Combatting Financial Fraud: A Coevolutionary Anomaly Detection Approach

Shelly Xiaonan Wu
Computer Science Department
Memorial University of Newfoundland
St John's, Canada, A1B 3X5
xiaonan@cs.mun.ca

Wolfgang Banzhaf
Computer Science Department
Memorial University of Newfoundland
St John's, Canada, A1B 3X5
banzhaf@cs.mun.ca

ABSTRACT

A major difficulty for anomaly detection lies in discovering boundaries between normal and anomalous behavior, due to the deficiency of abnormal samples in the training phase. In this paper, a novel coevolutionary algorithm which attempts to simulate territory establishment in ecology is conceived to tackle anomaly detection problems. Two species in normal and abnormal behavior pattern space coevolve competitively and cooperatively. Competition prevents individuals in one species from invading the other's territory; cooperation aims to achieve complete pattern coverage by adjusting the evolutionary environment according to the pressure coming from neighbors. In a sense, we extend the definition of cooperative coevolution from "coupled fitness" to "interaction of the evolutionary environment". This coevolutionary algorithm, enhanced with features like niching inside of species, global and local fitness, and fuzzy sets, tries to balance overfitting and overgeneralization. This provides an accurate boundary definition. Experimental results on transactional data from a real financial institution show that this coevolutionary algorithm is more effective than the evolutionary algorithm in evolving normal or abnormal behavior patterns only.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods

General Terms

Algorithms, Experimentation.

Keywords

Anomaly detection; Coevolutionary algorithms; Ecology; Self-nonself space pattern construction; Parisian Approach; Fuzzy logic; Financial Fraud

GECCO'08, July 12-16, 2008, Atlanta, Georgia, USA

1. INTRODUCTION

Financial fraud is a serious threat to peaceful economic development. According to the Retail Council of Canada, credit card fraud in Canada alone resulted in losses of $201 million to major credit card companies in 2005; debit card fraud resulted in losses of $70.4 million [22]. Although information or regular patterns about fraudulent transactions are often scarce or even non-existent, fraudsters tend to show distinct behavior from legitimate card holders. Financial fraud is therefore an ideal application for anomaly detection algorithms.

The traditional way of detecting anomalies is to build a model for normal behavior. Any deviation from the model is labeled as an anomaly. Artificial immune system (AIS) approaches, however, address anomaly detection by modeling abnormal behavior. The negative selection algorithm (NSA) in AIS removes all potential patterns that match self samples (normal data), thus resulting in a set of nonself patterns. Any matching to nonself patterns is labeled as an anomaly.

Forrest *et al.* [7] first proposed the NSA. However, binary matching rules and the need to exhaustively generate detectors are two weaknesses in their model. An empirical study conducted by Gonzalez *et al.* [8] pointed out that binary matching rules cannot properly process intrinsic meaning of difference or similarity in numeric data. They suggested fuzzy rules [9], or real-valued representations to encode numeric information. The later represented detectors as a vector of real numbers, expressing various sizes [14, 19] and geometric shapes [2, 4, 10, 19, 21]. Exhaustively generating detectors requires excessive computational time and unmanageable numbers of detectors, especially when the size of self patterns grows [16]. In fact, antibodies in the human immune system (analogous to detectors in AIS) are generated by random combinations of a set of gene segments. In addition, evolution is observed in the clonal selection process to produce effective antibodies. Therefore, Kim *et al.* [15] and Dasgupta *et al.* [4] both suggested applying evolutionary approaches, instead of randomly generating detectors. Recent trends in NSA, e.g. [2, 9, 19, 20, 21], are employing evolutionary algorithms with niching to evolve a set of detectors in different shapes and sizes.

Appropriately representing self data is a very important issue in NSA. Since self data are never complete, if one uses self data directly, the final detector set would be very sensitive to unseen self data. For this reason, Dasgupta *et al.* introduce a variability range r [4], which can be regarded as the radius from a self sample. They assume that a circular area around a self data point was entirely normal. Although

this method provides some generality, the constant value of r produces the "boundary dilemma" [13]. Hang *et al.* and Ostaszewski *et al.* both devised the idea of constructing self patterns from self samples. Hang [11] exploited a "co-evolutionary" genetic algorithm to evolve normal patterns. Essentially, each subpopulation evolved a particular pattern of normal behavior. In order to restrain self patterns from covering nonself space, nonself data was required to remove these overgeneral self patterns. Ostaszewski *et al.* [18, 19] extended the variability range r to a vector, which saved intervals for every dimension, thus allowing the construction of more accurate self patterns. A competitive "coevolutionary" algorithm was used in their research to guarantee the coverage of nonself patterns. However, in our opinion, Hang and Ostaszewski's algorithms cannot be labeled "coevolutionary". Coevolution means reciprocal evolutionary change in interacting species. In Hang's work there is no interaction between subpopulations; in Ostaszewski's work the second population consists of constant nonself samples. No evolution happens in this population.

In summary, anomaly detection is searching for boundaries between self and nonself space by taking only self data as training samples. Such boundaries should best minimize the impact of overfitting and overgeneralization, which cause false positive and false negative errors (shown in Figure 1). In this

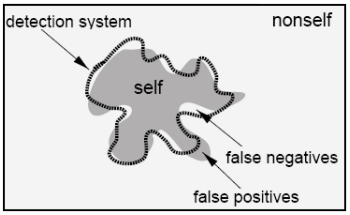


Figure 1: An anomaly detection system attempts to discover the boundary between normal and anomaly [12].

paper, we propose a coevolutionary algorithm, an extension of current NSA research, to address the "boundary dilemma". This algorithm is inspired by a phenomenon from ecology that colonial species establish territories through interaction with neighbors. In our algorithm, fuzzy self and nonself patterns are evolved simultaneously as two species. However, they also compete and cooperate mutually. Through the competitive interactions between species, and niching inside of species, appropriate generality of self and nonself patterns will be produced. Cooperative interactions promote and achieve thorough space coverage in each species by adjusting the evolutionary environment. Fuzzy patterns give a natural estimate of the amount of deviation from normal, hence providing a better definition of the boundaries. Experimental results demonstrate that this coevolutionary algorithm is effective than the evolutionary algorithm in searching only for self or nonself patterns.

The remainder of this paper is organized as follows: Section 2 presents a general framework of our algorithm; section 3 describes the algorithm in detail; section 4 discusses the data set, experimental setup and experimental results; section 5 offers conclusion and discusses for future work.

2. GENERAL FRAMEWORK

In nature, aggressive animals, such as lions, some ants and birds, forage and occupy exclusive territories. Territorial owners are bound socially to adjacent neighbors, due to the diverse roles neighbors play in territory acquisition. They can be enemies. Their territories are established by boundary disputes among neighboring residents. Each resident applies pressure, such as fighting and display, against its neighbors. Numerous experiments have shown that neighbors restrict one another's territory area. As a result, the boundary will be formed along where pressure exerted by adjacent residents is equal. They also can be friends when they encounter an unsettled group searching for a territory [1].

This interesting ecological phenomenon sheds light on discovering the boundaries in anomaly detection. Two species in the form of if-then rules, one living in anomaly space, and the other living in normal space, develop independently, and forage and expand their own territories. When either of them intrudes a neighbor's territory, it will be fought back. In the end, boundaries between normal and anomaly space will be defined by the outcome of interactions with neighbors. Compared with the evolution of only one species, coevolution in this context has two main advantages. Firstly, two species constrain each other from being overgeneral (over the boundaries), hence effectively decrease false negative and false positive errors; secondly, they reference each other. By sensing the pressure given by its neighbors, a species is able to adjust its evolutionary process. When the pressure that a species receives from its neighbor becomes intensive, generating an individual who can successfully claim an uncovered space becomes difficult. Hence, the rules size, crossover rate, mutation rate, and selection pressure should adapt to pressure changing in order to generate qualified individuals.

The general framework of our algorithm is described in Figure 2. Details will be discussed in the next section.

We start our algorithm with initializing the populations. We build rules in the first population of the positive species by feeding input data as seeds rather than starting from scratch. This is due to the fact that each positive rule must cover at least one normal data point, and due to the assumption of anomaly detection: normal behavior exhibits predictable, consistent patterns. For the negative species, we randomly generate rules in the initial population because it should cover everything else.

When updating populations, we choose a steady-state approach. In this model, the currently best rules are automatically maintained in the population, while weaker rules are being replaced by stronger ones. Hence, as the evolutionary process proceeds, a rule set will incrementally be built as a single solution.

Here we define two populations in each species, one is called new population (denoted as P_n), which is the initial population or the population generated by crossover and mutation; the other is the current population (denoted as P_c) which saves the best rules found so far and is used to breed a new generation. P_n will be merged into P_c by inserting, deleting and replacing rules. Subsequently, a niching method is applied on P_c to minimize overlaps inside a species.

After the two species are evolved in parallel, coevolution will bring them together, and check if there are any rule's overlaps ($r_p \cap r_n$) between two P_cs. Overlaps imply that some rules cover the wrong territories. Overlapped rules have to fight, thus coevolution here is competitive. By fighting, the boundary dispute is resolved. Furthermore, the incompleteness of rule sets has to be addressed. Generating effective

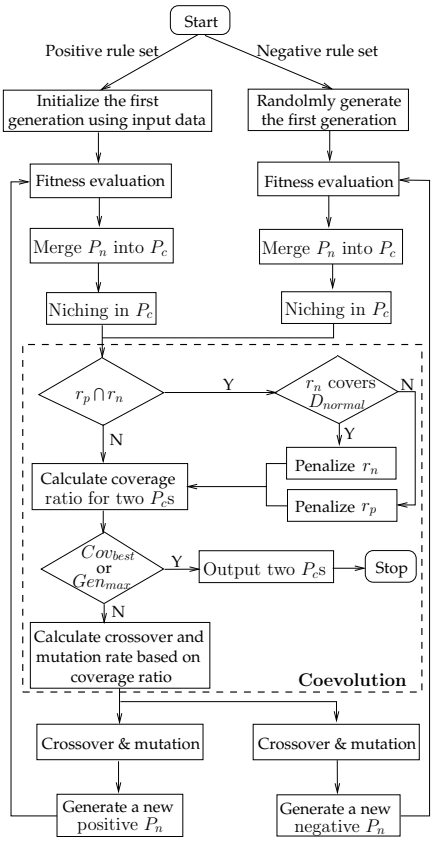


Figure 2: Framework of Generating Anomaly Detectors by Coevolutionary Algorithm. P_n denotes the new population; P_c the current population; r_p a positive rule; r_n a negative rule; D_{normal} normal data samples; Cov_{best} desired coverage; Gen_{max} maximum number of generations.

rules to fill holes will reduce detection errors. To this end, besides niching and generating variable-sized detectors, a dynamic evolutionary environment is controlled by cooperative coevolution. We first estimate the coverage ratio of one species by referring to the coverage of the other species; then we use this ratio to adjust its evolutionary process in order to increase the chance of generating a useful rule. Cooperativity here means that two species help each other to achieve maximum coverage.

Crossover and mutation are conducted on P_c of each species to generate P_n. We repeat this process until a desired coverage or the maximum number of generations is reached.

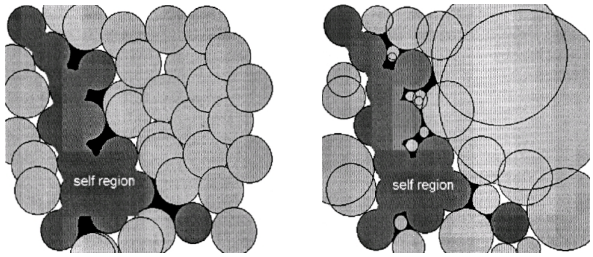
3. THE ALGORITHM

Two evolutionary algorithms are used to control the evolution of the two species. Both of them are instances of genetic algorithms. This section describes implementation details such as the representation, fitness functions, and coevolutionary processes.

3.1 Representation

Each individual in our algorithm is a fuzzy rule with a condition part and a class label part. Since rules in a species have the same class label, we only need to represent the condition part as a chromosome, as shown in Figure 4(a). A chromosome contains n genes, which correspond to n attributes in the input data. Each gene consists of m nucleotides, where m is the number of attribute values. The default logic operators are "AND" between genes and "OR" between the nucleotides.

Ji and Dasgupta suggested using variable-sized detectors, other than constant-sized detectors [14]. As illustrated in Figure 3(a), a large number of constant-sized detectors are

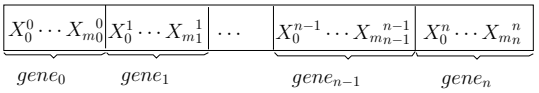


(a) Constant-sized detectors *(b) Variable-sized detectors*

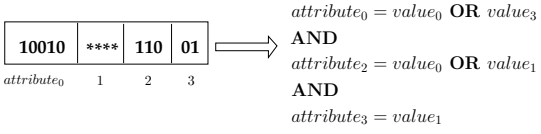
Figure 3: Constant-sized detectors V.S. Variable-sized detectors. The dark area represents self region. The light gray circles are detectors in the nonself region [14].

needed to cover the large area of nonself space, while no detectors can fit in the small area of nonself space, especially near the boundaries between self and nonself, thus leaving holes. Variable-sized detectors deal with both issues, as shown in Figure 3(b). The large area of non-self space is now covered by much fewer but larger detectors; smaller detectors work much better to cover the holes.

Learning classifier systems and Hang *et al.* [11] encode detectors on a ternary alphabet $\{0, 1, *\}$. The size of detectors is controlled by "don't care" symbol $(*)$, a symbol matching any value. The more $*$ symbols are in a rule, the more space the rule occupies, thus the more general it is. In our algorithm, we follow this encoding scheme. Figure 4(b) illustrates a mapping from a genotype to a phenotype.



(a) Structure of a chromosome



(b) Genotype to phenotype

Figure 4: Genotype and Phenotype

3.2 Fitness Function

Our algorithm is not expected to produce a dominant individual, but a population working together as a solution. Hence, during the evolution, performance of both individuals and populations should be measured. In addition, an individual's contribution to the final solution should be considered. If an individual improves overall performance, it will be rewarded by a positive feedback; otherwise, it will be penalized by a negative feedback. Actually this is the idea of Parisian Approach [3, 5]. We adopt this approach and

define a local fitness for individuals and a global fitness for populations.

Throughout this section some terms are used in fitness functions. Their definitions are as follows:

- Rule (r_i): the genotype of an individual, as shown in Figure 4(b). Positive rules describe normal behavior, while negative rules describe abnormal behavior.
- Atomic rule (ar): a rule in which only one bit is set to 1 in each gene. The rule shown in Figure 4(b) contains 16 ($2 \times 4 \times 2 \times 1$) atomic rules. * symbol in a gene can be considered as 1 in every bit.
- Specificity of a rule ($N_{specificity}$): $\frac{\#\ of\ 1's\ in\ a\ rule}{\#\ of\ bits\ in\ a\ rule}$. For example, the $N_{specificity}$ of the rule in Figure 4(b) is $\frac{2+4+2+1}{5+4+3+2}$.
- Rule space coverage ratio of a species (R_r): $\frac{\#^A_{ar}}{\#^A_{ar'}}$, where $\#^A_{ar}$ is the number of unique atomic rules in species A, and $\#^A_{ar'}$ is the number of atomic rules in the space where species A belongs to. In other words, this ratio is the space currently covered by species A over the space that should be covered by species A. Because $\#^A_{ar'}$ cannot be calculated, we use $\#_{all} - \#^B_{ar}$ as an approximation, where $\#_{all}$ is the number of atomic rules in whole searching space, and $\#^B_{ar}$ is the number of unique atomic rules in species B. As the evolutionary process of species B proceeds, $\#_{all} - \#^B_{ar}$ approaches $\#^A_{ar'}$. This ratio has multiple roles: controlling the adaptive evolution in cooperative coevolution, measuring performance of *populations*, and being a termination criterion of the coevolution.

These terms are suitable for both negative and positive rules.

- Data coverage ratio of a rule (R_{data}): $\frac{\#\ of\ data\ a\ rule\ actually\ covers}{\#\ of\ data\ a\ rule\ should\ cover}$. This ratio is used as a performance measurement of *a single positive rule*. For example if a rule covers 8 data points out of 10 it should cover, then the R_{data} is 0.8.
- Data coverage ratio of a species (R_d): $\frac{\#\ of\ data\ a\ population\ covers}{\#\ of\ normal\ data}$. This ratio is used as a performance measurement of *a positive population*. For example if a population covers 98 out of total 100 normal data in the training set, then the R_d is 0.98.

The above terms are used only for positive rule sets, because only positive rules cover normal data.

- Overlaps ($N_{overlap}$): the space shared by more than one rule. We define it as *# of atomic rules shared by two rules*.

We define our local and global fitness functions as follows:

- Local fitness for positive individuals.

$$f_{lp}(x) = \lambda \times R_{data} + (1-\lambda) \times (1 - N_{specificity}), \quad (1)$$

where λ is a weighting coefficient. Rules which cover more data with less space will have higher fitness.

- Local fitness for negative individuals.

$$f_{ln}(x) = 1 - \frac{min(N_{specificity}, N_{generation})}{max(N_{specificity}, N_{generation})}, \quad (2)$$

$N_{generation}$ is the generation number scaled between 0 to 1. We prefer general negative rules at the beginning of the evolutionary process, and specific rules later.

- Global fitness for the positive rule set in t generation.

$$f_{gp}(t) = \lambda \times R_d + (1-\lambda) \times (1 - \frac{\#^p_{ar}}{\#_{all}}), \quad (3)$$

where λ is a weighting coefficient. A positive population is marked as good when it covers more data with less space. That is to say that this fitness function favors populations with less atomic rules provided they have the same data coverage ratio.

- Global fitness for the negative rule set in t generation

$$f_{gn}(t) = \frac{\#^n_{ar}}{\#_{all}} \quad (4)$$

The more space the rule set covers, the better the negative population is.

When a positive rule covers new data points, or a negative rule covers a new niche, we will reward these rules as

$$f^{t+1}_{lp}(x) = f^t_{lp}(x) \times (1 + R^{t+1}_d - R^t_d), \quad (5a)$$

$$f^{t+1}_{ln}(x) = f^t_{ln}(x) \times (1 + f_{gn}(t+1) - f_{gn}(t)), \quad (5b)$$

where superscript denotes the generation number.

The advantage of considering feedback from global fitness is to prevent rare patterns or important patterns with small fitness from being replaced. It offers another way to maintain diversity in the population, especially for skewed datasets.

3.3 Coevolution

Our framework integrates both competitive and cooperative coevolution. Competitive coevolution happens first with the intention of picking up rules who cross boundaries. Crossing boundaries means two individuals from different species cover the same part of the territories. Since territories are the limited and exclusive resources two species are fighting for, we assign it by the following principle: if a positive rule r_p overlaps with a negative rule r_n, and r_n covers at least one normal sample, then r_n should be penalized; if r_n covers no normal data, which implies r_p covers part of the nonself space, then we penalize r_p. Note that the purpose of coevolution is not to kill an individual but to decrease its fitness, as it may contribute genetic material to future generations. However, for weakened individuals, the odds of being replaced by stronger ones are increased. Fitness is updated according to Equation (6) every time an overlap is detected. $f_{ar}(i)$ is the number of atomic rule i has.

$$f_{local}(i) = f_{local}(i) * (1 - \frac{N_{overlap}}{f_{ar}(i)}), \quad (6)$$

Coevolution in ecology is defined as *describing interactions like these where adaptation in one species has evolved in relation to adaptation in another* [17], so interactions in coevolution should not necessarily be constrained only to "coupled fitness". We extend the interaction to the evolutionary environment, including crossover rate, mutation rate,

and selection pressure. Like fitness, the evolutionary environment affects evolution. In our algorithm, the evolutionary environment of a species is controlled by its coverage ratio R_r. As previously analyzed, when R_r of a species is close to the desired value, generating a useful individual in this species is difficult. For this reason, we will increase the mutation rate, decrease the crossover rate and the selection pressure, and prefer rules in smaller size as R_r grows. A higher crossover rate at the beginning of the evolutionary process helps to generate rules which cover a large area in the space; rules in a smaller size as well as a higher mutation rate and lower selection pressure help to generate more rules which jump away from the current optimum, or fill holes. This is cooperative coevolution, because R_r is decided by the coverage of two species. In other words, two species cooperatively adjust each other's evolutionary environment to achieve the best coverage.

The evolutionary environment is updated as follows:

$$p_{xover} = p^u_{xover} - (p^u_{xover} - p^l_{xover}) \times R_r \tag{7a}$$

$$p_{mt} = p^l_{mt} + (p^u_{mt} - p^l_{mt}) \times R_r \tag{7b}$$

$$ts = ts^u - [(ts^u - ts^l) \times R_r] \tag{7c}$$

where p_{xover}, p_{mt}, ts are the crossover rate, the mutation rate, and the tournament size respectively; superscript l means the lower bound, and superscript u means the upper bound; [.] means round to the nearest integer; R_r is the coverage ratio, which considers the coverage of another species.

3.4 Niching

Forrest *et al.* confirmed the role of niching in discovering and maintaining multiple peaks [6]. Other authors [4, 15, 19, 20] applied niching to keep detectors separated in order to maximize the space coverage and minimizes the overlaps among them, thus avoiding holes. In our algorithm, we employ niching for the same reason. Our niching equation is defined as

$$f_{local}(i) = f_{local}(i) * \left(1 - \frac{\sum_{i=1}^{n} N_{overlap}}{n \times f_{ar}(i)}\right), \tag{8}$$

For individual i, we first calculate the average overlap between i and other individuals in the population, then we decrease the fitness by the proportion of this average and $f_{ar}(i)$, the number of atomic rules i has.

This fitness updating function protects rules of large size from decreasing too fast in fitness. We favor large rules because a population can carry only a limited number of patterns. If we replaced large rules with several small rules, they would take too much space without improving coverage.

3.5 Evolutionary Operators

Genetic operators allow passing genetic information between generations. Basically, the well-known operators of a typical GA include selection, crossover and mutation. Except for these three, here we introduce two new operators, deletion and replacement. These two are useful in identifying weak individuals who will be replaced in a steady-state scheme.

(a) Selection. The algorithm uses tournament selection to select parents. Tournament size decreases when space coverage increases. Decreasing tournament size actually implies lessening the competitive pressure. As more rules have a higher possibility to reproduce, the chance of finding new local optimal is increased.

(b) Crossover and mutation. Crossover and mutation operators are applied on selected individuals to produce a new generation. Classical two-point crossover and bit-flip mutation are applied.

(c) Replacement. Replacement happens when a new population P_n is merged into a current population P_c. Replacement is based on the subsumption relationship between two individuals, which means a rule logically subsumes another rule. For positive rules, if rule r_i in P_c covers the same data points as rule r_j in P_n does, but r_i subsumes r_j, then r_i is replaced by r_j. This is because r_j uses less space to cover the same amount of data. For negative rules, we prefer rules with larger space coverage under the premise that they cover no normal data. Hence, we will delete r_j if r_i subsumes r_j. If two rules do not have a subsumption relationship, we simply copy them from P_n to P_c. This operator effectively adjusts the generality of self and nonself patterns.

(d) Deletion. This operator removes rules in P_c if we attempt to insert rules from P_n into P_c, but P_c is full. Currently we simply delete the least fit rule.

3.6 Fuzzy rules

Normality is not a crisp concept, so defining a sharp distinction between normal and anomaly is improper. Gonzalez *et al.* [9] report that fuzzy rules provide a natural estimate of the amount of deviation from normal, and offer some generality for numeric attributes. Hence, they can avoid overfitting, improve detection accuracy, and present results in a human comprehensible way.

For fuzzy logic to be involved in our framework, first we apply the *k-means* clustering algorithm to determine fuzzy sets and their membership functions for continuous attributes. Membership functions are presented in trapezoidal shapes, where the first and the last trapezoid have open ends. The membership function for category attributes always outputs 0 or 1. Table 1, then, are used in the inference step to map logic operators, e.g. "AND" and "OR", in fuzzy rules to fuzzy operators. p and q are fuzzy sets, x is a crisp value, and μ is the membership function associated with a fuzzy set.

Table 1: Fuzzy logic operators

Logic Operator	Fuzzy Operator
p AND q	$\min\{\mu_p(x), \mu_q(x)\}$
p OR q	$\max\{\mu_p(x), \mu_q(x)\}$

When given input data D_t and a set of rules, we plug crisp values from D_t into the membership function of the corresponding fuzzy sets in rule R_i. According to Table 1, we can calculate how well the data D_t matches rule R_i. The matching threshold θ now is set to 0.6.

4. DATA SET AND EXPERIMENTS

In this section, we will evaluate the effectiveness of the proposed coevolutionary algorithm for anomaly detection.

4.1 Data Set

We tested our algorithm on anonymized transactional data from a real financial institution. This data set contains two-year ABM (Automated Bank Machine) and POS (Point Of

Sale) fraud-free transaction history of 3063 customers, with a total of 522,728 transaction records, each with 61 fields. For each customer, 90% of the data was used for training, and 10% was used for testing; further test data was provided by 346 fraud cases. Missing values were replaced by random values within the range of normal values for each field.

In our experiments, we used 5 basic fields, which describe type, time (hours, minutes), location, amount, and date (week days) of a transaction. Transaction type describes customers' operations on ABM or POS, e.g. withdrawal, deposit, or purchase. Location describes where the transaction is conducted. Transaction date tries to capture patterns like payroll deposit. These three fields are category attributes. The other two are continuous attributes. We fuzzified each with a fuzzy set. In addition, 5 statistical indicators were calculated to facilitate identifying anomalies. They recorded the average amount of purchase, withdraw, deposit, purchase at risky locations, as well as the throughput of an account inside an n-transaction sliding window.

4.2 Experimental Results

4.2.1 *Experiment 1 — Effectiveness of Coevolutionary Algorithm*

The first experiment assessed the ability of the coevolutionary algorithm to detect anomalies. Experiments were carried out to compare the results obtained by the coevolutionary algorithm and by a standard evolutionary algorithm in the same running environment. The standard evolutionary algorithm had the same representation, niching and fitness function definition as the coevolutionary one. The parameter setup of two algorithms is shown in Table 2.

Table 2: Parameter Setup

Parameters	Coevolution	Standard Evolution
Num_{runs}	50	50
$Max_{generation}$	500	500
$Size_{population}$	20	20
$P_{crossover}$	$p^u_{xover} = 0.8$	Positive:0.4
	$p^l_{xover} = 0.4$	Negative:0.6
$P_{mutation}$	$p^u_{mt} = 0.6$	Positive:0.4
	$p^l_{mt} = 0.2$	Negative:0.4
$Size_{tournament}$	$ts^u = 6$	4
	$ts^l = 2$	
alarm threshold	positive: < 0.6	
	negative: $>= 0.6$	

Table 3 presents the results of a 50-run experiment for each. The accuracy of this algorithm is described by the detection rate (DR) and the false alarm rate (FAR). Compared to the coevolutionary algorithm, the standard one achieved a high True Negative (TN) rate in the training phase, but a low DR in the testing phase. The reason can be found in the number of atomic rules. The positive rule set generated by the standard one has 146 atomic rules on average, while 102 atomic rules on average for the coevolutionary algorithm. This indicates that the former rule set incorrectly covered some space belonging to negative rules. Overgeneralization causes a low DR. Similarly for negative rules, the rule set generated by the standard one has 482 atomic rules on average. It covers less space than the one from the coevolutionary algorithm, which contains 503 atomic rules on average. If we compare the results of the coevolutionary algorithm, we will notice that the positive rule set has a higher FAR, while the negative rule set has a lower DR. This implies that overfitting and holes still exist in the positive and negative rule set, respectively. We suggest combining their results to achieve a better performance.

In conclusion, our coevolutionary algorithm outperformed the standard evolutionary algorithm in terms of DR. The FAR is slightly higher, but still in an acceptable range. In fact, the cost of false alarms is only a phone call to card holders, which is less than the costs of missing fraudulent events. The better performance is credited to the interactions and adaptive evolutionary environment introduced by coevolution. We will examine their contributions in detail in the following two experiments.

4.2.2 *Experiment 2 — Effectiveness of Competitive Interactions between Two Populations*

The second experiment assessed the effectiveness of competitive interactions between two populations. Competitive interaction is one aspect of our coevolutionary algorithm. It restrains the positive rule set from crossing the boundary and keeps it close to the space the normal data actually occupies.

Figure 5 shows the changes in the number of atomic rules in one run of the coevolutionary algorithm. As we can see,

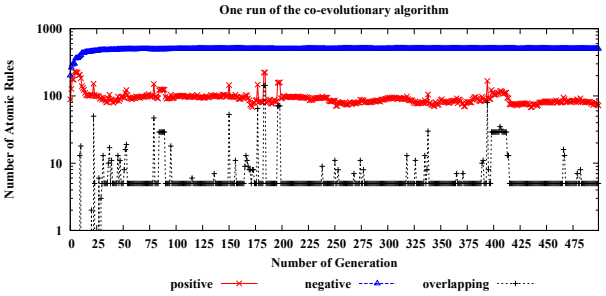


Figure 5: The competitive interactions between two populations will penalize the positive rules which have crossed the boundary indicated by the number of overlapping. As a result, these rules will be removed from the population gradually, and consequently the numbers of positive rules are dropping.

at the beginning of the evolutionary process, the numbers of atomic rules for both positive and negative populations increase quickly. Gradually, overlaps appear between two populations; therefore the interactions will penalize positive rules which have crossed the boundary by Equation (6). As a result, these rules will be removed from the population, and, consequently, the number of positive rules drops.

Another experiment was conducted to compare the coevolutionary algorithm with the standard algorithm. This experiment considers two factors in the positive rule set: rule space coverage and data space coverage. The results of one run for each algorithm are shown in Figure 6 and 7.

In Figure 6, we can see that for most of the time the rule space coverage in the standard algorithm is higher than 1.0, so the rule set is overgeneral. What's more, when rule space coverage drops, often the data space coverage drops too. This phenomenon actually implies that when replacing rules from the population, overlapping positive rules are not given priority. Comparatively speaking, in Figure 7 the data space coverage is hovering around 1.0 while rule space coverage

Table 3: Average Algorithm Performance in 50 Runs

		Training			Testing	
		TN	# of atomic rules	Time	DR	FAR
Coevolution	positive	99.479%	102	76 sec.	98.266%	4.138%
	negative	100%	503		97.688%	2.878%
Standard Evolution	positive	100%	146	23 sec.	92.486%	0.689%
	negative	100%	482	26 sec.	95.376%	0.689%

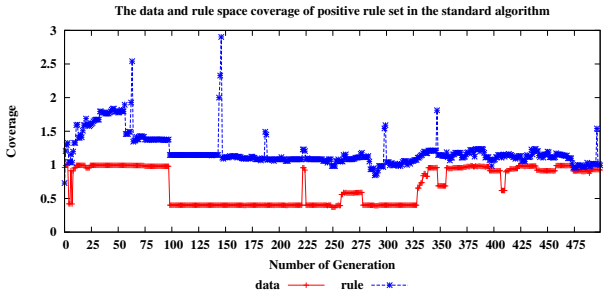


Figure 6: In the standard algorithm, the rule space coverage is higher than 1.0 for most of the time; when rule space coverage drops, often the data space coverage drops too.

fluctuates. So the changes in rule space coverage do not affect data space coverage very much. Accordingly, given same data space coverage, coevolution is inclined to find a rule set with less atomic rules, thus avoiding overgeneralization to a certain degree. This experiment further confirms the advantage of competitive interactions between two populations in coevolutionary algorithm.

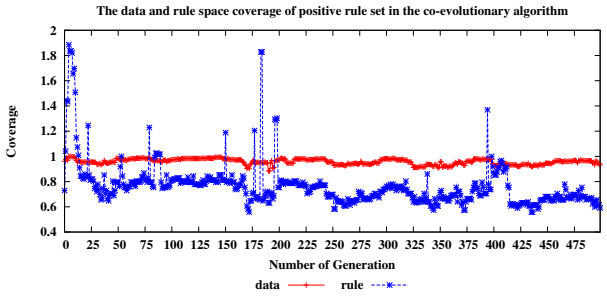


Figure 7: In the coevolutionary algorithm, the data space coverage is hovering around 1.0 while rule space coverage fluctuates. So the changes in rule space coverage do not affect data space coverage very much.

4.2.3 Experiment 3 — Effectiveness of Dynamic Evolutionary Environment

The adaptive evolutionary environment is another aspect of our coevolutionary algorithm. It is controlled by the pressure from the other species. In our algorithm, we increase the mutation probability and decrease crossover probability and tournament size as the estimated negative rule space grows. Compared with positive space, negative space is wider and more continuous. Therefore, a higher crossover rate at the beginning of the evolutionary process helps to cover space with more volume first; a higher mutation rate and a smaller tournament size help to cover holes later. In this experiment, the coevolutionary algorithm is compared with the standard algorithm on generating a negative rule set. One sample run is shown in Figure 8.

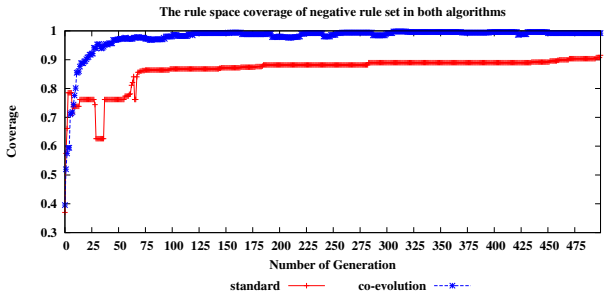


Figure 8: The rule space coverage in the coevolutionary algorithm increased quicker than in standard evolutionary algorithm.

We can easily see that the rule space coverage in the coevolutionary algorithm increases from 0.4 to nearly 1.0 during the first 100 generations. For the rest of the generations, though the increase is very small, it is constantly improving toward 1.0. However, rule space coverage in the standard one increases slowly. In the end it might find the same results, but it will require many more generations. The average number of atomic rules (Avg_{ar}) and the generation returns the best result (Gen_{best}) in 50 runs are shown in Table 4.

Table 4: Avg_{ar} and Gen_{best} for Negative Rule Set

	Avg_{ar}		Gen_{best}	
	Mean	SDV	Mean	SDV
Coevolution	503.42	3.64	336.78	42.90
Standard	482.42	9.64	461.76	30.26

5. CONCLUSION

This paper proposed a coevolutionary algorithm to tackle the anomaly detection problem, which, in our opinion, is a process of discovering the boundary between normal and abnormal behavior. Inspired by territory establishment in ecology, this boundary is identified through the cooperative and competitive interactions between self and nonself patterns. Competition penalizes individuals who cross the boundary, and cooperation adjusts evolutionary process of a species. Experimental results on detecting financial fraud demonstrated that this coevolutionary algorithm is more effective than the standard algorithm in the evolution of negative or positive rule set only.

This coevolutionary algorithm has the following advantages: competitive interactions between populations minimize detection errors; the adaptive evolutionary environment

accelerates the process of finding good solutions; global fitness and local fitness provide good criteria for selecting solutions; the rewarding mechanism helps to maintain useful individuals but with small fitness, hence a complete rule set can be guaranteed; Fuzzy rules improve detection accuracy, and allow to present results in a human comprehensible way.

This work is our preliminary effort to map an anomaly detection system to territory acquisition. There are many issues in need of further study. First of all, the fighting between two species should be modeled more accurately. Currently we simply consider the overlapping area as a factor. In fact, the fighting ability in ecology is decided by biomass of the defending colony, territory size and the distance to the colony's nest [1]. Secondly, we oversimplified the cooperation between populations. Actually, cooperation plays an important role in the colony establishment process, so it should be thoroughly researched in ecology, and then carefully mapped to a computer algorithm. Thirdly, to get further insight from our algorithm, we will need to apply it to some benchmark data.

6. ACKNOWLEDGMENTS

W.B. would like to acknowledge funding from NSERC under the MITACS NCE and from Verafin Inc., St. John's. S.X.W. is grateful for the opportunity of an internship at Verafin Inc., under the guidance of Charles Roberson.

7. REFERENCES

[1] E. S. Adams. Territory size and shape in fire ants: a model based on neighborhood interactions. *Ecology*, 79(4):1125–1134, June 1998.

[2] S. Balachandran, D. Dasgupta, F. Nino, and D. Garrett. A general framework for evolving multi-shaped detectors in negative selection. In *IEEE Symposium Series on Computational Intelligence*, Honolulu, Hawaii, April 2007.

[3] P. Collet, E. Lutton, and F. Raynal. Polar IFS + parisian genetic programming = effcient IFS inverse problem solving. *Genetic Programming and Evolvable Machines*, 1:339–361, 2000.

[4] D. Dasgupta and F. Gonzalez. An immunity-based technique to characterize intrusions in computer networks. *IEEE Transactions on Evolutionary Computation*, 6(3):281–291, June 2002.

[5] E. Dunn, G. Olague, and E. Lutton. Automated photogrammetric network design using the parisian approach. In *Applications on Evolutionary Computing*, pages 356–365. Springer Berlin / Heidelberg, 2005.

[6] S. Forrest, B. Javornik, R. E. Smith, and A. S. Perelson. Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation*, 1(3):191–211, 1993.

[7] S. Forrest, A. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 202–212, 1994.

[8] F. Gonzalez, D. Dasgupta, and J. Gomez. The effect of binary matching rules in negative selection. In *Genetic and Evolutionary Computation Conference (GECCO '03)*, pages 195–206, Chicago, IL., US, 12-16 July 2003.

[9] F. Gonzalez, J. Gomez, M. Kaniganti, and D. Dasgupta. An evolutionary approach to generate fuzzy anomaly signatures. In *Proceedings of the Fourth Annual IEEE Information Assurance Workshop*, pages 251–259. West point, NY, 2003.

[10] F. A. Gonzalez and D. Dasgupta. Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines*, 4(4):383–403, December 2003.

[11] X. Hang and H. Dai. Applying both positive and negative selection to supervised learning for anomaly detection. In *Genetic and Evolutionary Computation Conference (GECOO '05)*, 2005.

[12] S. A. Hofmeyr. *An immunological model of distributed detection and its application to computer security*. PhD thesis, The University of New Mexico, 1999.

[13] Z. Ji. A boundary-aware negative selection algorithm. In *Proceedings of the 9th IASTED International Conference on Artificial Intelligence and Soft Computing (ASC 2005)*, Benidorm, Spain, 2005.

[14] Z. Ji and D. Dasgupta. Real-valued negative selection using variable-sized detectors. In *Genetic and Evolutionary Computation Conference (GECCO '04)*, Seattle, Washington, 26-30 June 2004.

[15] J. Kim and P. Bentley. Negative selection and niching by an artificial immune system for network intrusion detection. In *Genetic and Evolutionary Computation Conference (GECCO '99)*, pages 149–158, Orlando, Florida, 13-17 July 1999.

[16] J. Kim and P. J. Bentley. Evaluating negative selection in an artificial immune system for network intrusion detection. In *Genetic and Evolutionary Computation Conference (GECCO '01)*, 2001.

[17] J. R. Krebs and N. B. Davies. *An Introduction to Behavioural Ecology*. Sinauer Associates Inc., 1981.

[18] M. Ostaszewski, F. Seredynski, and P. Bouvry. Immune anomaly detection enhanced with evolutionary paradigms. In *Genetic And Evolutionary Computation Conference (GECCO '06)*, pages 119 – 126, Seattle, WA., US, 8-12 July 2006.

[19] M. Ostaszewski, F. Seredynski, and P. Bouvry. Coevolutionary-based mechanisms for network anomaly detection. *Journal of Mathematical Modelling and Algorithms*, 6:411–431, 2007.

[20] S. T. Powers and J. He. Evolving discrete-valued anomaly detectors for a network intrusion detection system using negative selection. In *The 6th Annual Workshop on Computational Intelligence (UKCI '06)*, pages 41–48, 2006.

[21] J. M. Shapiro, G. B. Lamont, and G. L. Peterson. An evolutionary algorithm to generate hyper-ellipsoid detectors for negative selection. In *Genetic and evolutionary computation Conference (GECCO '05)*, pages 337–344, Washington DC, USA, 2005.

[22] M. Toneguzzi. Theft, fraud cost retailers $8 million a day. Ottawa Citizen, March 2 2007. Newspaper.