LA-UR- 96-2412

CONF-9607153--2

Title: MESSY GENETIC ALGORITHMS: RECENT DEVELOPMENTS

Author(s): Hillol Kargupta

Submitted to: NATO ASI Meeting
Gran de Canaria de Spania
July 1-5, 1996

Los Alamos
NATIONAL LABORATORY

DISCLAIMER

Messy Genetic Algorithms: Recent Developments

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

Abstract

Messy genetic algorithms define a rare class of algorithms that realize the need for detecting appropriate relations among members of the search domain in optimization. This paper reviews earlier works in messy genetic algorithms and describes some recent developments. It also describes the gene expression messy GA (GEMGA)—an $O(\Lambda^k(\ell^2+k))$ sample complexity algorithm for the class of order-k delineable problems (Kargupta, 1995) (problems that can be solved by considering no higher than order-k relations) of size ℓ and alphabet size Λ. Experimental results are presented to demonstrate the scalability of the GEMGA.

1 Introduction

One of the most important task of a "general purpose" blackbox optimization (BBO) algorithm is to search for appropriate relations among the members of the search space. In genetic algorithms (GAs) the representation is the main source of relations. Unfortunately, like many other BBO algorithms GAs do not properly search for relations. Messy genetic algorithms (mGAs) (Goldberg, Korb, & Deb, 1989) realize the need for adequate search for relations and make an effort to do that.

The main objective of this paper is to present a brief review of the earlier works on messy GAs, describe the problems and accomplishments, and present some recent works. Section 2 presents the motivation behind messy GAs and identify the main distinguishing features of mGAs. Section 3 reviews previous efforts in messy GA research and also points out the main accomplishments and shortcomings of the previous efforts. Section 4 presents the gene expression messy GA (GEMGA)—a recent development in messy GA research. The scalability of GEMGA for order-k delineable problems are supported by experimental results in Section 5. Finally, Section 6 concludes this paper.

2 Messy GAs: Why, What, And All That

Since most of the realistic optimization problems do not come with sufficient knowledge about the properties of objective function, optimization algorithms like genetic algorithms (Holland, 1975), simulated annealing (Kirpatrick, Gelatt, & Vecchi, 1983) that do not require much of problem information, gained popularity among practitioners. Unfortunately this has resulted in a race for developing apparently different, new algorithms with each claiming superiority from others. Some algorithms are quite general, while some of them are for special purposes. Some emphasize the role of representation and some do not. The nature of the search operators varies widely between algorithms. Given this situation, an introduction of mGAs first demand theoretical justification and motivation behind the effort. Section 2.1 presents this. Section 2.2 briefly describes the main feature of messy GAs.

2.1 Why messy GAs?

Since messy GAs are a class of genetic algorithms, it is natural to first discuss the strengths and weaknesses of simple GA (De Jong, 1975; Goldberg, 1989) in order to justify the development of mGAs. Doing that

*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

rigorously requires first defining what it means to be a good BBO algorithm. The SEARCH framework introduced elsewhere (Kargupta, 1995) provided a general framework for BBO by decomposing blackbox search in relation, class, and sample spaces. SEARCH also explored the conditions for polynomial complexity search on analytical ground. In the remaining portion of this paper we shall use SEARCH as the foundation for making qualitative remarks about the strengths and weaknesses of an algorithm.
The simple GA has many interesting features. The first important aspect is that it emphasizes the role of representation in search. Representation is one possible way to define a rich source of relations. Searching by constructing and ordering equivalence relations is very natural in GA. Apart from that, crossover provides an interesting tool to implement sample generation process from the intersection set of different equivalence classes. Probably one of the most interesting observations that Holland made in his book (1975) is the idea of implicit parallelism. Although this conjecture was not thoroughly laid in terms of computational arguments, the SEARCH framework confirms this observation in a quantitative manner. Moreover, the population based approach also makes GA highly suitable for parallel implementations. Despite these interesting features, the simple GA has several major problems. The main bottlenecks of the simple GA are listed below:

1. **relation, class, and sample spaces are all combined together:** Almost every paradigm of evolutionary algorithms including simple GA uses a single population of samples. This essentially means that the relation, class, and sample spaces are combined and as a result the decision making processes in each of them can affect others in an undesirable way. Only one selection operator is used for deciding in both relation and class spaces together.

2. **poor search for relations:** Simple GA does not emphasize the role of search for relations. Although Holland (1975) designed GAs along the right direction using the perspective schemata and partitions, unfortunately simple GA fails to process relations in a reasonable way. SGA with single point crossover does not exploit the complete relation space defined by the representation. It evaluates and processes only those relations that are defined over positions close to one another. Uniform crossover (Syswerda, 1989) does not let proper evaluations of relations unless selection produces more copies. Since the relation space and the sample space are combined, random perturbation of the sample strings also result in disrupting proper evaluations of the relations. Uniform crossover should also fail to accomplish proper search in the relation space. In fact, this is exactly what Thierens and Goldberg (1993) reported. Their analysis and experimental results showed that the sample complexity grows exponentially with the problem size for solving bounded deceptive problems (Thierens & Goldberg, 1993) using a simple GA with uniform crossover.

3. **Lack of precise mechanism for implicit parallelism:** Parallel evaluation of relations using the same sample set is often termed as implicit parallelism. Unfortunately, simple GA does not have a mechanism to precisely evaluate relations. This may result in decision error as a price for implicit parallelism.

Clearly, these are very fundamental bottlenecks and evolutionary algorithms cannot enjoy the taste of success for wide classes of problems unless these bottlenecks are eliminated. The main objective of messy GAs is to eliminate these problems of simple GA.

2.2 What is a messy GA?

The messy GAs have many unique differences from simple GAs. Although some of them like variable-length strings, local search template are implementation specific differences, there are several fundamental distinctions as listed in the following:

1. mGAs emphasize explicit processing of relations, classes, and samples;

2. Since in absence of relations blackbox search cannot be any better than random enumerative search, mGAs do not search for optimal solutions until appropriate relations are detected with desired accuracy and reliability. The search process is distinctly divided into two stages, namely:

   - Primordial stage: Detects appropriate relations and preserves better classes;
   - Juxtapositional stage: Computes intersection among the better classes to find the optimal solutions. This stage corresponds to the *resolution* operation of SEARCH.

3. Adequate decision making in relation and class spaces satisfying desired accuracy and reliability.

4. Messy GAs have been traditionally designed for solving quasi-decomposable problems with bounded inappropriateness of representation. In this paper, however, we shall use the recently proposed class of order-k delineable problems—problems that can be solved by searching for a polynomially bounded order of relations—(Kargupta, 1995) as the scope of messy GAs.

The following section presents a brief review of previous works on messy GAs.

3 Previous versions of messy GAs

The work on mGAs was initiated by Goldberg, Korb, and Deb (1989). The originally proposed version of messy GA (Deb, 1991; Goldberg, Korb, & Deb, 1989) took at least two important steps:

1. separated the relation and class spaces from the sample space.

2. deterministically processed all order-k relations and classes.

Original messy GA used a population that contained all (deterministically enumerated) order-k classes defined by the chosen representation. This population defined the relation and class spaces together. These classes were evaluated using a template string (a locally optimal solution), which defined the sample space. In other words, in mGA the sample space was comprised of the locally optimal solutions, found by the algorithm in different iterations. In a particular iteration the sample space was however defined by only one locally optimal solution, called template. This decomposition and the emphasis on search for relations made decision making in mGA more accurate compared to many other evolutionary algorithms.
Different versions of messy GAs studied different aspects of BBO by decomposing blackbox search along different dimensions. These investigations have directly influenced the development of SEARCH and the design of the GEMGA. Another interesting aspect of their work was the class of problems they wanted to solve. The class of bounded deceptive problems captures the essence of order-k delineability developed in SEARCH. An order-k bounded deceptive problem is order-k delineable with respect to class average comparison statistics. It has been shown elsewhere that order-k delineable problems can be solved in polynomial sample complexity. Messy GAs tries to solve this class of problems efficiently. The original version of messy GA was $O(|\Lambda|^k \ell^k)$ complexity algorithm, where ℓ is the problem length and Λ is the alphabet size of the representation.
Merkle (1992) developed a parallel implementation of original messy GA. Merkle and Lemont (1993) addressed data distribution strategies for the parallel implementation of mGA. Although the population size in primordial stage grows polynomially with problem size ℓ, $O(\ell^k)$ is a fairly large number for any reasonable value of k. Plevyak (1992) investigated the possibility of smaller population size in the primordial stage. Messy GAs have also been reported to be used for solving some application problem. Mohan (1993) applied mGAs for clustering. A hierarchical controller based on messy GAs is reported elsewhere (Hoffmann & Pfister, 1995).
The original messy GA had many problems. Some of them are listed in the following:

1. **Combined relation and class spaces:** In mGA the relation space is implicitly defined together with the class space. The SEARCH framework pointed out that these two different spaces require different decision makings and they should be processed separately in order to avoid undesirable errors in decision making.

2. **Sample space comprised of one template:** A locally optimal solution, the template, defines the sample space. Evaluating and comparing different classes on the basis of one member make not be appropriate.

3. **Lack of implicit parallelism:** The explicit enumeration of all order-k classes is very expensive ($O(|\Lambda|^k \ell^k)$). SEARCH pointed out that the same sample set can be used for evaluating different relations, which may be viewed as a quantification of the benefits of implicit parallelism. The mGA lacked such computational benefits.

The fast messy GA (fmGA) proposed elsewhere (Goldberg, Deb, Kargupta, & Harik, 1993; Kargupta, 1995) made an effort to reduce the cost of deterministic initialization by using the so called *probabilistically complete*

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

initialization (PCI) and *building-block filtering* (BBF) techniques. Instead of explicitly generating all the order-k classes, PCI technique initialized the population with order-ℓ classes. This is followed by the BBF process in which the good order-k classes are gradually detected using random gene deletion and thresholding selection. The fmGA indeed reduced the cost of initialization but it succumbed to a fundamental problem that all the versions of mGAs had—implicit definition of the relation space in the class space. The primary objective of the thresholding selection of messy GAs was to select good classes defined by the same relation. However, thresholding selection was also implicitly responsible for selecting good relations. This was simply because the chosen value of thresholding parameter was always less than the string length. The main problem of the fmGA was that thresholding selection could not satisfactorily maintain the growth of strings which are instances of good classes. Undesirable cross-competition among classes from different relations usually eliminated some of the good classes and as a result the algorithm required several expensive iterations for solving large problems (Kargupta, 1995). Apart from this problem, the fmGA also faced the same questions regarding the use of single local template. Another problem of fmGA was it was slow because of the thresholding selection and it required order ℓ population sizing since selection took place only at the string or sample level.

Despite several practical applications as noted earlier, different versions of the messy GAs primarily studied different fundamental issues and were used for verifying the theoretical observations. Nevertheless, messy GAs should get the credit for taking the following conceptual leaps: (1) realizing the need for searching for relations, (2) decomposing the search space into sample space and class space (with implicit definition of relation space), and (3) paying careful attention to the decision making in genetic algorithms.

In the following section, I shall present a new generation of messy GA, called the *gene expression messy GA* (GEMGA) that eliminates many of the above mentioned problems of messy GAs by taking the inspiration from the lessons of SEARCH and the alternate perspective of natural evolution offered by SEARCH.

4 The Gene Expression Messy GA

The main objective of the gene expression messy GA (GEMGA) is to eliminate the computational shortcomings of the previous versions of messy GA. Apart from the conceptual motivations, the GEMGA also has a strong biological motivation. The implementation of GEMGA is motivated by the intra-cellular expression genetic information often known as gene expression. Section 4.1 presents that. Section 4.2 discusses the representation in GEMGA. Section 4.3 explains the population sizing in GEMGA. This is followed by Section 4.4 that describes the main operators, transcription, selection, and recombination. Section 4.5 presents of the overall mechanisms.

4.1 The biological motivation

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. A more detailed description of the biological motivations behind GEMGA can be found elsewhere (Kargupta, 1996a).

4.2 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, which contains the *locus*, *value*, *weight* and 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. 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. 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.

4.3 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 population size in turn determines what order of relations will be processed. For a population size of 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. If GEMGA cannot solve the problem for a given population size, a higher population size should be used to address possible higher order delineability.

4.4 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.

4.4.1 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.

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

  dwt = chrom[pick].Weight();
    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 1: 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.

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 problem*, 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 1 shows the Transcription 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 2 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 described elsewhere (Kargupta, 1996b).

4.4.2 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 3 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.

4.4.3 Recombination

Figure 4 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 **chrom1** and **chrom2** corresponding to the members of **ExchangeSet** are exchanged.

The following section describes the overall mechanism of the algorithm.

4.5 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 5 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 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,

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

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 look back to our SEARCH framework 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 defines the resolution operation of SEARCH.

The following section presents the test results.

5 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, problem size, noisy objective function. In this paper, we present the performance of GEMGA for problems with varying degree of difficulties along the first three dimensions. The following sections describe the test functions and present the experimental results.

5.1 Experimental design

A test function is constructed by concatenating multiple numbers of order-5 trap functions (Ackley, 1987). Each of these subfunctions is an order-5 trap function. The particular version of the deceptive trap function used 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 string length. If we carefully observe this trap function, we shall note that it has two peaks. One of them corresponds to the string with all 1-s and the other is the string with all 0-s. 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. As the problem length increases the number of local optima exponentially increases. This class of problems is order-5 delineable with respect to the class average comparison statistics (i.e. when classes are compared with respect to the distribution means). This problem in order-5 deceptive representation has only $\ell/5$ proper relations among the $\binom{\ell}{5}$ order-5 relations. Therefore, searching for the appropriate relations is not a trivial job in this class of problems. This is the primary reason behind the failure of most of the existing blackbox optimization algorithms for such problems. Clearly this class of problems are massively multimodal and has bounded inappropriateness of the relation space, defined by the representation. The following section presents the test results.

5.2 Results

The GEMGA is tested against order-5 deceptive problems of different sizes. Figure 6 shows the average number of sample evaluations from six independent runs needed to find the globally optimal solution for different problem sizes. The population size is 500, chosen as described earlier in this paper. As we see, the sample complexity linearly depends on the problem size.

Figure 7 show the gradual detection of the relations during the primordial and juxtapositional stages for a 30-bit order-5 deceptive problem. Each figure represent the relation space of the whole population at a certain generation. The x-axis denotes the weights in the genes, ordered on the basis of the *locus* of the gene. In other words the values along the x-axis correspond to the actual value of the locus of a gene in a chromosome. The y-axis corresponds to the different members in the population. The z-axis, perpendicular to the page denotes the weights of the corresponding gene in the corresponding chromosome. Since the test function is comprised of order-5 trap functions, for any particular gene in a chromosome, there are only 4 other genes that are related with it. The complete relation space has a cardinality of 2^{30}. Among $\binom{30}{5}$ order-5 relations there are only 6 relations that correctly correspond to the actual dependencies defined by the problem. GEMGA needs to detect the relations that relate genes with loci ranging from 0 to 4 together, from 5 to 9 together and so on. These relations are gradually detected in different chromosomes that contain good classes from those relations. More instances of good classes are produced by selection and they are exchanged among different strings to create higher order relations that finally lead to the optimal solution. The following section concludes this paper.

6 Conclusion

Research on messy GAs contributed in understanding several important issues in blackbox optimization. The most important one is the search for appropriate relations from the relation space defined by the representation. Decomposition of search into relation, class, & sample spaces and decision theoretic construction of ordering among classes & relations are examples other important contributions. In this paper we also reviewed the shortcomings of earlier version of messy GAs. The GEMGA eliminated many of these problems. The main accomplishments of GEMGA 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. introducing high degree of parallelism (even more than simple GA), and
5. reducing the running time by a large factor.

Experimental results clearly showed that GEMGA can detect appropriate relations efficiently for different classes of problems. However, the transcription phase I and phase II requires more work. For example, the current version of transcription phase II will fail to work satisfactorily for some functions in which the overall objective function value is the product of the function values of individual subfunctions. Moreover, currently GEMGA can at most solve problems that are order-k delineable in the chosen representation. One future possibility is transforming non order-k delineable problems to delineable problems by representation construction.

7 Acknowledgment

This work was supported by US. Department of Energy. The author also acknowledges many useful discussions with Professor David E. Goldberg and Liwei Wang.

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.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. *Dissertation Abstracts International*, *36*(10), 5140B. (University Microfilms No. 76-9381).

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.

Forrest, S. (Ed.) (1993). *Proceedings of the Fifth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* New York: Addison-Wesley.

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). Rapid, accurate optimizaiton of difficult problems using fast messy genetic algorithms. See Forrest (1993), pp. 56–64.

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).

Hoffmann, F., & Pfister, G. (1995, July). *A new learning method for the design of hierarchical fuzzy controller using messy genetic algorithms.* Presented on IFSA'95, Sao Paulo.

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. (1996a). SEARCH and a Computational Perspective of Evolution. In *Proceedings of the Artificial Life V* (pp. 56–63).

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

Kirpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Merkle, L. D. (1992, December). *Generalization nd parallelization of messy genetic algorithms and communication in parallel genetic algorithms.* Master's thesis. Air Force Institute Of Technology, WPAFB OH 45433.

Merkle, L. D., & Lemont, G. B. (1993). Comparison of parallel messy genetic algorithm data distribution strategies. See Forrest (1993), pp. 191–205.

Mohan, C. K. (1993). A messy genetic algorithm for clustering. In Dagli, C. H., Burke, L. I., Fernádez, & Ghosh, J. (Eds.), *Intelligent Engineering Systems Through Artificial Neural Networks* (pp. 831–836). New York: ASME Press.

Plevyak, J. (1992). *A messy GA with small primordial population.*

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

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In Schaffer, J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 2–9).

Thierens, D., & Goldberg, D. (1993). Mixing in genetic algorithms. See Forrest (1993), pp. 38–45.

```
// 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) {
      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[pick1].SetWeight(1.0);
        chrom[pick2].SetWeight(1.0);
      }
      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 2: Transcription Phase II operator for minimization problem.

```
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 3: 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.

```
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 4: 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.

```
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
    TranscriptionPhaseI(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 5: Pseudo-code of GEMGA. The constant C ¡ |A|, where |A| is the cardinality of the alphabet set.

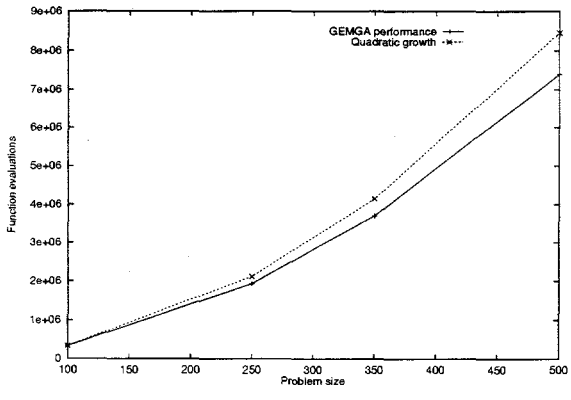

Figure 6: Growth of the number of function evaluations with problem size.

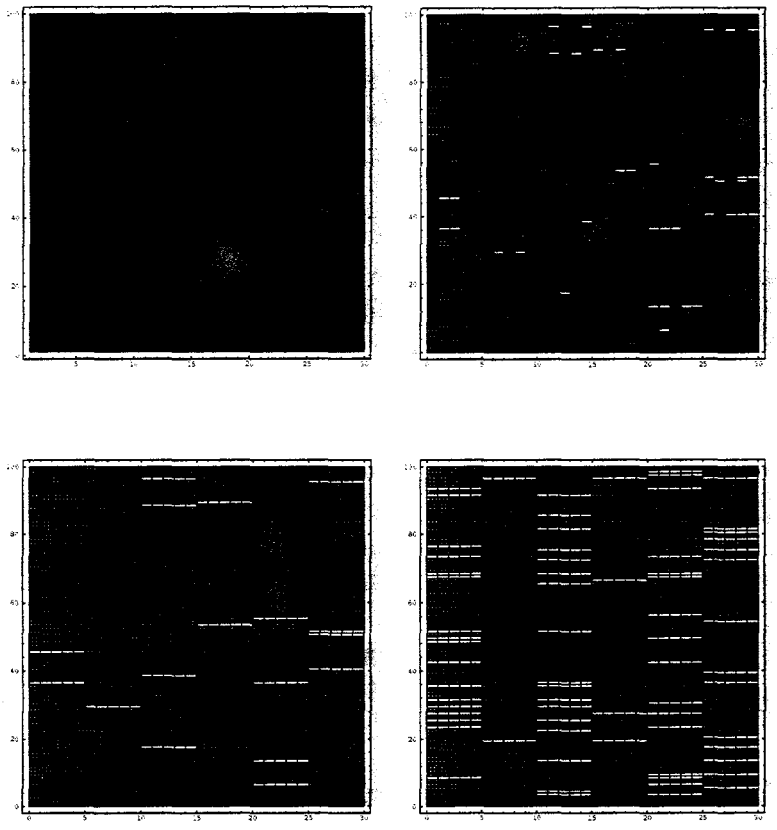

Figure 7: The relation space during primordial generation 1, 10, and juxtapositional generations 1, 4 (from top to bottom).