LA-UR- 96-1020

CONF-960830--1

Title: CREDIT CARD FRAUD DETECTION: AN APPLICATION OF THE GENE EXPRESSION MESSY GENETIC ALGORITHM

Author(s): H. Kargupta
K. Buescher
J. Gattiker

Submitted to: Knowledge Discovery & Data Mining (KDD-96)
Portland, OR
August 2-4, 1996

Los Alamos
NATIONAL LABORATORY

Form No. 836 R5
ST 2629 10/91

Credit Card Fraud Detection: An Application Of The Gene Expression Messy Genetic Algorithm

Hillol Kargupta, James R. Gattiker, & Kevin Buescher

Computational Science Methods Group, X Division
P.O. Box 1663, XCM, Mail Stop F645
Los Alamos National Laboratory
NM 87545, USA.

email: hillol@lanl.gov

Abstract: *This paper describes an application of the recently introduced gene expression messy genetic algorithm (GEMGA) (Kargupta, 1996) for detecting fraudulent transactions of credit cards. It also explains the fundamental concepts underlying the GEMGA in the light of the SEARCH (Search Envisioned As Relation and Class Hierarchizing) (Kargupta, 1995) framework.*

Keywords: SEARCH, optimization, credit card fraud, genetic algorithms, GEMGA.

Credit Card Fraud Detection: An Application Of The Gene Expression Messy Genetic Algorithm

Hillol Kargupta*, James R. Gattiker & Kevin Buescher

Computational Science Methods Group, X Division
Los Alamos National Laboratory
Los Alamos, NM, USA.

Abstract

This paper describes an application of the recently introduced *gene expression messy genetic algorithm* (GEMGA) (Kargupta, 1996) for detecting fraudulent transactions of credit cards. It also explains the fundamental concepts underlying the GEMGA in the light of the SEARCH (Search Envisioned As Relation and Class Hierarchizing) (Kargupta, 1995) framework.

1 Introduction

Predicting credit card frauds requires a synergy of different technologies from different fields such as optimization, system identification, and machine learning. Search algorithms play an important role in all of them. In optimization a search algorithm is used for finding the optimal solution. In system identification a search algorithm may be used to find the optimal structure or optimal set of parameters of a model. In machine learning applications, a search algorithm may be used for inducing fraud detection rules. Clearly understanding the fundamental issues in search that makes an algorithm efficient is important.

The SEARCH (Search Envisioned As Relation and Class Hierarchizing) framework introduced elsewhere (Kargupta, 1995) offered a foundation of blackbox search (BBS)—search in presence of little domain knowledge—in terms of relations, classes, and partial ordering. SEARCH identified the class of order-k delineable problems—problems that can be solved using a bounded order of relations—that can be solved efficiently. SEARCH also offered an alternate perspective of evolution which lead to a biologically inspired evolutionary search technique called the *gene expression messy genetic algorithm* (GEMGA) (Kargupta, 1996). In this paper we describe an application of the GEMGA for credit card fraud detection.

Section 2 describes the problem and identifies the potential applications of search algorithms. In stead of applying an arbitrarily chosen algorithm we resort to understanding the fundamental issues in BBS following the SEARCH framework. Section 3 accomplishes that. Section 4 briefly describes the SEARCH perspective of evolutionary computation. Section 5 presents the GEMGA. Finally, Section 6 concludes this paper.

2 Credit Card Fraud Detection: A Brief Description

A single use of a credit card, that is, the transaction authorization procedure, represents a complex flow of information. In the most common scenario, the merchant sends card, purchase, and environment information to an acquiring (merchant acquiring) bank, this information is forwarded through a credit card network to the issuing (credit card issuing) bank, who then makes a decision regarding the authorization of the transaction. The response is transmitted back along

*The author can be reached at, P.O. Box 1663, XCM, Mail Stop F645, Los Alamos National Laboratory, Los Alamos, NM 87545, USA. e-mail: hillol@lanl.gov

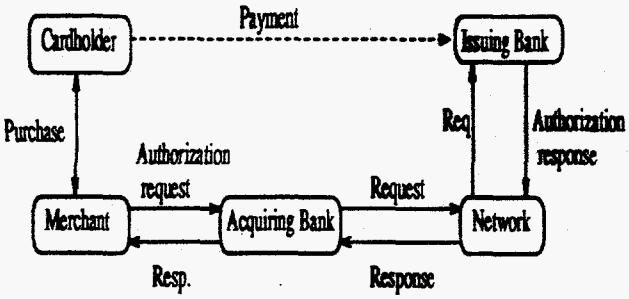

Figure 1:

these channels to the merchant. This general process is shown in Figure 1. The entire environment includes further aspects such as credit and funds transfer between the banks, and payment to the issuing bank by the cardholder. These aspects are irrelevant to the fraud scoring methodology, although they do impact qualitative aspects of the process implementation, e.g. methods of performance evaluation.

At some point in the authorization request to the issuer, the authorization is scored with a number correlated to probability of fraud. Credit industry fraud scoring remains inaccurate enough that the scores can only be considered as indicators of fraud rather than a conclusive classification. The computer-based scoring alone is not sufficient evidence to conclude whether a transaction is fraudulent, and it seems unlikely that it is possible to make such a system conclusive given the information available. Instead a fraud classification with some degree of certainty is treated as a requirement to follow up personally with the cardholder. Performance at significant detection levels is in the range of 10 to 100 times better than random performance, in terms of misclassification rates, can be achieved by computer based scoring. This level of performance can have a great positive impact in the overall operations of a card issuer. Fraud in the credit card industry is over $1 billion per year.

There are two common places in the authorization cycle to score transactions: at the issuer level, and at the network level. These two environments have distinct and complementary information. The issuer has detailed information about the cardholder, including credit history, purchase history and profiles, credit information, and account information, along with the information that is passed in an authorization request. The network does not have access to detailed information about the cardholder, but does see all transactions from all merchants and acquiring banks. Scoring at these two levels is thus a very different problem, utilizing complementary information. Of the two problems, issuer-based scoring parhaps has richer information. Our problem is network-based fraud detection.

There are several issues in the network level scoring that make this a challenging problem. For example, a large network may handle as many as 10 million authorization requests daily. This implies very large datasets to investigate, since a reasonable investigation may require data from weeks or even years. There are actually a large number of available raw variables at the network level, whose cardinality ranges from binary to 10K or more. The preprocessing and selection of input variables is a very broad and difficult problem. Finally, as with any problem of unknown complexity, the best choice of techniques for application to the problem is not clear at the outset.

Probably the most important aspect in successful scoring or classification, after having good information in the input variables, is how this information is presented to a modeling or optimization technique. Network-level scoring has the particular problem that most of the available information is non-numeric. In the problem under investigation, the input dataset has been reduced to a set of 86 normalized numeric features. For the purposes of this analysis, we are trying to find the best linear combination of these features. Optimum performance does not arise from simply minimizing the MSE to the output class for each example. The true evaluation criteria comes from assessing operational performance on a real dataset. Since little domain knowledge is available, searching for optimal model parameters from the noisy dataset offers a challenging task.

Since the fundamental problem is essentially a BBS, let us now focus on some theoretical issues in BBS, offered by the SEARCH framework. The idea is to avoid ad hoc trial of arbitrary search algorithms and replace the process by a

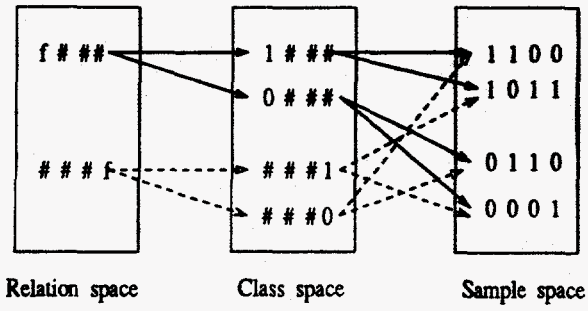

Figure 2: Decomposition of blackbox optimization in SEARCH.

well grounded theoretcially sound strategy. In the following section we briefly review some of the foundation concepts of the SEARCH framework.

3 SEARCH: A Brief Review

The foundation of SEARCH is laid on a decomposition of the blackbox search problem into relation, class, and sample spaces. A relation is a set of ordered pairs. For example, in a set of cubes, some white and some black, the color of the cubes defines a relation that divides the set of cubes into two subsets—set of white cubes and set of black cubes. Consider a 4-bit binary sequence. There are 2^4 such binary sequences. This set can be divided into two classes using the equivalence relation[1] $f###$, where f denotes position of equivalence; the # character matches with any binary value. This equivalence relation divides up the complete set into two equivalence classes, 1### and 0###. The class 1### contains all the sequences with 1 in the leftmost position and 0### contains those with a 0 in that position. The total number of classes defined by a relation is called its index. The order of a relation is the logarithm of its index with some chosen base. In a BBS problem, relations among the search space members are often introduced through different means, such as representation, operators, heuristics, and others. The above example of relations in binary sequence can be viewed as an example of relation in the sequence representation. In a sequence space of length ℓ, there are 2^ℓ different equivalence relations. The search operators also define a set of relations by introducing a notion of neighborhood. For a given member in the search space, the search operator define a set of members that can be reached by one or several application of the operators. This introduces relations among the members. Heuristics identifies a subset of the search space as more promising than others often based on some domain specific knowledge. Clearly this can be a source of relations. Relations can sometimes be introduced in a more direct manner. For example, Perttunen and Stuckman (1990) proposed a Bayesian optimization algorithm that divides the search space into Delaunay triangles. This classification directly imposes a certain relation among the members of the search space. The same goes for interval optimization (Ratschek & Voller, 1991), where the domain is divided into many intervals and knowledge about the problem is used to compute the likelihood of success in those intervals. As we see, relations are introduced by every search algorithm, either implicitly or explicitly. The role of relations in BBS is very fundamental and important.

Relations divide the search space into different classes and the objective of sampling based BBS is to detect those classes that are most likely to contain the optimal solutions. To do so requires constructing a partial ordering among the classes defined by a relation. The classes are evaluated using samples from the search domain and a *class comparison statistic* is used for comparing different classes. For a given *class comparison statistic* $\leq_\tau$ and some number M, a relation is said to *properly delineate* the search space if the class containing the optimal solution is within the top M classes, when the set of all classes defined by the relation are ordered using $\leq_\tau$. This basically means that if a relation satisfies the delineation constraint then, given sufficient samples, the relation will pick up the class containing the optimal solution within the top M ranked classes. If a relation does not satisfy this, then the relation leads to wrong decision and as a result success in finding the optimal solution is very unlikely.

A particular relation may not satisfy the delin-

[1] An equivalence relation is a relation that is reflexive, symmetric, and transitive.

eation constraint for different problems, different class comparison statistics, and different values of M. One relation may work for a particular case and may fail to do so for a different setting. Therefore, any algorithm that aspires to be applicable for a reasonably general class of problems, must search for appropriate relations. Determining whether or not a relation satisfies this delineation constraint requires decision making in absence of complete knowledge. For a given relation space Ψ_r, a BBS algorithm must identify the relations that properly delineate the search space with certain degree of reliability and accuracy. This requires comparing one relation with another using a *relation comparison statistic* and constructing a partial ordering among them.

A BBS algorithm in SEARCH cannot be efficient if it needs to consider relations that divide the search space in classes, with the total number of classes growing exponentially with the problem dimension. For example, in an ℓ-bit sequence representation, if there is a class of problem which requires considering the equivalence relations with $(\ell - 1)$ fixed bits then there is a major problem. This relation divides the search space into $2^{\ell-1}$ classes and we cannot solve this problem in complexity polynomial in ℓ. However, in BBS the ultimate objective is to identify the optimal solution which basically defines a singleton class. The smaller the cardinality of the individual classes, the larger the index of the corresponding relation. So we need the higher order relations for finally identifying the optimal solution, but we cannot directly evaluate them since their index is large. The solution is to limit our capability and realize that we can only solve those problems which can be addressed using low order relations and when high order relations are decomposable to those low order relations. This means that the information about low order relations can be used to evaluate the higher order relations. Consider the following example. Let r_0 be a relation that is logically equivalent to $r_1 \wedge r_2$, where r_1 and r_2 are two different relations; the sign $\wedge$ denotes logical AND operation. If either of r_1 or r_2 was earlier found to properly delineate the search space, then the information about the classes that are found to be bad earlier can be used to eliminate some classes in r_0 from further consideration. This process in SEARCH is called *resolution*. Resolution basically evaluates the relations of higher order using the information gathered by direct evaluation of bounded order relations.

The above description gives a brief informal overview of the SEARCH framework. As we saw, SEARCH addresses BBS on three distinct grounds: (1) relation space, (2) class space, and (3) sample space. Figure 2 shows this fundamental decomposition in SEARCH. The major components of SEARCH can be summarized as follows:

1. classification of the search space using a relation
2. sampling
3. evaluation, ordering, and selection of better classes
4. evaluation, ordering, and selection of better relations
5. resolution

A detailed description of each of these processes and their analysis, leading to the development of a bound on sample complexity, can be found elsewhere (Kargupta, 1995).

The SEARCH framework has clearly pointed out the different facets of decision making in BBS and explained why searching for relations is essential in BBS. This also identified the class of order-k delineable problems, that can be solved in polynomial sample complexity in SEARCH. An order-k delineable problem is one that can be solved using a polynomially bounded number of relations. The main lessons that will be used in the coming sections are, (1) search for appropriate relations is essential for transcending the limits of random enumeration, (2) both relation and class spaces require correct decision making, (3) we can only efficiently solve problems that need to consider a bounded number of relations from the given relation space, i.e. the class of order-k delineable problems, (4) the SEARCH perspective of *implicit parallelism* (Holland, 1975)—evaluation of different relations from the same sample set.

This sets the stage for launching an algorithm for solving the k-delineable problems. However, we shall take a detour and first establish the physical validity of the analytical findings in the context of a classical BBS algorithm of nature—the evolution of life on earth. In the following section we shall breifly examine the correspondence among the different components of SEARCH and the computational processes in natural evolution. This will later lead us to the development of the GEMGA—a BBS algorithm for order-k delineable problems motivated by the alternate perspective of evolutionary computation offered by SEARCH.

4 Evolutionary computation: The SEARCH perspective

Previous sections have clearly explained the need for understanding the processing of relations in blackbox search. In this section we take one step ahead by drawing a one to one correspondence between the evolutionary search mechanisms and decomposition of BBS in SEARCH.

- **Sample space**: DNA constitute the sample space. Crossover and mutation generate new samples of DNA. A population of organisms defines the sample space for the evolutionary search.

- **Class space**: Base sequences of mRNA transcribed in a cell correspond to only a part of the complete DNA. The sequence of amino acids in protein in turn correspond to base sequence in mRNA. The genetic code tells us that there is a unique relationship between the nucleotide triplets of the DNA and the amino acids in the protein. Therefore, if we consider the DNA as a representation defined over the evolutionary search space for life and different forms of life, then the amino acid sequence of a protein corresponds to a class of different DNA; every DNA in this class must have a certain sequence of nucleotides that can be transcribed to that particular sequence of amino acids. Since the genetic code is unique, a particular sequence of amino acids can only be produced by a certain sequence of nucleotides. In other words, the sequence of amino acids in a protein defines an equivalence class over the DNA space.

- **Relation space**: Recall that amino acid sequences in protein are translated from the nucleotide sequences of mRNA. The construction of mRNA is basically controlled by the transcription process. Since an equivalence relation is an entity that defines the equivalence classes, the transcription regulatory mechanism can be viewed as the relation space that defines classes in terms of the nucleotide sequences in mRNA and finally in terms of the amino acid sequences in proteins. Among the different components of this regulatory mechanism, regulatory proteins, promoter and terminator regions play a major role. Regulatory proteins exist as a separate entity from the DNA, but the promoter and terminator regions are defined on the DNA. It appears that there is a distinct relation space comprised of the different regulatory agents, such as activator and inhibitor proteins. However, it is quite interesting to note that this space also directly makes use of information from the sample space—the DNA. Expression of genetic information in eukaryotic organisms is more interesting than that in prokaryotes.

These possible relationships between the different spaces of SEARCH and natural evolution are summarized in Table 1.

Table 1: Counterparts of different components of SEARCH in natural evolution.

SEARCH	Natural evolution
Relation space	gene regulatory mechanism
Class space	amino acid sequence in protein
Sample space	DNA space

5 The Gene Expression Messy GA

The Gene Expression Messy GA (GEMGA), introduced elsewhere (Kargupta, 1996), is explicitly designed based on the lessons from the SEARCH framework, sketched in previous sections. The GEMGA is an $O(|\Lambda|^k(\ell+k))$ sample complexity algorithm for order-k delineable problems in sequence representation of length ℓ and alphabet Λ. In this section we describe the algorithm and related issues. Section 5.1 discusses the representation in GEMGA. Section 5.2 explains the population sizing in GEMGA. This is followed by Section 5.3 that describes the main operators, transcription, selection, and recombination. Section 5.4 presents of the overall mechanisms.

5.1 Representation

GEMGA uses a sequence representation. Each sequence is called a *chromosome*. Every member of this sequence is called a *gene*. A gene is a data structure, containing the *locus*, *value*, and *weight*. The *locus* determines the position of the member in the sequence. The locus does not necessarily have to be the same as the physical position of the gene in the chromosome. For example, the gene with locus i, may not be at the i-th position of the chromosome. When the chromosome is evaluated, however the gene with locus i gets the i-th slot. This positional independence in coding was introduced elsewhere (Deb, 1991; Goldberg, Korb, & Deb, 1989) to enforce the proper consideration for all relations defined by the representation. GEMGA does not depend on the particular sequence of coding. For a given ℓ bit representation, the genes can be placed in arbitrary sequence. A gene also contain the *value*, which determines the value of the gene, which could be any member of the alphabet set, Λ. The relation space is explicitly evaluated using the weights associated with each member. Weights take a positive real number except at the initial stage. All weights are initialized to -1.0. No two members with the same locus are allowed in the sequence. In other words, unlike the original messy GA (Deb, 1991; Goldberg, Korb, & Deb, 1989) no under or overspecifiction are allowed. A population in GEMGA is a collection of such chromosomes.

5.2 Population sizing

GEMGA requires at least one instance of the optimal order-k class in the population. For a sequence representation with alphabet Λ, a randomly generated population of size Λ^k is expected to contain one instance of an optimal order-k class. The population size in GEMGA is therefore, $n = c\Lambda^k$, where c is a constant. When the signal from the relation space is clear, a small value for c should be sufficient. However, if the relation comparison statistic produces a noisy signal, this constant should statistically take care the sampling noise from the classes defined by any order-k relation. Since GEMGA uses sequence representation, the relation space contains total 2^ℓ relations. However, GEMGA processes only those relations with order bounded by a constant, k. In practice, the order of delineability (Kargupta, 1995) is often unknown. Therefore, the choice of of population size in turn determines what order of relations will be processed. For a population size n, the order of relations processed by GEMGA is, $k = \log(n/c)/log|\Lambda|$. If the problem is order-k delineable (Kargupta, 1995) with respect to the chosen representation and class comparison statistics then GEMGA will solve the problem otherwise not. In that case a higher population size should be used to consider higher order relations.

5.3 Operators

GEMGA has four primary operators, namely: (1) *transcription*, (2) *class selection*, (3) *string selection*, and (4) *recombination*. Each of them is described in the following.

5.3.1 Transcription

As mentioned before, the weight space in GEMGA chromosomes is used to process relations. The transcription operator detects the appropriate order-k relations. Comparing relations require a relation comparison statistics. GEMGA

```
// k is the currently considered gene
Transcription(CHROMOSOME chrom,int k)
{
  double phi, delta, dwt;
  int dummy;

  dwt = chrom[k].Weight();
  if(dwt > 0.0 OR dwt == -1.0) {
    phi = chrom.Fitness();
    dummy = chrom[k].Value();
    // Change the value randomly
    chrom[k].PerturbValue();
    // Compute new fitness
    chrom[k].EvaluateFitness();
    // Compute the change in fitness
    delta = chrom[k].Fitness() - phi;
    // For minimization problem
    if(delta < 0.0)
      delta = 0.0;
    // Set the weight
    if(dwt < delta OR delta == 0.0)
      chrom[k].SetWeight(delta);
    // Set the value to the original value
    chrom[k].SetValue(dummy);
    // Set the original fitness
    chrom[k].SetFitness(phi);
  }
}
```

Figure 3: Transcription operator for minimization problem. For maximization problem, if `delta`< 0 absolute value of `delta` is taken and otherwise `delta` is set to 0.

does not process the relations in a centralized global fashion; instead it evaluates relations locally in a distributed manner. Every chromosome tries to determine whether or not it has an instance of a good class belonging to some relation. In GEMGA, the quality of a relation is determined by the quality of its good classes distributed over the population. Again, no centralized processing of relations is performed. The transcription operator is a deterministic one. It considers one gene at a time. The value of the gene is randomly flipped to note the change in fitness. For a *minimization problem*, if that change cause a improves the fitness (i.e. fitness decreases) then the original instance of the gene certainly do not belong to the instance of the best class of a relation, since fitness can be further improved. Transcription sets the corresponding weight of the gene to zero. On the other hand if the fitness worsens (i.e. fitness increases) then the original gene may belong to a good class; at least that observation does not say it otherwise. The corresponding weight of the gene is set to the absolute value of the change in fitness. Finally, the value of that gene is set to the original value and the fitness of the chromosome is set to the original fitness. In other words, ultimately transcription does not change anything in a chromosome except the weights. For a maximization problem the conditions for the weight change are just reversed. The same process is continued deterministically for all the ℓ genes in every chromosome of the population. Figure 3 shows the pseudocode for the transcription operator. For genes with higher cardinality alphabet set (Λ) this process is repeated for some constant $C < |\Lambda|$ times. The following section describes the two kinds of selection operators used in GEMGA, which correspond to the selective pressures in protein and DNA spaces of natural evolution.

5.3.2 Selection

Once the relations are identified, selection operator is applied to make more instances of better classes. GEMGA uses two kinds of selections—(1) class selection and (2) string selection. Each of them is described in the following:

- **Class Selection:** The class selection operator is responsible for selecting individual classes from the chromosomes. Better classes detected by the transcription operator are explicitly chosen and given more copies at the expense of bad classes in other chromosomes. Two chromosomes are randomly picked; the weights of the genes are compared and the gene with higher weight overwrites the corresponding gene in other chromosome with lower weight.
- **String Selection:** This selection operator gives more copies of the chromosomes. A

standard binary tournament selection operator (Brindle, 1981; Goldberg, Korb, & Deb, 1989) is used. Binary tournament selection randomly picks up two chromosomes from the population, compares their objective function values, and gives one additional copy of the winner to the population at the expense of the looser chromosome.

The following section describes the recombination operator in GEMGA.

5.3.3 Recombination

Recombination operator in GEMGA works as follows. It randomly picks up two chromosomes from the population and considers all the genes in the chromosomes for possible swapping. It randomly marks one among them. If the weight of a gene from the marked chromosome is greater than that of the corresponding gene from the other chromosome then it swaps the genes.

The following section describes the overall mechanism of the algorithm.

5.4 The Algorithm

GEMGA has two distinct phases: (1) primordial stage and (2) juxtapositional stage. The primordial stage simply applies transcription operator for ℓ generations, deterministically considering every gene in each generation. During this stage the population of chromosomes remains unchanged, except that the weights of the genes change. This is followed by the juxtapositional stage, in which the selection and recombination operators are applied iteratively. Figure 4 shows the overall algorithm. The length of the juxtapositional stage can be roughly estimated as follows. If t be the total number of generations in juxtapositional stage, then for binary tournament selection, every chromosome of the population will converge to same instance of classes when $2^t = n$, i.e. $t = \log n / \log 2$. Substituting $n = c|\Lambda|^k$, we get, $t = \frac{\log c + k \log |\Lambda|}{\log 2}$. A constant factor of t is recommended for actual practice. Clearly the number of generations in juxtapositional stage is $O(k)$. Let us now compute the overall sample complexity of GEMGA. Since the population size is $O(|\Lambda|^k)$ and the primordial stage continues for $C\ell = O(\ell)$ generations, the overall sample complexity,

$$SC = O(|\Lambda|^k(\ell + k))$$

GEMGA is a direct realization of the lessons from the SEARCH framework. Following SEARCH, it can be recognized that the sample complexity is also a function of the desired quality of the solution and the reliability of the process. However, the implementation of GEMGA through distributed local evaluation of relations and classes outweighs the satisfaction of quantifying the success probability that is straight forward in case of centralized comparison (as it was in SEARCH) from the practical perspective. Therefore, the reader must realize the dependence of the sample complexity on the desired accuracy of the solution and reliability, implicit in the above arguments. The following section concludes this work.

6 Conclusion

This paper reported an application of the GEMGA for credit card fraud detection. As noted earlier search algorithms play a universal role in almost every aspect of data mining. Therefore, the GEMGA awaits many other possible applications. If the problem is order-k delineable with respect to the representation and class comparison statistic GEMGA will solve the problem in polynomial sample complexity.

7 Acknowledgement

This work has been supported by the US. Department of Energy. The author also acknowledges many useful discussions with Professor David E. Goldberg.

References

Brindle, A. (1981). *Genetic algorithms for function optimization.* Unpublished doctoral dissertation, University of Alberta, Edmonton, Canada.

Deb, K. (1991). *Binary and floating-point function optimization using messy genetic algorithms* (IlliGAL Report No. 91004). Urbana: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3*(5), 493–530. (Also TCGA Report 89003).

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

Kargupta, H. (1995, October). *SEARCH, Polynomial Complexity, and The Fast Messy Genetic Algorithm.* Doctoral dissertation, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. Also available as IlliGAL Report 95008.

Kargupta, H. (1996, January). *SEARCH, evolution, and the gene expression messy genetic algorithm.* Los Alamos Unclassified Report LA-UR-96-60.

Perttunen, C., & Stuckman, B. (1990). The rank transformation applied to a multi-univariate method of global optimization. *IEEE Transactions on System, Man, and Cybernetics*, *20*, 1216–1220.

Ratschek, H., & Voller, R. L. (1991). What can interval analysis do for global optimization? *Journal of Global Optimization*, *1*, 111–130.

```
void GEMGA() {
POPULATION Pop;
int i, j, k, C, k_max;

// Initialize the population at random
Initialize(Pop);
i = 0;
// Primordial stage
While(i < C) { // C is a constant
  j = 0;
  Repeat {
    // Identify better relations
    Transcription(Pop, j);
    // Increment generation counter
    j = j + 1;
  } Until(j == Problem_length)
  i = i + 1;
}
k = 0;
// Juxtapositional stage
  Repeat {
    // Select better strings
    Selection(Pop);
    // Select better classes
    ClassSelection(Pop);
    // Produce offspring
    Recombination(Pop);
    Evaluate(Pop); // Evaluate fitness
    // Increment generation counter
    k = k + 1;
  // k_max is of O(log(Problem_length))
  } Until ( k > k_max )
}
```

Figure 4: Pseudo-code of GEMGA. The constant C ¡ |Λ|, where |Λ| is the cardinality of the alphabet set.