation

Here, the chromosome model is built according to initial clusters, i.e. let the chromosome length equal to the size of initial clusters, which will help to reduce the search space. The chromosome is initialized with the binary code. For example, suppose $C = \{c_1, c_2, \ldots, c_K\}$ after the upper stage, a chromosome $\underbrace{10101\cdots01100}_{K}$ is set up at random to represent the distribution of clusters selected in the clustering order. If c_i is selected, the ith position of the chromosome will be 1; otherwise, it will be 0. In IDBGC, the cluster c_m ($m \leqslant K$) with more instances than others is always selected, i.e. the mth position of the chromosome is 1. Meanwhile, to prevent the fitness function from being misled by some abnormal clusters that are much farther away from the normal than other abnormal clusters, the upper limitation factor H is introduced. That is, $d_H^* = H * 1/n \sum_{j=1}^{n} d_j$. If $d_k^* > d_H^*$, where $k = 1,\ldots,K$, and $m \neq k$, then the kth position of the chromosome is 0, i.e. the cluster c_k ($k \leqslant K$) is always tabooed and labeled as intrusions.

3.3.2. Selection operation

In this article, those individuals with high fitness values are selected by the survival of the fittest concept of natural genetic rules. Based on above assumptions, the optimal cluster c' with the most instances is taken as the normal, and the others are all intrusions. Therefore, the normal instance set will be the main object to be concerned about, i.e. c' will be established in the end after some clusters are merged into the cluster c_m under certain conditions. The computation of the individual fitness includes two steps: creation of the new cluster and its fitness computation.

- *Creation of the new cluster*: Given a chromosome S at random, the new cluster is created as follows:

$$S_r = s_1^r s_2^r \cdots s_i^r \cdots s_{K-1}^r s_K^r, \quad r = 1,\ldots,P,\ i = 1,2,\ldots,K,$$

$$\begin{cases} s_i^r = 1, & c_i \text{ is selected,} \\ s_i^r = 0, & \text{otherwise,} \end{cases}$$

P denotes the population size, s_i^r denotes the ith gene in the rth chromosome and S_r denotes the rth individual, then

$$c_m^r = c_m^r \cup c_i \quad \text{if } \|c_i - c_m\| \leqslant \|c_i - c_k\| \quad \text{and}$$
$$s_m^r, s_k^r = 1, s_i^r = 0, \quad \text{where } i \neq k, i, k = 1,\ldots,K,$$

where s_m^r, s_k^r denotes the mth and kth gene in the rth chromosome, respectively, $\|c_i - c_k\| = \min_{\substack{I_i \in c_i \\ I_k \in c_k}} \|I_i - I_k\| = \min_{\substack{I_i \in c_i \\ I_k \in c_k}} (\sum_{q=1}^{D} (I_{iq} - I_{kq})^2)^{1/2}$. After this step, new c_m^r is obtained, and the number of clusters merged is μ.

- *Fitness computation*: Here, the concept of under-partitioned state, optimal-partitioned state and over-partitioned state as proposed in Ref. [26] is introduced to build the validity criterion of clustering. As the clustering state enters the optimal-partitioned state, the value of the individual fitness tends to be maximum. On the other hand, as the partition state moves to the under-partitioned or over-partitioned one, the value of the individual fitness will decrease. Two distance concepts are proposed here, INTRA-Cluster Distance (INTRACD) and INTEL-Cluster Distance (INTELCD). INTRACD is used to measure the degree of nearness among the clusters in c_m^r, and INTELCD is used to measure the degree of separation among the clusters in c_m^r and other outside clusters. The fitness function is

defined as follows:

$$\begin{cases} F(S) = F_c & \text{if } s_m^r = 1,\ s_i^r = 0, \\ & \quad i \neq m,\ i = 1, \ldots, K, \\ F(S) = d_{\text{INTELCD}}/d_{\text{INTRACD}} & \text{if } d_{\text{INTELCD}} \geqslant d_{\text{INTRACD}}, \\ F(S) = F_c & \text{if } d_{\text{INTELCD}} < d_{\text{INTRACD}}, \end{cases}$$

where

$$\begin{cases} d_{\text{INTRACD}} = \dfrac{1}{\mu - 1} \displaystyle\sum_{i=1}^{\mu-1} \min_{\substack{i,j \in c_m^r \\ i \neq j}} (d(c_i - c_j)), \\ d_{\text{INTELCD}} = \dfrac{1}{\mu} \displaystyle\sum_{i=1}^{\mu} \min_{\substack{i \in c_m^r \\ j \notin c_m^r}} (d(c_i - c_j)) \end{cases}$$

if $s_i^r = 0,\ i \neq m$, then $s_i^{r'} = \text{rand}[0, 1]$. $s_i^{r'}$ denotes the update of s_i^r and F_c is a pre-defined fitness which is less than others. Some explanations may be helpful in understanding the definition of $F(S)$. After the initialization step and some evolution steps, the individual will change in chromosome structure. When only the gene s_m^r is selected, this individual will be washed out to the most because it is difficult to obtain the best solution at the first stage. The same action will be taken for those solutions where d_{INTELCD} is less than d_{INTRACD}. The reason is that the nearest neighbor method ought to ensure that the distance among the clusters in c_m^r is less than that between the clusters in c_m^r and those outside clusters. According to the equations stated above, the individual fitness includes two parts: d_{INTRACD} denotes the average distance between two clusters belonging to c_m^r; d_{INTELCD} denotes the average distance between two clusters belonging to c_m^r and $C^r - c_m^r$, respectively. In this way, the rth individual in population, which includes the most normal instances and the least intrusions in c_m^r, is the best, i.e. $\max(F(S))$ is the best.

- *Annealing selection criterion*: In each step, called a generation, GAs produce a set of candidate solutions, called child solutions, using two types of genetic operators: mutation and crossover. GAs select good solutions as survivors to the next generation according to the fitness value. The mutation operator takes a single parent and modifies it randomly in a localized manner, so that it makes a small jump in the solution space. On the other hand, the crossover operator takes two solutions as parents and creates their child solutions by combining the partial solutions of the parents. Crossover tends to create child solution that differs from both parent solutions. It results in a large jump in the solution space. Therefore, GAs have two features: one is that GAs maintain a population of solutions and uses them to search the solution space; the other is that GAs make use of the crossover operator to achieve a large jump in the solution space. These features allow GAs to globally search the large region of the solution space. But GAs have no explicit ways to produce a sequence of small moves in the solution space. Mutation creates a single small move one at a time instead of a sequence of small moves. As a result, GAs cannot search local region in the solution space exhaustively with the proportional selection strategy such as roulette wheel selection criterion.

In this paper, the simulated annealing algorithm is proposed as the selection criterion to improve the performance of GAs [28]. It can escape from local minimum and produce a sequence of small moves in the solution space. However, it cannot cover a large region of the solution space within a limited computation time. The selection criterion is described as follows:

$$\begin{cases} p_s = 1 & \text{if } F_n \geqslant F_0, \\ p_s = \mathrm{e}^{-F_0 - F_n/T} & \text{if } F_n < F_0, \end{cases}$$

where p_s denotes the survival probability of the child individual, F_0 denotes the individual fitness of the parent generation, F_n denotes the individual fitness of the child generation and T denotes the annealing temperature.

3.3.3. Crossover operation

Crossover is a probabilistic process that exchanges information between two parents for generating two children. In this paper, the single-point crossover with a fixed crossover probability p_c is used. For chromosomes of length L, a random integer, called the crossover point, is generated in the range $[1, L]$. The portions of the chromosomes lying to the right of the crossover point are exchanged to produce two offspring.

3.3.4. Mutation operation

Mutation is also a probabilistic process that takes a single parent and modifies some genes with its alleles in a localized manner. As a result, bits of the chromosomes in the population will be chosen with probability p_m in the mutation phase and each chosen bit will be changed from 0 to 1 or vice versa if the binary code is chosen.

3.3.5. Implementation of the second stage

The second stage of the IDBGC algorithm is implemented as follows:

Step 1: Initialization. Set the evolutionary generation $M = 300$, crossover rate $p_c = 0.75$, mutation rate $p_m = 0.01$, annealing control factor $\alpha = 0.70$, initial annealing temperature $T_0 = 10$, and binary code mode. Generate an initial population $\xi_0 = (S_1 S_2 \cdots S_r \cdots S_P),\ r = 1, \ldots, P$, where the population size $P = 200$.

Step 2: Evaluation and Selection. Evaluate the individual fitness value $F(S_r),\ r = 1, 2, \ldots, P$. And if the current generation is not equal to one, then generate a new individual S_r' according to p_s. Otherwise proceed to Step 3.

Step 3: Crossover. After the selection, cross two individuals with probability p_c in single point crossover mode.

Step 4: Mutation. After crossover, mutate individual with probability p_m.

Step 5: Termination check. If $q \leqslant M$, then $q = q + 1$ and $T_q = \alpha * T_{q-1}$, return to Step 2. Otherwise proceed to Step 6.

Step 6: Output c'. Set $c' = 1,\ i = 1, \ldots, P$ and compute c_m^i, if $|c_m^i| \geqslant |c'|$, then $c' = c_m^i$; when $i = P$, output c'.

Table 1
The structure of the experimental data sets

Data set name	Instance number	Number normal of instances	Number of abnormal instances	Number of attack type	Number of attack category
Data set 1	1100	1000	100	22	4
Data set 2	1100	1000	100	20	4
Data set 3	1100	1000	100	22	4
Data set 4	1100	1000	100	22	4
Data set 5	1100	1000	100	22	4

Table 2
The distribution of attack types in the experimental data sets

Attack type	Data set 1	Data set 2	Data set 3	Data set 4	Data set 5	Attack category
Back	8	4	17	10	8	DOS
Buffer_overflow	4	4	4	3	2	U2R
Ftp_write	4	4	4	4	1	R2L
Imap	4	4	4	3	4	R2L
Ipsweep	4	8	8	6	10	PROBE
Land	4	4	4	2	4	DOS
Loadmodule	4	4	4	4	3	U2R
Multihop	9	3	4	7	2	R2L
Neptune	4	4	4	3	4	DOS
Nmap	8	4	5	10	7	PROBE
Perl	5	3	3	2	4	U2R
Phf	4	4	0	3	4	R2L
Pod	4	8	4	4	6	DOS
Portsweep	4	4	4	4	3	PROBE
Rootkit	4	4	4	2	4	U2R
Satan	4	4	4	2	4	PROBE
Smurf	4	4	7	10	12	DOS
Spy	2	2	0	1	2	R2L
Teardrop	4	4	4	4	3	DOS
Warezclient	4	4	4	3	4	R2L
Warezmaster	4	4	4	3	4	R2L
Guess_passwd	4	8	4	10	5	R2L

4. Experimental results

The KDD Cup 1999 Data is used as the experimental data set with many kinds of network intrusions simulated [29]. This data set contains over 40 features except the label feature, which is derived from the 1998 DARPA Intrusion Detection Evaluation Program held by MIT Lincoln Labs. The objective of the evaluation program is to survey and evaluate researches in intrusion detection. A standard set of data to be audited is provided, which includes a wide variety of intrusions simulated in a military network environment.

Lincoln Labs set up an environment to acquire nine weeks of raw TCP dump data for a local-area network (LAN) simulating a typical US Air Force LAN. They operated the LAN as if it were a true Air Force environment, but peppered it with multiple attacks. The raw training data are about 4 Gbytes of compressed binary TCP dump data from 7 weeks of network traffic. This is processed into about five million connection records. A connection is a sequence of TCP packets starting and ending at some well-defined times, between which data flow to and from a source IP address to a target IP address under some well defined protocol. Each connection is labeled as either normal, or as an attack,

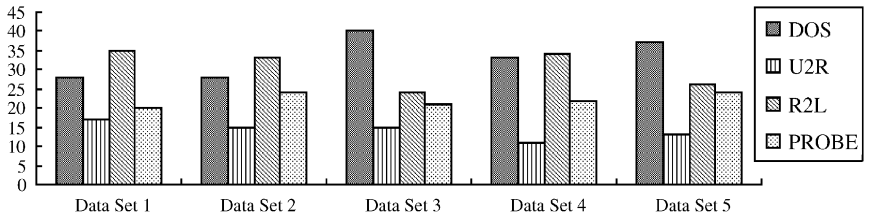

Fig. 4. Attack category distribution of the experimental data sets.

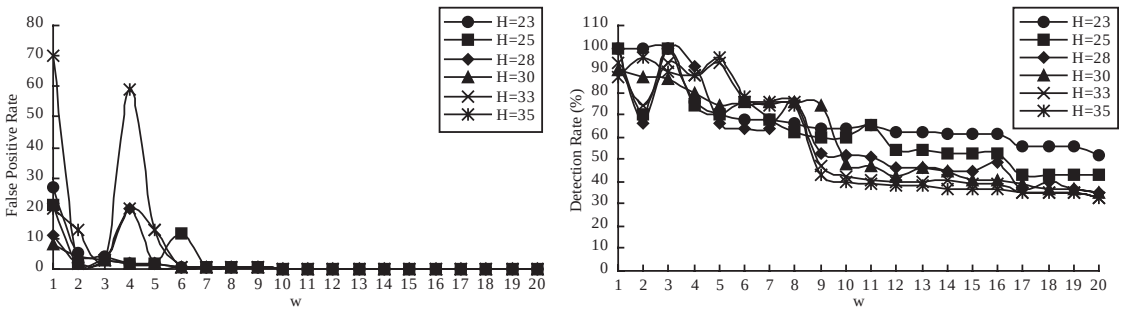

Fig. 5. False positive rate and detection rate of data set 1.

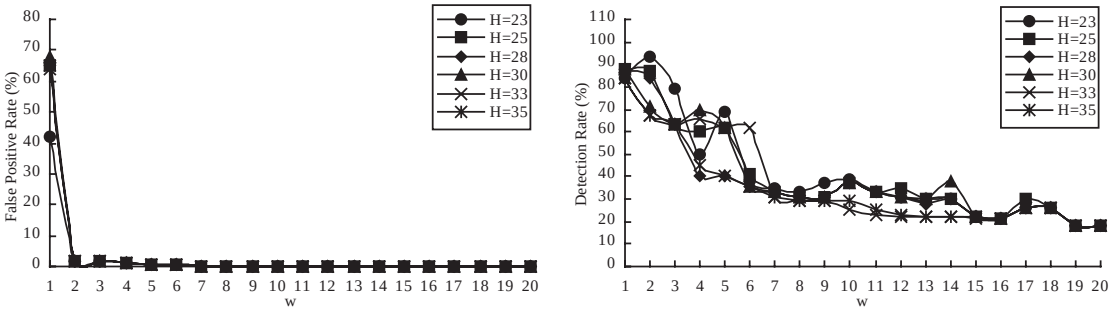

Fig. 6. False positive rate and detection rate of data set 2.

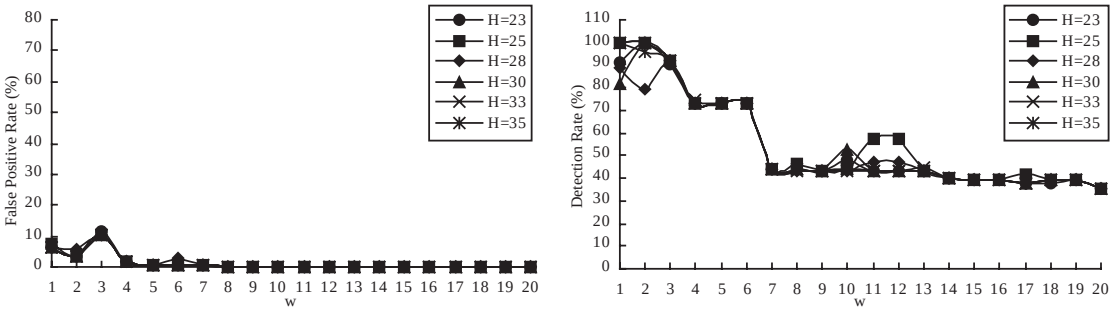

Fig. 7. False positive rate and detection rate of data set 3.

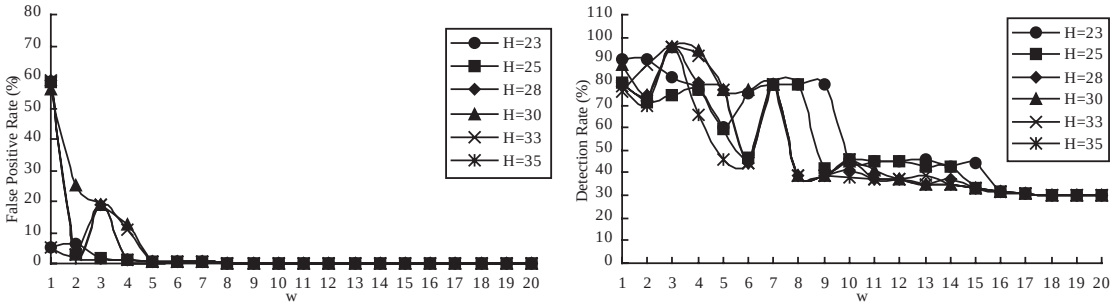

Fig. 8. False positive rate and detection rate of data set 4.

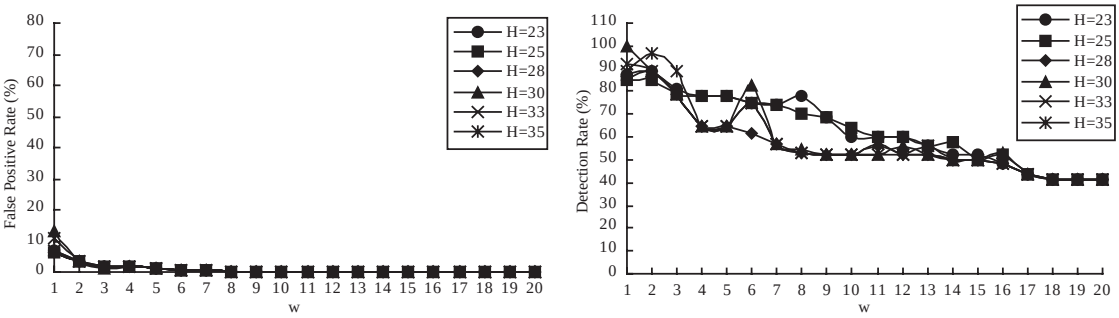

Fig. 9. False positive rate and detection rate of data set 5.

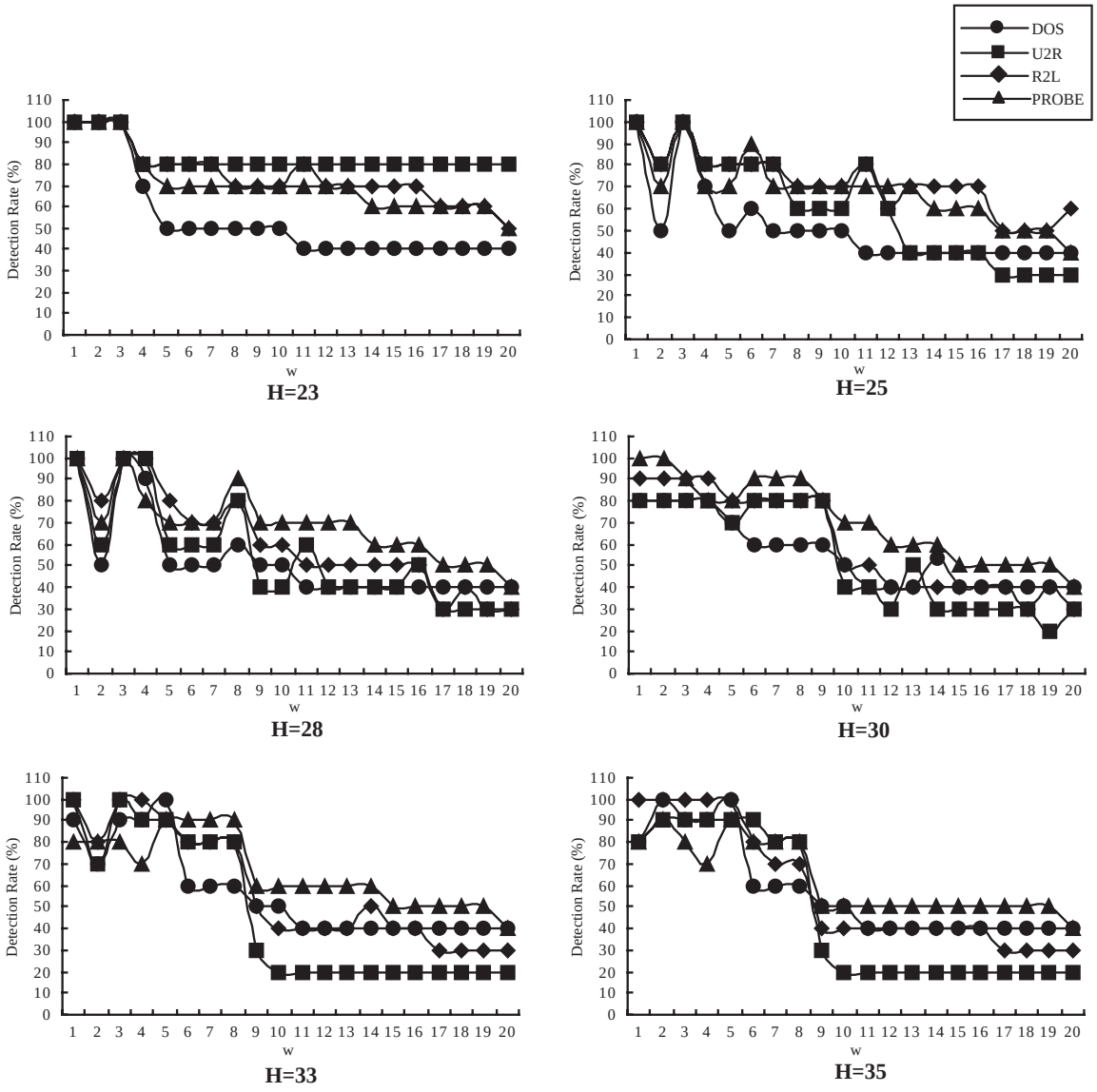

Fig. 10. Detection rate of data set 1 for attack category.

with exactly one specific attack type. Each connection record consists of about 100 bytes. The attack types fall in one of the following four categories:

DOS—Denial of service (e.g. land attack).

R2L—Unauthorized access from a remote machine (e.g. password guessing).

U2R—Unauthorized access to super user or root functions (e.g. buffer overflow attack).

PROBE—Surveillance and other probing for vulnerabilities (e.g. port scanning).

For convenience, the entire KDD data set is partitioned into ten 10% subsets, each containing approximately 500,000 instances. But many of these data sets contain instances of only a single type, which does not satisfy our assumptions. For example, the fourth, fifth, sixth, and seventh 10% portions of the full data set contain only the SMURF attack, and data instances in the eighth portion are almost entirely the NEPTUNE intrusion. Meanwhile, some subsets only include several attack types that cannot reflect the reality very well. Therefore, five new data sets are set up to perform the IDBGC algorithm. Each data set contains 1000 normal instances and 100 intrusion instances, all of which are selected at random. The label feature is neglected when the IDBGC algorithm is implemented. The structure

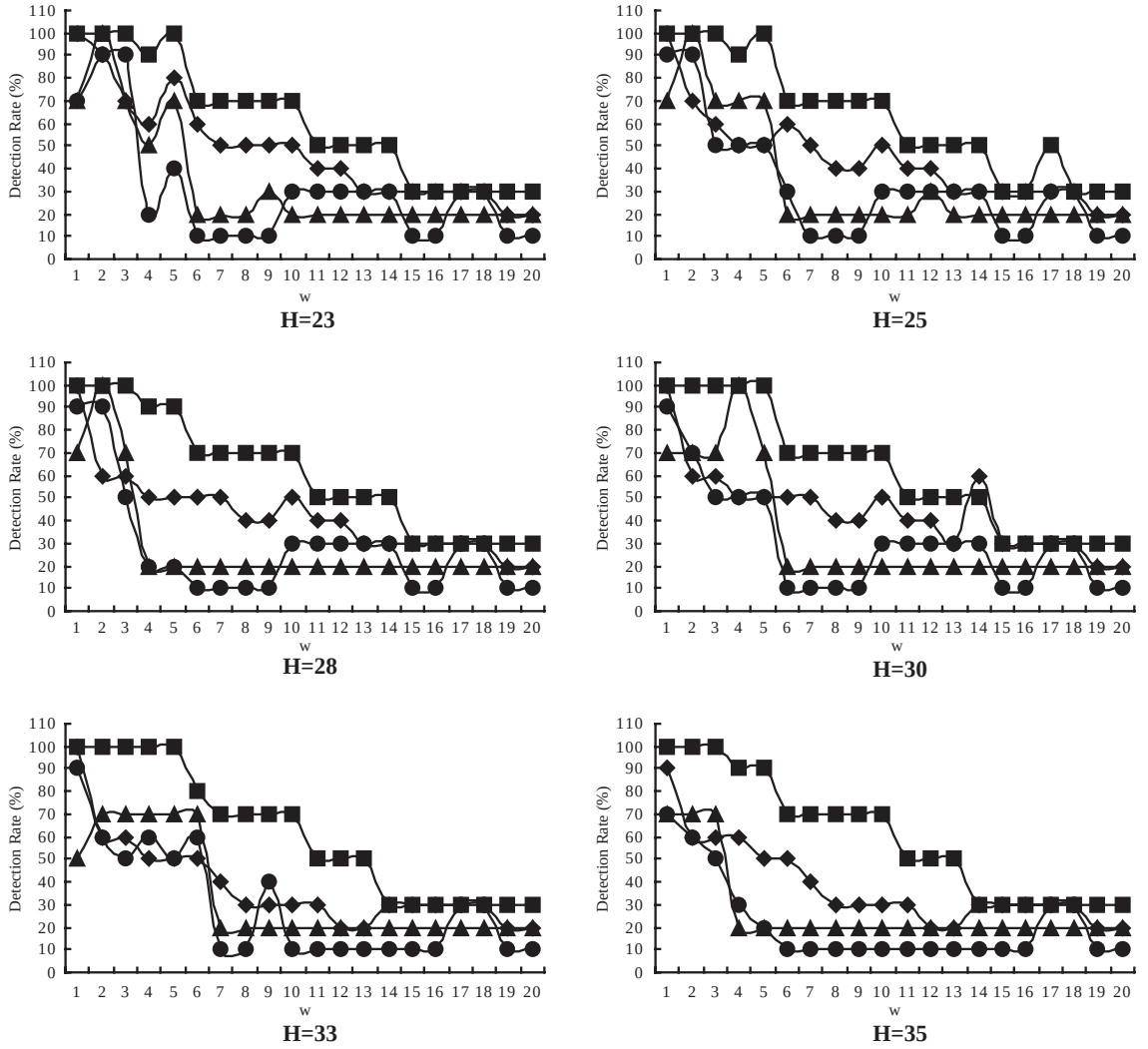

Fig. 11. Detection rate of data set 2 for attack category.

of the five experimental data sets including the respective size of normal instance, intrusion instance, specific attack and intrusion category in each data set is shown in Table 1.

In Table 2, the specific attack type distribution of the experimental data sets is listed.

At the same time, attack category distribution in each data set is shown in Fig. 4.

Experiments are conducted on an Intel Pentium III processor running at 550 MHz with 128 MB real memory. Figs. 5–9 show the clustering results of these five data sets.

Two major indicators of performance can be found in Figs. 5–9: the false positive rate and the detection rate. The false positive rate is defined as the number of normal instances that are (incorrectly) classified as intrusions divided by the total number of normal instances. The detection rate is defined as the number of intrusion instances detected by the system divided by the total number of intrusion instances present in the test set. These are good indicators of performance, since they measure what percentage of intrusions the system is able to detect and how many incorrect classifications are made in the process.

In Figs. 5–9, as the growth of the weighted factor w, both the false positive rate and the detection rate decrease, and tend to the same value respectively under different upper limitation factor H for different data sets. When $w \leqslant 6$, the false positive rate fluctuates substantially in most cases, which causes the detection rate varies within a great range for different H. The reason is explained in detail in the following. When w is small, some noisy data filtered at the first

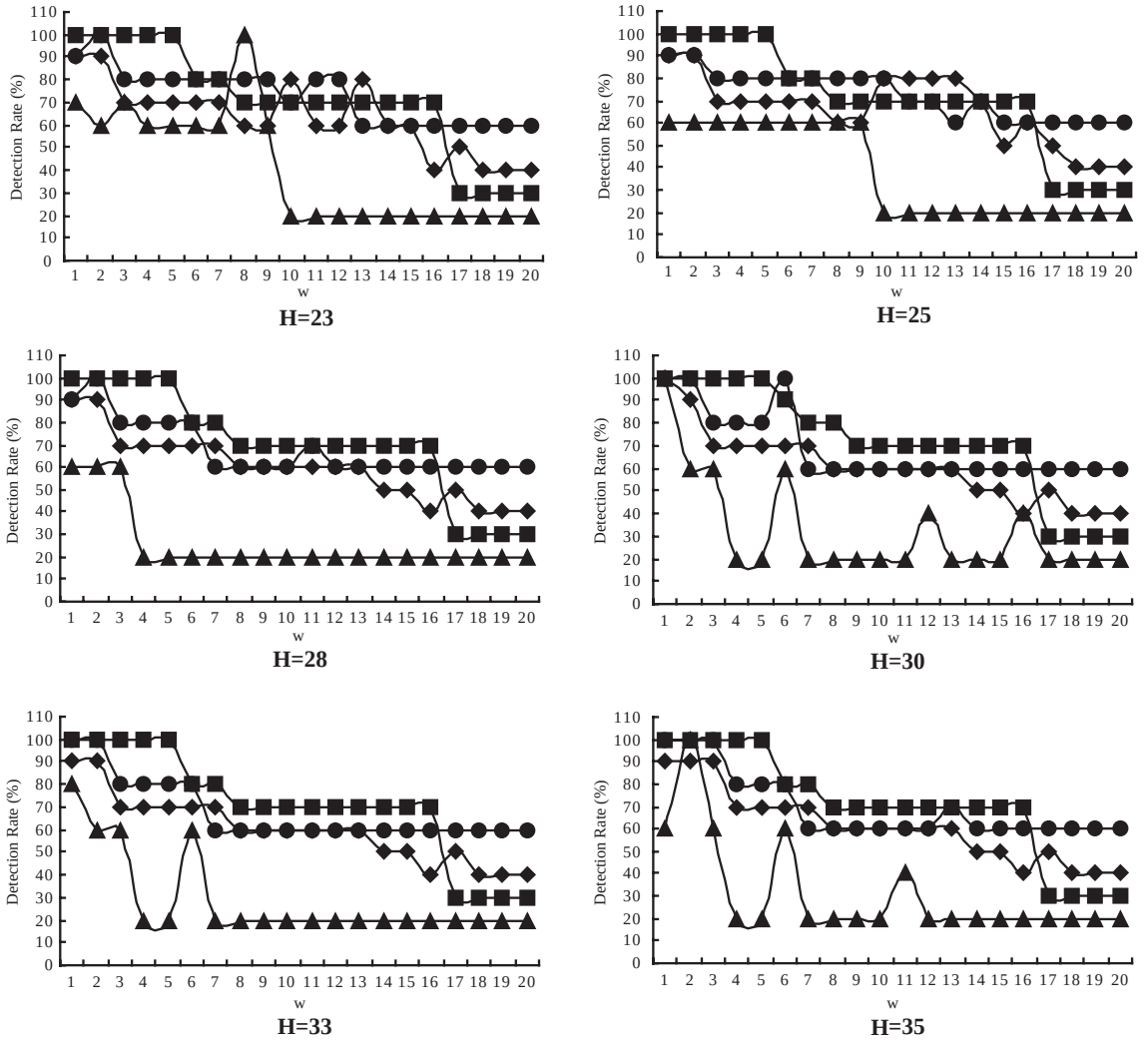

Fig. 12. Detection rate of data set 3 for attack category.

stage may not be intrusions. Being normal instances that are relatively far away from the main normal cluster at small w, these instances generally play the bridge role among different normal clusters. Once these instances are labeled as intrusions, the association degree among some normal clusters will decrease distinctly and normal instances will not be combined together according to the proposed validity index. At this time, although the detection rate maintains a high level, the false positive rate is also high in most cases. In these experiments, the value of upper limitation factor H remains at a relatively high level to avoid causing the same variation of the weighted factor w, i.e. a small H will cause the growth of the false positive rate. Let us consider the case that $H = 25$ and $w = 7$, then the final detection result are given as follows:

Figs. 10–14 show the detection results for different attack categories in these five data sets (Table 3).

Table 3
Experimental results for five data sets

Data set name	H	w	Detection rate (%)	False positive rate (%)
Data set 1	25	7	68	0.8
Data set 2	25	7	33	0.2
Data set 3	25	7	74	0.4
Data set 4	25	7	44	0.3
Data set 5	25	7	79	0.3
Average	25	7	59.6	0.4

In Figs. 10–14, influences of the weighted factor w and the upper limitation factor H on the detection rate for different attack categories in five data sets are shown. The basic tendency is the same as that of Figs. 5–9, i.e. when $w \leqslant 6$, the detection rate for different attack categories maintains at a high level and fluctuates in a wide range. As

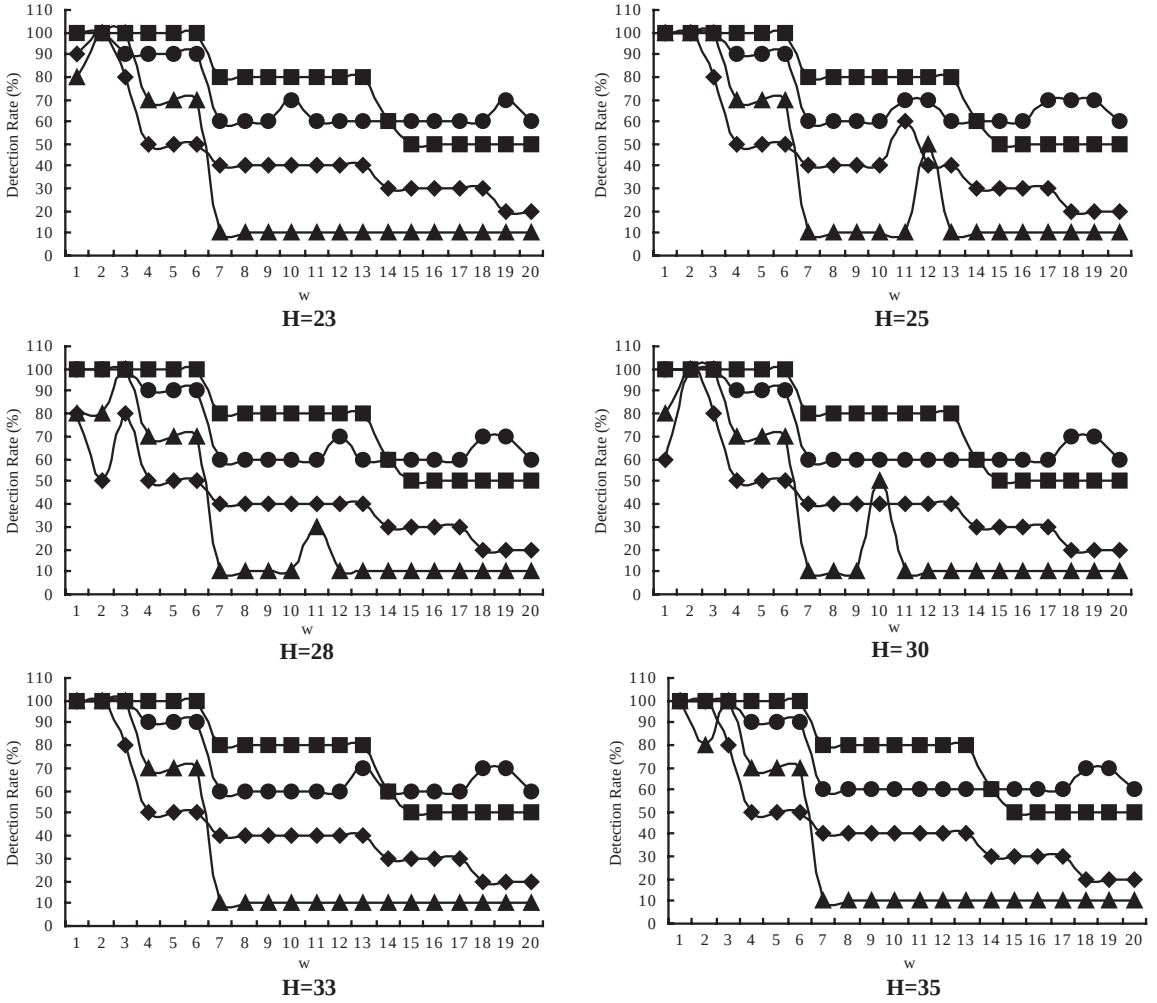

Fig. 13. Detection rate of data set 4 for attack category.

the growth of w, the detection rate for all attack categories reduces to certain extent. In these experiments, the value of upper limitation factor H keeps at a relatively high level and helps to efficiently filter those abnormal clusters far away from normal instances in some sense. Moreover, as the growth of H, the sensibility to intrusions degrades and high sensitivity zone becomes narrow. Meanwhile, the IDBGC algorithm can keep good detection characteristics for most attack categories under certain parameter condition, but also show some great discrepancies for different attack categories. Consider the same case that $H = 25,\ w = 7$. The detection result for these attack categories are described in the following.

In computer simulations, the average detection rate is close to 60%, and the average false positive rate is only 0.4%, all of which show that the IDBGC algorithm is effective in detecting unknown intrusions. From Table 4, more than 50% unknown attacks belonging to DOS, R2L and U2R are detected. In particular, about four-fifths of the U2R attack can be detected. Meanwhile, for attacks fall into the category of PROBE, the detection ratio is less than 50%, i.e. relatively low.

Table 4
Experimental results for different attack categories

Data set name	Detection DOS rate (%)	Detection U2R rate (%)	Detection R2L rate (%)	Detection PROBE rate (%)
Data set 1	50	80	80	70
Data set 2	10	70	50	20
Data set 3	80	80	70	60
Data set 4	60	80	40	10
Data set 5	80	80	90	60
Average	56	78	66	44

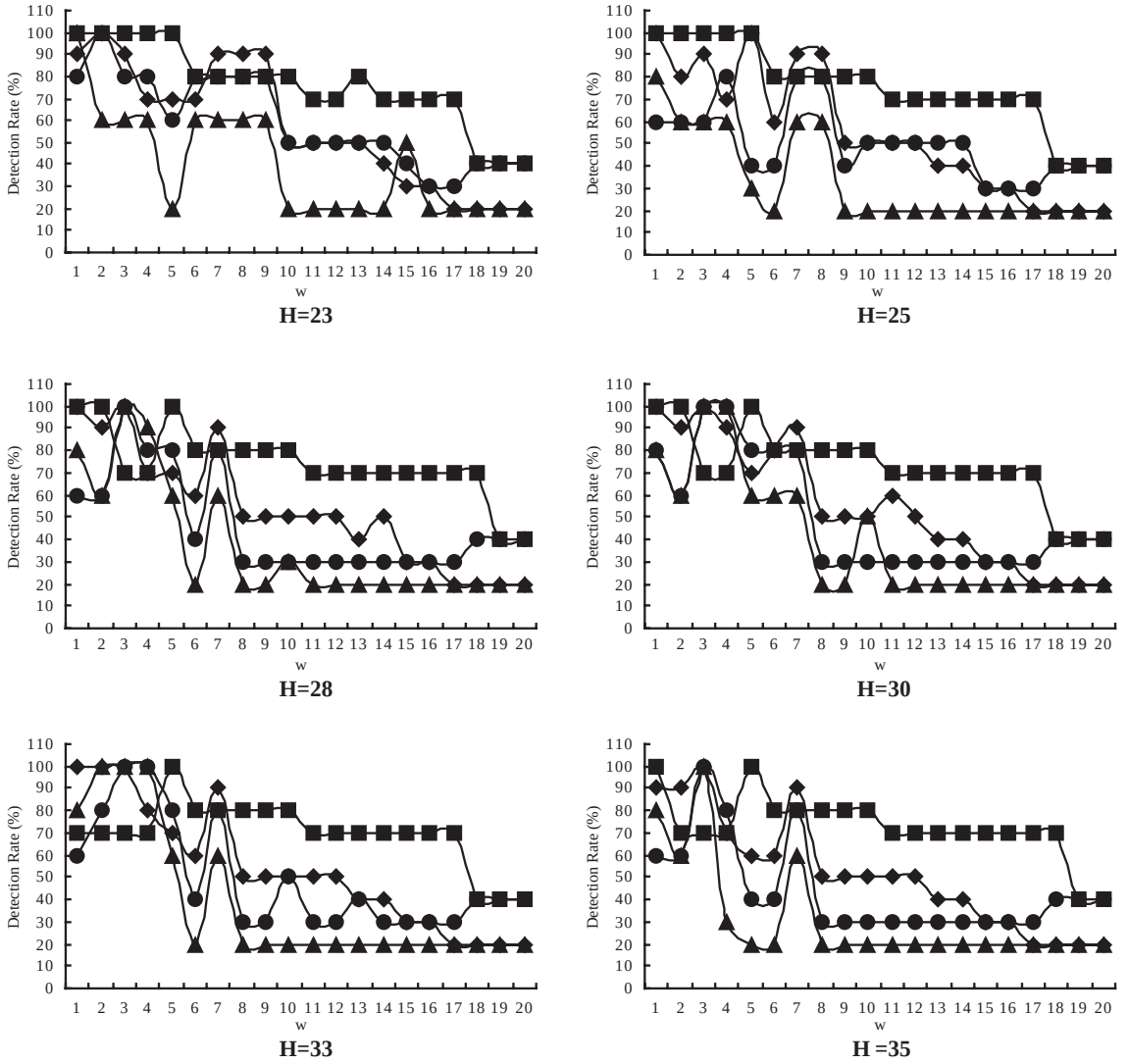

Fig. 14. Detection rate of data set 5 for attack category.

5. Conclusions

In modern society, the security of computer networks becomes an increasingly vital issue to be solved. Traditional detection methods lack extensibility in face of changing network configurations and adaptability in face of unknown intrusion types. While current machine-learning algorithms need labeled data for training in advance, which is greatly expensive in general. As the types of intrusions are unknown, those clustering algorithms that the designer need specify the number of clusters are not effective for this issue. In this paper, a genetic clustering algorithm for intrusion detection is proposed, which is an unsupervised learning algorithm with two stages. The first stage takes the nearest neighbor method to group network data, which avoids the limitation of the clustering center in some clustering problems. The second one is a genetic optimization process to obtain the near optimal detection result. The main purpose of the first stage is to reduce the size of data objects to a moderate one so as to be suitable for genetic algorithms in the second stage. The experimental results show that the IDBGC algorithm is feasible and effective for intrusion detection.

Acknowledgements

The authors are greatly indebted to anonymous referees for their constructive comments. Meanwhile, we thank Dr. E. Eskin for his valuable suggestions on this paper.

This research is supported in part by the National Natural Science Foundation of China (Grants 60273049, 90104005 and 60271019), the Doctoral Foundation of Ministry of Education of China (Grant 20020611007) and the applying basic research program of the Committee of Science and Technology of Chongqing (Grant 7370).

References

[1] R. Heady, G. Luger, A. Maccabe, M. Servilla, The architecture of a network level intrusion detection system, Technical report CS90-20, University of New Mexico, 1990.
[2] W. Lee, A data mining framework for constructing features and models for intrusion detection systems, Ph.D. Thesis, Columbia University, USA, 1999.
[3] J. Allen, et al., State of the practice of intrusion detection technologies, Technical Report CMU/SEI-99-TR-028, Software Engineering Institute, Carnegie Mellon University, 1999.
[4] D.E. Denning, An intrusion-detection model, IEEE Trans. Software Eng. 13 (1987) 222–232.
[5] D.E. Denning, P.G. Neumann, Requirements and model for IDES—A real-time intrusion detection system, Technical report, Computer Science Laboratory, SRI International, USA, 1985.
[6] T. Lunt, Detecting intruders in computer systems, in: Proceedings of the 1993 Conference on Auditing and Computer Technology, Canada, 1993, http://www.ccert.edu.cn/documents/intrusion.pdf.
[7] E. Eskin, Anomaly detection over noisy data using learned probability distributions, in: Proceedings of the International Conference on Machine Learning, USA, 2000, pp. 255–262.
[8] W. Lee, S.J. Stolfo, K. Mok, Data mining in work flow environments: experience in intrusion detection, in: Proceedings of the 1999 Conference on Knowledge Discovery and Data Mining (KDD-99), San Diego, 1999, pp. 114–124.
[9] W. Lee, S.J. Stolfo, Data mining approaches for intrusion detection, in: Proceedings of the 1998 USENIX Security Symposium, San Antonio, 1998, pp. 79–84.
[10] M.R. Anderberg, Cluster Analysis for Application, Academic Press, New York, 1973.
[11] A.K. Jain, R. Dubes, Algorithms for Clustering Data, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[12] S.Z. Selim, M.A. Ismail, K-means-type algorithm: generalized convergence theorem and characterization of local optimality, IEEE Trans. Pattern Anal. Mach. Intell. 6 (1984) 81–87.
[13] W.L. Koontz, P.M. Narendra, K. Fukunaga, A branch and bound clustering algorithm, IEEE Trans. Comput. c-24 (1975) 908–915.
[14] S.Z. Selim, K.S. Al-Sultan, A simulated annealing algorithm for the clustering problem, Pattern Recognition 24 (1991) 1003–1008.
[15] R.W. Klein, R.C. Dubes, Experiments in projection and clustering by simulated annealing, Pattern Recognition 22 (1989) 213–220.
[16] G.P. Babu, M.N. Murty, Clustering with evolution strategies, Pattern Recognition 27 (1994) 321–329.
[17] C.A. Murthy, N. Chowdhury, In search of optimal clusters using genetic algorithms, Pattern Recognition Lett. 7 (1996) 825–832.
[18] R. Cucchiara, Genetic algorithms for clustering in machine vision, Mach. Vision Appl. 11 (1998) 1–6.
[19] M.C. Cowgill, R.J. Harey, A genetic algorithm approach to cluster analysis, Comput. Math. Appl. 37 (1999) 99–106.
[20] K. Krishna, M.N. Murty, Genetic K-Means Algorithm, IEEE Trans. Syst. Man Cybern. Part B 29 (1999) 433–439.
[21] Y.C. Chiou, L.W. Lan, Genetic clustering algorithms, Eur. J. Oper. Res. 135 (2001) 413–427.
[22] U. Maulik, S. Bandyopadhyay, Genetic algorithm-based clustering technique, Pattern Recognition 33 (2000) 1455–1465.
[23] S. Bandyopadhyay, U. Maulik, An evolutionary technique based on K-Means algorithm for optimal clustering in R^N, Inf. Sci. 146 (2002) 221–237.
[24] S. Bandyopadhyay, U. Maulik, Genetic clustering for automatic evolution of clusters and application to image classification, Pattern Recognition 35 (2002) 1197–1208.
[25] L.Y. Tseng, S.B. Yang, A genetic approach to the automatic clustering problem, Pattern Recognition 34 (2001) 415–424.
[26] D.J. Kim, Y.W. Park, D.J. Park, A novel validity index for determination of the optimal number of clusters, IEICE Trans. Inf. Syst. E84-D (2001) 281–285.
[27] Z. Michalewicz, Genetic Algorithm + Data Structure = Evolution Program, Springer, New York, 1996.
[28] S. Kirpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[29] KDD99 cup dataset, http://kdd.ics.uci.edu/databases/kddcup99/kddcup1999.html, 1999.

About the Author—YONGGUO LIU received the B.S. degree in mechanical engineering from Sichuan Institute of Light Industry and Chemical Technology in 1997, the M.S. degree in mechanical engineering from Sichuan University in 2000 and the Ph.D. degree in computer science from Chongqing University in 2003. Currently, he is a postdoctoral researcher at Shanghai Jiaotong University, China. His research interests include pattern recognition, intrusion detection and clustering analysis.

About the Author—KEFEI CHEN was born in Beijing, China. He received the B.S. and the M.S. degree in applied mathematics from Xidian University, Xi'an, in 1982 and 1985, respectively, and the Ph.D. degree from Justus-Liebig University Gießen, Germany in 1994. Dr. Chen is currently a professor, vice dean of School of Information Security Engineering, director of Cryptography and Information Security Lab. He is invited as a consultant of Shanghai Science and Technology Committee, and a member of Appraisal Committee (Information Science) for National Natural Science Award of China 2001/2002, he also is a member of the academic committee both of the State Key Laboratory of Information Security at the Chinese Academy of Sciences and the State Key Laboratory for Novel Software Technology at Nanjing University. His research interests include cryptology and information security.

About the Author—XIAOFENG LIAO was born in Sichuan province, China in 1964. He received the B.S. and the M.S. degree in Mathematics from Sichuan University, Chengdu, China in 1986 and 1992, respectively. He received the Ph.D. degree in circuits and systems from University of Electronic Science and Technology of China in 1997. Form 1999 to 2001, he was a postdoctoral researcher at Chongqing University. At present, he is a professor at Chongqing University and University of Electronic Science and Technology of China. From November 1997 to April 1998, he was a researcher associate at Chinese University of Hong Kong. From October 1999 to October 2000, he was a researcher associate at City university of Hong Kong. From March 2001 to June 2001 and March 2002 to June 2002, he is a senior researcher associate at City University of Hong Kong, respectively. He has published over 100 academic journal and conference papers. Currently, his main interest include neural networks, nonlinear dynamical systems, bifurcation and chaos, synchronization and control of chaos, and signal processing.

About the Author—WEI ZHANG received the B.S. degree from Chongqing University in 1992, the M.S. degree from Southwest China Normal University in 1999 and the Ph.D. degree from Chongqing University in 2003, all in computer science. He is currently an associate professor, dean of Department of Computer & Modern Education Technology, Chongqing Education College, China. His research interests include information security and data mining.