LA-UR- 96-2748

Title: EXTENDING THE CLASS OF ORDER-K DELINEABLE PROBLEMS FOR THE GENE EXPRESSION MESSY GENETIC ALGORITHM

CONF-960820--5

Author(s): H. Kargupta, D. E. Goldbert, L. Wang

Submitted to: Foundations of Genetic Algorithms(FOGA4)
San Diego, CA, August 1996

Los Alamos
NATIONAL LABORATORY

LA-UR-96- 2748

Extending the Class of Order-*k* Delineable Problems For the Gene Expression Messy Genetic Algorithm

Hillol Kargupta
Computational Science Methods Group
Los Alamos National Laboratory
Los Alamos, NM 87545

David E. Goldberg & Lewei Wang
Department of General engineering
University of Illinois at Urbana-Champaign
Urbana, IL

Extending The Class of Order-k Delineable Problems For The Gene Expression Messy Genetic Algorithm

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

David E. Goldberg & Lewei Wang
Department of General Engineering
University of Illinois at Urbana-Champaign
Urbana, IL, USA.

Abstract

This paper revisits the recently introduced *gene expression messy genetic algorithm* (GEMGA) (Kargupta, 1996) and offers some modifications to extend the class of order-k delineable problems (class of problems that can be solved using a bounded order of relations) in GEMGA. The fundamental components that control the delineability of relations are reviewed in the light of the recently proposed SEARCH framework (Kargupta, 1995). Modified class and relation comparison statistics of GEMGA are proposed. The sample complexity of this improved version of GEMGA is shown to be subquadratic. Theoretical conclusions are also substantiated by experimental results for large, multimodal order-k delineable problems with respect to class average comparison statistic. We also present results for the recently constructed Goldberg-Lewei test functions.

1 Introduction

The gene expression messy genetic algorithm (GEMGA)—a new class of messy GAs was introduced in the recent past (Kargupta, 1996) following the alternative perspective of natural evolution offered by the SEARCH (Search Envisioned As Relation and Class Hierarchizing) framework developed elsewhere (Kargupta, 1995). SEARCH offered a foundation of blackbox optimization (BBO) in terms of relations, classes, and partial ordering.

The GEMGA embodied the lessons from SEARCH in a distributed manner by using the natural motivation of gene expression in evolution. The sample complexity of this algorithm can be proven to be subquadratic with the problem size for the class of order-k delineable problems, defined with respect to the class comparison statistic used in GEMGA.

The main objective of this paper is to extend the class of order-k delineable problems for the GEMGA by eliminating the interference among the class and relation comparison statistics. Unlike the previous version of GEMGA, the proposed scheme explicitly deals with separate relation, class, and sample spaces. We modify the *transcription operator* of GEMGA to eliminate the interference and show the overall sample complexity of the process still remains subquadratic. However, the population size still remains independent of problem length. This paper also presents experimental results for this modified version of GEMGA.

Section 2 briefly reviews the SEARCH framework. Section 3 revisits the issue of order-k delineability that plays an important role in GEMGA. Section 4 presents a description of the GEMGA and points out the interference among the class and relation comparison statistics in the design of GEMGA. Section 5 describes the proposed modifications. This is followed by Section 6 which presents the test results for large multimodal, order-k delineable problems. Finally, Section 7 concludes this paper.

2 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

*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

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

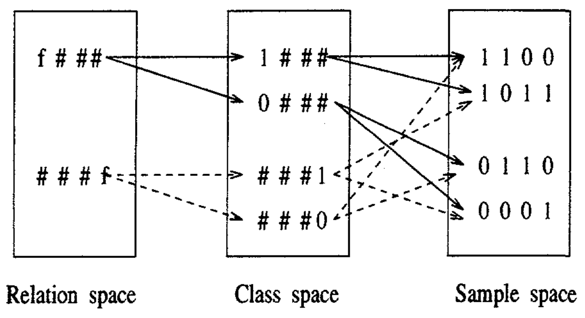

Figure 1: Decomposition of blackbox optimization in SEARCH.

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 BBO 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 BBO is very fundamental and important.

Relations divide the search space into different classes and the objective of sampling based BBO 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* $\leq_{\mathcal{T}}$ 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 $\leq_{\mathcal{T}}$. 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 delineation 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 BBO 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 BBO 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 BBO 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 BBO on three distinct grounds: (1) relation space, (2) class space, and (3) sample space. Figure 1 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 BBO and explained why searching for relations is essential in BBO. 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.

Before we conclude this review on SEARCH, let us revisit the issue of order-k delineability in order to clear up our current objectives and future directions for designing BBO algorithms.

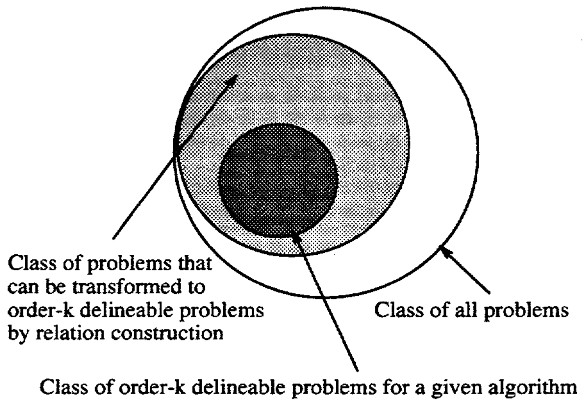
Class of problems that can be transformed to order-k delineable problems by relation construction
Class of all problems
Class of order-k delineable problems for a given algorithm

Figure 2: BBO problems from the delineability perspective.

3 Implications Of Order-k delineability

As we saw in the previous section, the notion of order-k delineability presents a picture of the general class of BBO problems from the perspective of an algorithm. In SEARCH, defining a BBO algorithm requires specifying the relation space, class comparison statistic, and the constant M that defines how many "top" classes will be picked up. Therefore, by definition an algorithm in SEARCH specifies the class of order-k delineable problems. For a chosen class comparison statistic and M, the relation space restricts the class of order-k delineable problems for an algorithm. Changing the relation space by constructing new relations may convert a non-order-k delineable problem to an order-k delineable one. For some problems finding such transformation by constructing new relations may be possible in sample complexity, polynomial in problem size, reliability, and accuracy of the solution. Clearly, there may exist a class of non-order-k delineable problems, that can be transformed to order-k delineable problems in polynomial sample complexity. Figure 2 shows a schematic description of this classification of BBO problems.

It is important to note that, membership of a problem in the class of order-k delineable problems does not necessarily guarantee that the algorithm will solve that problem. It only says that the problem is " efficiently solvable" in the chosen relation space, class comparison statistic, and M. The algorithm needs to perform adequate sampling and make decisions with high confidence in the relation and class spaces in order to find the desired quality solution. Therefore, the first step of an algorithm should be to make sure it can solve its own order-k delineable class of problems. That will define the first milestone. The next step should be to introduce mechanism for new relation construction and investigate what kind of problems can be dynamically transformed to order-k delineable problems efficiently. Unfortunately, there hardly exists any algorithm that can adequately guarantee the capability of solving even its order-k delineable problems. The goal of this paper is to take this elementary step. The proposed version of gene expression messy GA does not construct new relation space; rather it guarantees the capability of solving its own order-k delineable problems efficiently.

This sets the stage for launching the GEMGA for solving its k-delineable problems. The following section presents the computational and biological motivations behind the GEMGA. The main features of the earlier version of GEMGA are described and the need for explicit processing of relations, classes, and samples are noted.

4 The GEMGA: Why, What, And All That

In this section, we shall first present the motivation behind the GEMGA. This will be followed by a qualitative description of previous version of the GEMGA. Finally, we identify some features of the GEMGA that restricts its class of order-k delineability problems and suggest some remedies.

4.1 Why GEMGA?

The current version of GEMGA is designed to solve its own class of order-k delineable problems with high reliability and solution accuracy efficiently. Although the next step will be to explore the possibility of introducing relation construction for extending the class of order-k delineable problems, the proposed version does not make any effort to do that. The computational and biological motivations behind the GEMGA are presented below.

- **The Computational perspective:** Despite the distinct need of searching for relations in order to transcend the limits of enumerative search, there exists hardly any blackbox optimization algorithm that realizes this. Moreover, the decision makings in the relation and class spaces often interfere with each other in common search algorithms since the spaces are not distinctly defined. For example both simulated annealing, genetic algorithms use the sample space for evaluating both relations and classes. The GEMGA transcends these bottlenecks. It explicitly defines the relation, class, and the sample spaces. It precisely defines the relation and class comparison statistics. It also provides a mechanism for controlling the constant M. Moreover, the GEMGA is highly suitable of distributed parallel implementation.
- **The biological perspective:**
The GEMGA also has a strong biological foundation, which is based on a mapping from the relation, class, and sample spaces to different components of natural evolutionary search space. Table 1 summarizes this correspondence. It also hypothesized a perspective of the search for appropriate relations in evolution through *gene expression.*
Unfortunately, many of the existing computational models of evolution address only the extracellular flow of genetic information. Existing models of evolutionary computation like genetic algorithms (Holland, 1975), evolutionary strategie (Rechenberg, 1973), and evolutionary algorithms (Fogel, Owens, & Walsh, 1966) are some examples. These existing perspectives of evolutionary computation do not assign any computational role to the nonlinear mechanism for transforming the information in DNA into proteins. The same DNA is used for different kinds of proteins in different cells of living beings. The development of different expression control mechanisms and their evolutionary objectives are hardly addressed in these models. They primarily emphasize the extra-cellular flow. The main difference among these models seems to be the emphasis on crossover compared to mutation or vice versa. The GEMGA emphasizes the computational role of gene expression in evolution. It makes use of transcription like operators for detecting relations and classes.

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

4.2 Earlier version of the GEMGA

The gene expression messy GA (GEMGA) was introduced elsewhere (Kargupta, 1996) following the need for a BBO algorithm that properly searches for relations and pays careful attention to the fundamental components of BBO. The main characteristic of the GEMGA is the decomposition of the search space in

Table 2: Components of the *gene* data structure in GEMGA.

Gene components	Purpose
locus	position in sequence space
value	sample space
weight	class space
linkage set	relation space

relation, class, and sample spaces. The structure of the *gene* in the GEMGA played an important role in the implementation of this decomposition. The earlier proposed version of GEMGA used genes that defined,

- relation space in terms of weights associated with genes;
- class and sample space together in terms of values and loci.

Although the earlier version of GEMGA performed well for different problems, the class of order-k delineable problems for the GEMGA can be further extended by completely separating the relation, class, and sample spaces from each other and modifying the relation and class comparison statistics. The recently proposed Goldberg-Lewei class of test functions, which will be described later in this paper, demonstrated that the such modifications are essential for making the GEMGA more widely applicable. The following section describes the proposed version of the GEMGA.

5 The GEMGA

This section introduces a modified version of the GEMGA and shows that the overall sample complexity is subquadratic. 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

The 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; As in the previous version of the GEMGA, it contains the *locus*, *value*, and *weight*. However, in addition to that every gene also contains a dynamic list of integers, called the *linkage set*. 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. The 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 contains the *value*, which determines the value of the gene, which could be any member of the alphabet set, Λ. The weights associated with every gene take a positive real number except at the initial stage. All weights are initialized to -1.0. The weight space over all the genes define the class space of the GEMGA. The linkage set of a gene is a list of integers defining the set of genes related with it. If the genes with loci, $\{1, 5, 10, 15\}$ are related to each other then the gene with locus 1 will have the linkage set $\{5, 10, 15\}$. Similarly, the gene with locus 5 will have the linkage set $\{1, 10, 15\}$. The *linkage set* space over all genes defines the relation space of the GEMGA. Table 2 illustrates different components of a gene in the GEMGA. No two genes 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 overspecifictions are allowed. A population in GEMGA is a collection of such chromosomes.

5.2 Population sizing

The 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 the proposed version of 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

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

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

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

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 of the proposed version of the GEMGA chromosomes represents the class space. On the other hand the relation space is defined by the linkage set associated with every gene. The transcription operator detects the appropriate order-k relations. The transcription phase I operator determines the instances of genes contributing to the locally optimal classes. The transcription phase II operator determines the clusters of genes precisely defining the relations among those instances of genes. Comparing relations requires a relation comparison statistics. The GEMGA does not process the relations in a centralized 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. The transcription phase I operator considers one gene at a time. The value of the gene is randomly flipped to note the change in fitness. For a *minimization prob-*

```
// pick1, pick2 are the indices of a pair of genes
TranscriptionPhaseII(CHROMOSOME chrom,
    int pick1, int pick2)
{
  double phi, delta;
  int dummy1, dummy2;

  if(chrom[pick1].Weight() > 0) {
    dummy1 = chrom[pick1].Value();
    phi = chrom.Fitness();
    chrom[pick1].PerturbValue();
    chrom.EvaluateFitness();
    if(chrom[pick2].Weight() > 0.0) {
      chrom[pick2].PerturbValue();
      dummy2 = chrom[pick2].Value();
      chrom[pick2].PerturbValue();
      chrom.EvaluateFitness();
      delta = chrom.Fitness() - phi;
      // For minimization problem
      if(delta < 0.0)
        delta = 0.0;
      if(delta == chrom[pick2].Weight()) {
        chrom[pick1].AddLinkageSet(pick2);
        chrom[pick2].AddLinkageSet(pick1);
      }
      chrom[pick2].SetValue(dummy2);
    }
    chrom[pick1].SetWeight(1.0);
    // Set the value to the original value
    chrom[pick1].SetValue(dummy1);
    // Set the original fitness
    chrom.SetFitness(phi);
  }
}
```

Figure 4: Transcription Phase II operator for minimization problem.

lem, if that change cause an improvement in 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 phase I 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 Transcripton phase I operator. Transcription phase II identifies the exact relations among the genes and constructs the linkage set of every gene in a choromosme. This operator performs pairwise consideration of genes. The objective is to identify the set of genes that are related with any given gene from the chromosome. Among the $\binom{\ell}{2}$ possible pair of choices only those pairs are considered in which both the genes have non-zero weights. In other words if a gene is identified as a possible contributor to an instance of locally optimal set of genes then its dependencies on other such genes in that chromosome are tested using the transcription phase II operator. For every gene with non-zero weight the linkage set is constructed and the real weights are replaced by boolean weights. Figure 4 shows the pseudo-code for this operator, where `pick1` and `pick2` define the loci of the pair of genes.

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.

```
ClassSelection(chrom1, chrom2)
CHROMOSOME chrom1, chrom2;
{
int i;

for(i=0; i<Problem_length; i++) {
  if(Rnd()<0.5 AND chrom1[i].Weight()>0) {
    if(chrom1[i].LinkageSet.Length() >
      chrom2[i].LinkageSet.Length()) {
      // Collect linkage sets of chosen genes
      SelectSet.Collect[LinkageSet[i]]; }
  }
}
for(i=0; i<SelectSet.Length(); i++)
  chrom2[SelectSet[i]]=chrom1[SelectSet[i]];
}
```

Figure 5: Class selection operator in GEMGA. A consistent coding (where *chrom*1[*i*] and *chrom*2[*i*] has common *locus*) is used in place of messy coding for the sake of illustration. `Rnd()` generates a random number in between 0 and 1.

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. Figure 5 describes the operator. Two chromosomes are randomly picked; A set of genes with non-zero weights are chosen from one of them, `chrom1`; those genes with cardinality of their `LinkageSet` strictly greater than those of their counterparts in the other participating chromosome are collected in a list called `SelectSet`. Then the genes of the chromosome `chrom1` corresponding to `SelectSet` are copied on the corresponding genes of `chrom2`.
- **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

Figure 6 shows the mechanism of the recombination operator in GEMGA. 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. Just like the `ClassSelection` operator `Recombination` selects a set of genes called the `ExchangeSet`. Genes of

```
Recombination(chrom1, chrom2)
CHROMOSOME chrom1, chrom2;
{
int i;
GENE dummy;

for(i=0; i<Problem_length; i++) {
  if(Rnd()<0.5 AND chrom1[i].Weight()>0) {
    if(chrom1[i].LinkageSet.Length() >
      chrom2[i].LinkageSet.Length()) {
      // Collect linkage sets of chosen genes
      ExchangeSet.Collect[LinkageSet[i]];
    }
  }
}
for(i=0; i<ExchangeSet.Length(); i++) {
  dummy=chrom1[ExchangeSet[i]];
  chrom1[ExchangeSet[i]]=
      chrom2[ExchangeSet[i]];
  chrom2[ExchangeSet[i]]=dummy;
}
}
```

Figure 6: Recombination operator in GEMGA. A consistent coding (where `chrom1[i]` and `chrom2[i]` has common *locus*) is used in place of messy coding for the sake of illustration. `Rnd()` generates a random number in between 0 and 1.

`chrom1` and `chrom2` corresponding to the members of `ExchangeSet` are exchanged.

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 first applies the transcription phase I operator for ℓ generations, deterministically considering every gene in each generation. This is followed by the application of the transcription phase II operator for each pair of genes with non-zero weights. During this stage the population of chromosomes remains unchanged, except that the weights of the genes change and the linkage sets get constructed. This is followed by the juxtapositional stage, in which the string selection, class selection, and recombination operators are applied iteratively. Figure 7 shows the overall algorithm. The length of the transcription phase I application is ℓ. The length of the application of the transcription phase II application is $\ell^2 - \ell$ in the worst case. The length of the juxtapositional stage can be roughly estimated as follows. If t be the total number of gener-

```
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;
}
TranscriptionPhaseII(Pop);
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 7: Pseudo-code of GEMGA. The constant C ¡ $|\Lambda|$, where $|\Lambda|$ is the cardinality of the alphabet set.

ations 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,

$$\begin{aligned} SC &= O(|\Lambda|^k(\ell + \ell^2 - \ell + k)) \\ &= O(|\Lambda|^k(\ell^2 + k)) \end{aligned}$$

Note that the transcription phase II operator is applied on those pair of genes that have non-zero weights. Therefore, the complexity of this operation is quadratic only in the worst case when all the genes in a chromosome have non-zero weights.

GEMGA is a direct realization of the lessons from the SEARCH framework. Now that we described the algorithm let us take a pause and see how the different components of GEMGA maps onto the SEARCH.

- **Relation, class, and sample spaces:** The linkage set defines the relation space; the weights define the class space and the (value, locus) pair define the sample space.
- **Class comparison statistics:** The class comparison statistic in GEMGA is defined in two stages. First of all, the transcription phase I operator identifies the locally optimal set of genes by bitwise perturbation. Once the transcription phase II operation is performed to identify the relations among these set of genes, all the genes defining locally optimal classes are assigned a non-zero weight 1. This makes sure that the classes are not given any undeserved bias based on their local evaluation. The earlier version of GEMGA had this undue bias. The local evaluation was used to compare classes at a global level.
- **Relation comparison statistics:** Transcription phase II identifies the linkage set of the locally optimal set of genes. The relation and class comparison statistics are mutually dependent in GEMGA.
- **Constant M:** The value of M is controlled in a distributed manner in GEMGA. The class selection operator controls the value of M in the gemga. As we increase the class selection probability the M value is decreased and the vice versa.
- **Recombination:** This implements the resolution operation of SEARCH.

The following section presents the test results.

6 Test Results

Designing a test set up requires careful consideration. An ideal set up should contain problems with different dimensions of problem difficulty, such as multi-modality, bounded inappropriateness of relation space (BIRS), problem size, scaling, noise. The GEMGA has been tested against problems with all of these dimensions of difficulties (Kargupta, 1996). However, because of limited space, in this section, we present the performance of GEMGA for problems with only massive multimodality and controlled amount of BIRS. For all functions the average number of function evaluations per success (AFPS) is measured. For every function we choose the desired solution value (DSV) a priori. We say the algorithm was successful if it reaches the DSV.

Recombination probability	1.0
Class selection probability	1.0
String selection probability	1.0

Table 3: GEMGA parameters.

6.1 Problems with BIRS and massive multimodality

Deceptive trap functions (Ackley, 1987) are used as basic building blocks for designing this test suite. A trap function can be defined as follows:

$$\begin{aligned} f(x) &= \ell' \text{ if } u = \ell' \\ &= \ell' - 1 - u \text{ otherwise,} \end{aligned}$$

where u is the number of 1-s in the string x and ℓ' is the length of the sequence used for representing the variable x. Goldberg, Deb, and Clark (1992) showed that such deceptive problems can be used to design problems of bounded difficulty. In a trap function defined over a sequence of length ℓ' the order of delineability is ℓ' with respect to the class average comparison statistics. Although GEMGA does not work using the class average comparison statistic (i.e. when classes are compared with respect to the distribution means) this gives us a simple way to capture the main essence. When multiple number of such functions are concatenated with each other a problem defined over a sequence of length ℓ with order-ℓ' delineability can be designed. Since the order of delineability directly controls the BIRS, such concatenated functions can be effectively used for designing problems with BIRS by controlling the ℓ'. Such functions have only ℓ/ℓ' proper relations among the $\binom{\ell}{\ell'}$ order-5 relations that must be detected in order to find the global solution. Therefore, searching for the appropriate relations is not a trivial job in this class of problems. Apart from BIRS, such functions also offer multimodality. If we carefully observe, we shall note that a trap function has two peaks. One of them corresponds to the string with all 1-s and the other is the string with all 0-s. If we design a problem by concatenating m such functions, it will have a total of 2^m local optima and among them only one will be the globally optimal solution.Clearly this class of problems are massively multimodal and has bounded inappropriateness of the relation space, defined by the representation.

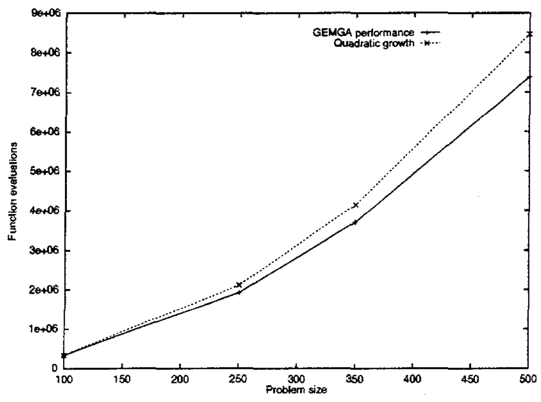

Figure 8: Problems with BIRS and massive multimodality: Growth of the number of function evaluations with problem size. The population size for all problem sizes was 300. Note that the sample complexity is subquadratic.

For testing the GEMGA, a test function is constructed by concatenating multiple numbers of trap functions, each with $\ell' = 5$. Therefore the order of delineability is five. As we increase the number of functions, *in other words the overall problem length* ℓ, the degree of BIRS remains constant, but the degree of multimodality increases exponentially. For $\ell = 200$, the overall function contains 40 subfunctions; therefore, an order-5 bounded 200-bit problem has 2^{40} local optima, and among them, only one is globally optimal.

The GEMGA is tested against order-5 deceptive problems of different sizes. Table 3 shows the GEMGA parameters used for all of them. Figure 8 shows the average number of function evaluations from five independent runs needed to find the DSV for different problem sizes. For all problems, the DSV is set to the globally optimal solution, which is equal to problem size, ℓ. The population size is chosen as described earlier in this paper. The chosen population size for all the problems was 300. The sample complexity clearly grows linearly and the population size is constant.

6.2 Goldberg-Lewei test functions

Following the development of the GEMGA, Goldberg and Lewei proposed (personal communication) two so called weight misleading test functions that apparently mislead the original version of GEMGA. Let us first define those two functions.

• **GL1:** This function is comprised of order 5 subfunctions. Each of the order 5 subfunctions are defined as follows:

$$\phi(1\#\#\#1) = 4$$

```
φ1(x, v1, v2) {
int i=0;
while(x[i]==0) i++;
if((i%2) == 1)) return v1;
else return v2;
}
```

Figure 9:

$$\begin{aligned}\phi(1\#\#\#0) &= 8\\ \phi(0\#\#\#1) &= 10\\ \text{Otherwise} &= 0\end{aligned}$$

Character # represents don't care values, in other words the objective function value does depend on those values. Such order 5 subfunctions are concatenated one after another for constructing problems of larger size. The second function is defined as follows:
• **GL2:** This is a function of unitation, i.e. the number of 1's in the string. Let us denote the unitation variable by u and subfunction size by k. A subfunction of size k can be defined as follows:

$$\begin{aligned}\phi(x) &= 10 \quad \text{if} \quad u = 0;\\ &= \phi_1(x, 7, 2) \quad \text{if} \quad u = 1;\\ &= \phi_1(x, 4, 3) \quad \text{if} \quad u = k-1;\\ &= 8 \quad \text{if} \quad u = k;\\ &= 0 \quad \text{Otherwise}\end{aligned}$$

where $\phi_1(x)$ is defined in Table 9.

The class comparison statistic in the original version of GEMGA had undue reliance on the local evaluation of the classes. In that version, the goodness of the class was estimated by the change in objective function value caused by local perturbation. If the transcription phase I operator produces a positive weight for a gene, then we know that the gene belong to a locally optimal class. The absolute value of the fluctuation in objective function value does not necessarily represent the relative goodness of different classes. Moreover, the linkage set used to be implicitly determined by relative similarity in weights among different genes. These problems may lead to fooling GEMGA for certain class of problems. This is exactly what Goldberg and Lewei showed. Let us now follow the construction of their test functions.

Consider function GL1. Table 4 shows the weights of the strings corresponding to one order 5 subfunction and the objective function values. Irrelevant bits are represented by # character. Clearly strings with 0 in the leftmost and 1 in the rightmost position (0###1) are the globally optimal solutions. However, not all the weights of these strings are greater

	Strings	Weights	Φ
GL1	0###1	6 0 0 0 10	10
	1###0	8 0 0 0 4	8
	1###1	0 0 0 0 0	4
	0###0	0 0 0 0 0	0
GL2	0000000000	3 8 3 8 3 8 3 8 3 8	10
	1111111111	4 5 4 5 4 5 4 5 4 5	8

Table 4: Weights after primordial stage for GL1 and GL2.

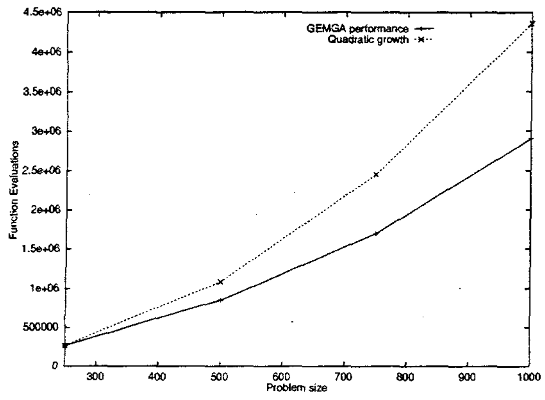

Figure 10: Goldberg-Lewei test function GL1: Growth of the number of function evaluations with problem size. The population size for all problem sizes was 400. Note that the sample complexity of the GEMGA is subquadratic.

than the counterparts in other strings. For example, strings 1###0 will have the highest weight in the leftmost position. As a result repeated application of weight based class selection and recombination will produce the strings 1###1, which are suboptimal.

Function GL2 shares the same philosophy. Table 4 shows the weights and objective function values of the top two kinds of strings. Again as we see the weights in the odd positions of the best solution are less than the corresponding weights of the second best solution. Repeated application of weight based class selection and recombination will produce the suboptimal solution, 1010101010.

The original version of GEMGA failed to solve these problems because of the reasons noted earlier in this section. However, the currently proposed version of GEMGA eliminated those shortcomings. This version of GEMGA is tested on GL1 up to the problem size of 1000. Figure 10 shows the growth of the number of function evaluations along problem size. The same set of parameters as given in Table 3 is used. We also applied GEMGA on a 250-bit version of GL2. A population size of 5000 is used, since order of delineability in that instance of GL2 is 10; $2^{10} = 1024$; A constant factor of 5 is used to make sure the presence of all order-10 instances of good classes. The GEMGA solved this problem with a total number of function evaluations $1.34168e + 06$.

The following section concludes this paper.

7 Conclusion

This paper revisited the gene expression messy genetic algorithm. It proposed a further decomposed processing of relations, classes, and samples. The undue reliance on local evaluation in ordering the classes is removed. The transcription phase II operator is introduced for explicitly detecting the relations. We also showed that the worst case complexity of the algorithm is quadratic. Experimental results demonstrated that the GEMGA can solve large multimodal problems with bounded inappropriateness of representation.

The GEMGA eliminates many problems of the previous versions of messy GAs. The main improvements are (1) explicit processing of relations and classes, (2) eliminating the need for a template solution, (3) reducing the population size from $O(|\Lambda|^k \ell)$ to $O(|\Lambda|^k)$ for order-k delineable problems in sequence representation of length ℓ, (4) eliminating the thresholding scheduling problem of the fmGA (Goldberg, Deb, Kargupta, & Harik, 1993), and (4) reducing the running time by a large factor.

As we noted earlier, our immediate goal is to design an algorithm that can effectively solve its own order-k delineable class of problems. This paper brought us little closer to that goal. So far, we have only used those relations that are defined by the chosen representation. The next step will be to introduce new relation construction for converting non-order-k delineable problems to order-k delineable problems.

8 Acknowledgement

The first author acknowledges the support from US. Department of Energy, AFSOR Grant F49620-94-1-0103.

References

Ackley, D. H. (1987). *A connectionist machine for genetic hill climbing.* Boston: Kluwer Academic.

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.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution.* New York: John Wiley.

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, *6*, 333–362.

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimizaiton of difficult problems using fast messy genetic algorithms. In Forrest, S. (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms* (pp. 56–64). San Mateo, CA: Morgan Kaufmann.

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.

Rechenberg, I. (1973). Bionik, evolution und optimierung. *Naturwissenschaftliche Rundschau*, *26*, 465–472.