Revisiting The GEMGA: Scalable Evolutionary Optimization Through Linkage Learning

Sanghamitra Bandyopadhyay, Hillol Kargupta, and Gang Wang

Abstract— **The *Gene expression messy genetic algorithm* (GEMGA) is a new generation of messy genetic algorithms (GAs) that pays careful attention to linkage learning and in a broader context the search for appropriate relations [6]. This paper revisits the earlier work on the GEMGA [8], [9], [7] and proposes a new version of the algorithm that brings back the time complexity to $O(\Lambda^k(\ell))$, where ℓ is the problem length, Λ is the alphabet set of the representation, and k is the order of delineability [6] of the problem. This paper also reports the scalable linear performance of the GEMGA for various difficult, large optimization problems.**

Keywords— **GEMGA, linkage learning, messy GAs.**

I. Introduction

Inducing relations that capture the local symmetries of the fitness landscape is important for making a blackbox search transcend random enumeration. The gene expression messy GA (GEMGA) is a class of evolutionary algorithms that pays careful attention to this process. Foundation of the GEMGA is built on the decomposition of black box optimization (BBO) into three spaces (proposed by the SEARCH framework [6]): (1) relation, (2) class, and (3) sample spaces. The relation space introduces classes which in turn define the set of symmetries that the BBO algorithm tries to induce from the search space. This is essentially defined by the representation, operators, and other biases of the algorithm. Like many other GAs, the GEMGA uses the representation to define this prior bias for the set of symmetries (a set of relations in set theoretic terms) to look for. In the current version of the GEMGA this set of relations is actually a set of partitions defined over the canonical basis of the chosen representation. The GEMGA makes an attempt to solve the so called *order-k delineable* problems, the class of problems in which partitions, defined by a constant (k) number of dimensions of the canonical basis set are sufficient for capturing fitness symmetries. The GEMGA is also a descendent of the messy GA breed [2], [4], [3], [6]. The GEMGA shares the philosophy of linkage learning in messy GAs and emphasizes it to even broader context of learning relations.

Although the initial version of the GEMGA [9], [7] was linear in sample complexity (growth of number of function evaluations along problem size), better performance in terms of the relation search required introducing quadratic experimentation [11]. This paper presents a new version of GEMGA, where *linkage learning* (a special case of learning relations using canonical basis of the representation) is significantly modified, bringing back the GEMGA to the domain of linear sample complexity without sacrificing the performance.

Section 2 describes different aspects of GEMGA. Section III presents the test results for several large, multimodal, order-k delineable problems. Finally, Section IV concludes this paper.

S. Bandyopadhyay is with the Machine Intelligence Unit, Indian Statistical Institute, Calcutta.

H. Kargupta is with the Dept. of Computer Science, Washington State University, Pullman; all correspondences should be sent to hillol@eecs.wsu.edu

G. Wang is with the Dept. of Computer Science, Michigan State University at East Lansing.

II. The Gene Expression Messy GA

This section introduces a modified version of the GEMGA and shows that the overall sample complexity is linear. Section II-A discusses the representation in GEMGA. Section II-B explains the population sizing in GEMGA. This is followed by Section II-C that describes the main operators, Transcription and RecombinationExpression. Section II-D presents of the overall mechanisms.

A. 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*, and *capacity*. The chromosome also contains a dynamic list of lists called the *linkage set*. The *locus* determines the position of the member in the sequence. The GEMGA uses position independent coding of genes introduced elsewhere [2], [4]. A gene also contains the *value*, which determines the value of the gene, which could be any member of the alphabet set, Λ. The *capacity* associated with every gene takes a positive real value. The *linkage set* of a chromosome is a list of weighted lists. Each member of this sequence consists of a list, termed *locuslist* which defines the set of genes that are related, and three factors, the *weight*, *goodness* and *trials*. The weight is a measure of the number of times that the genes in locuslist are found to be related in the population. The goodness value indicates how good the linkage of the genes is in terms of its contribution to the fitness. This value is normalized between 0 and 1, and is initialized to 0. The trial field indicates the number of times this linkage set has been tried. (Note that if the trial of any element of the linkage set is zero, then its goodness is temporarily assumed to be 1, unless proved otherwise.) The *linkage set* space over all genes defines the relation space of the GEMGA. Figure 1 shows the structure of a chromosome in GEMGA. Unlike the original messy GA [2], [4] no under or over-specifications are allowed. A population in GEMGA is a collection of such chromosomes.

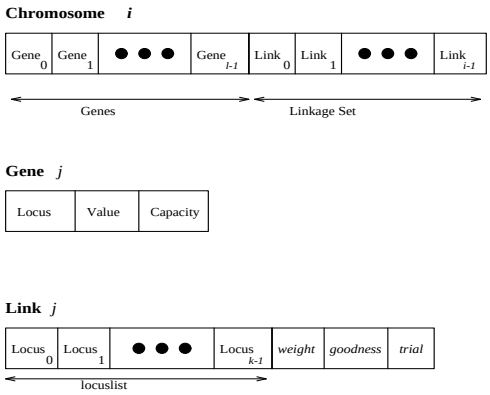

Fig. 1. Structure of a chromosome in GEMGA.

B. Population sizing

In GEMGA symmetries of the fitness are searched for using equivalence classes called schemata [5] defined by the partitions over canonical basis set. The GEMGA requires at least one instance of the optimal order-k schemata 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 schemata. The population size in GEMGA is therefore, $n = c\Lambda^k$, where c is a constant. Although we treat c as a constant, c is likely to depend on the variation of fitness values of the members of schema. The GEMGA only searches for schemata defined by only order-k delineable partitions. In practice the order of delineability [6] 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, $k = \log(n/c)/log|\Lambda|$. Since, the current version of GEMGA does not construct new basis set, if GEMGA cannot solve the problem for a given population size, a higher population size should be used to address possible higher order delineability.

C. Operators

GEMGA has two primary operators, namely: (1) *Transcription*, (2) *RecombinationExpression*. Each of them is described in the following.

C.1 Transcription

In nature the transcription operator may apparently look quite different from the GEMGA transcription operator. However, there are fundamental similarities and in the future GEMGA transcription may look quite similar to its natural counter part. We view natural transcription as a representation transformation defined over the alphabet set of the DNA. We also view the gene expression process as a process of representation construction for defining new basis set. However, although the GEMGA research is motivated by gene expression, the research agenda is decomposed into two stages namely (1) solving problems accurately and reliably that are order-k delineable in the given canonical basis set and (2) construction of new basis set. The current version of GEMGA reported in this paper makes not attempt to address the second stage. However, our research directions for the next stage is discussed elsewhere [10]. Since, current GEMGA deals with only the given canonical representation, unlike natural transcription, the GEMGA makes use of transformations over the canonical basis to detect search space symmetries. This is the fundamental reason behind using the name transcription for the operator described below. The transcription operator applies a random subset of all alphabet transformations to every gene one at a time. The value of the gene is flipped and the change in fitness value is noted. The objective is to note the "local" symmetry of the fitness landscape in a statistical sense. For a *minimization problem*, if that change causes an improvement in the fitness (i.e. fitness decreases) then the original instance of the gene certainly does not belong to the instance of the best schema, since fitness can be further improved. Transcription sets the corresponding capacity of the gene to one (for binary problems, indicating that the gene has a capacity to change by one). 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 capacity of the gene is set to zero, indicating that the value at that gene position cannot be changed. Finally, the value of that gene is set to the original value and the fitness of the chromosome is set to the original fitness. All the genes whose capacities are reduced to zeroes are collected in one set, called the initial linkage set. This is stored as the first element of the linkage set associated with the chromosome. Its weight, goodness, and trial factors are initialized to 1, 0, and 0 respectively. The transcription operator does not change anything in a chromosome except the capacities and initiates the formation of the linkage sets. For a maximization problem the conditions for the capacity change are just reversed. The same process is continued deterministically for all the ℓ genes in every chromosome of the population. Figure 2 shows the Transcription operator. The following section describes the RecombinationExpression operator in GEMGA.

C.2 RecombinationExpression

Figure 3 shows the mechanism of the Recombination-Expression operator in GEMGA. It primarily consists of two phases - the PreRecombinationExpression phase and the GEMGA Recombination phase. Please note that the recombination operator as described here is different from the conventional notions of recombination in GAs. Careful study of this operator will reveal many similarities with the overall process of recombination in nature.

The PreRecombinationExpression operator determines the clusters of genes precisely defining the relations among those instances of genes. This is applied several times, specified by NoOfLinkageExpt, during the first generation for the chromosomes. First, a pair of chromosomes is selected and one of them is marked. Of the genes present in the initial linkage set of the marked chromosome (and included as the first element of its linkage set), only those that have the same value and capacities in the other are extracted and

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

  dcap = chrom[pick].Capacity();
  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)
    chrom[pick].SetCapacity(0.0);
  else
    chrom[pick].SetCapacity(1.0);
  // Set the value to the original value
  chrom[pick].SetValue(dummy);
  // Set the original fitness
  chrom[pick].SetFitness(phi);
  if (chrom[pick].capacity == 0)
    chrom.LinkageSet[0].Add(pick);
  chrom.LinkageSet[0].Init();
}
```

Fig. 2. Transcription operator for minimization problem. For maximization problem, if `delta`< 0 then the capacity is set to 0. Function Add() adds element *pick* to the locuslist of LinkageSet[0] and Init() initializes the other factors.

```
RecombinationExpression(Pop, gen )
POPULATION Pop;
int gen; //the generation number
{
  int i,sel;
  if (gen == 0){
    for (i=0;i<=NoOfLinkageExpt;i++)
      PreRecombinationExp(Pop,i)
  }
  else Recombination(Pop)
}
```

Fig. 3. RecombinationExpression operator in GEMGA.

grouped as a separate set. If this set is already present in the linkage set of the marked chromosome, then the corresponding weight is incremented by INCR_WEIGHT. Otherwise, it is included as a new linkage set and the different factors are initialized. The operator is outlined in Figure 4.

At the end of the requisite number of experiments (NoOfLinkageExpt), an $\ell \times \ell$ conditional probability matrix is formed, (Figure 5), whose entry i, j indicates the probability of the occurrence of gene i, when gene j is present in a linkage set. Finally, the final linkage sets are computed using the GetFinalLinkage operator. For each row i of the Conditional matrix, its maximum value is computed, and the genes that have their probability values within an EPSILON of the maximum are included in the linkage set for i. Its weight is set to the average value of the conditional probabilities of every gene in the set. Figure 6 shows the pseudo code for this operator.

```
PreRecombinationExpression(chrom1,
chrom2,counter)
CHROMOSOME chrom1,chrom2;
int counter; // no of expt counter
{
  int i,n,index,locus,ln,R[];
  double Conditional[][];
  if (counter<NoOfLinkageExpt){
    ln=chrom1.LinkageSet[0].Length();
    for (i=0;i<ln;i++){
      locus=chrom1.LinkageSet[0][i];
      if ( (chrom1[locus].value==
      chrom2[locus].value) and
      chrom2[locus].capacity == 0)
        R.Add(locus);
    }
    if (PresentIn(R,chrom1.LinkageSet,
        index))
      chrom1.LinkageSet[index].
        weight+=INCR_WEIGHT;
    else
      chrom1.LinkageSet.Add(R,INIT_WEIGHT);
  }
  else{ //at end of this phase
    Compute_Conditional_Matrix(Pop
      i,Conditional);
    GetFinalLinkage(Pop,i,Conditional);
  }
}
```

Fig. 4. PreRecombination operator using the linkage set of chrom2 to resolve the initial linkage set of chrom1. PresentIn(A,B,n) returns true if A is present in list B, and the position in n. Otherwise it returns false.

After the PreRecombinationExpression phase, the GEMGA Recombination operator, Figure 7, is applied iteratively on pairs of chromosomes. First, copies of a given pair is made, and one of them is marked. An element of the linkage set of the marked chromosome is selected, based on a linearly combined factor of its weight and goodness, for swapping. The corresponding genes are swapped between the two chromosomes provided the goodness values of the disrupted linkage sets of the unmarked chromosome are less than that of the selected one. The linkage sets of the two chromosomes are adjusted accordingly. Depending on whether the fitness of the unmarked chromosome decreases or not, the goodness of the selected linkage set element is decreased or increased. Finally, only two of the four chromosomes (including the two original copies) are retained based on several factors outlines in Figure 7.

The following section describes the overall mechanism of the algorithm.

D. The algorithm

GEMGA has two distinct phases: (1) Transcription stage and (2) RecombinationExpression stage. The transcription operator is applied for ℓ generations, deterministically considering every gene in each generation. This is followed by the application of the RecombinationExpression operator which continues a number of times determined by some termination criterion. This stage also has two distinct phases: an initial PreRecombinationexpression stage and GEMGA Recombination stage.

During the PreRecombinationExpression stage the pop-

```
Compute_Conditional_Matrix(chrom
Conditional[][])
CHROMOSOME chrom;
double Conditional[][];
{
  int cnt1,cnt2;

  For every pair of genes (i,j) in
  chrom.LinkageSet[0] {
    cnt1=No.  of times (i,j) in
    chrom.LinkageSet;
    cnt2=No.  of times (j) in
    chrom.LinkageSet;
    Conditional[i][j]=cnt1/cnt2;
  }
}
```

Fig. 5. Compute_Conditional_Matrix operator.

```
GetFinalLinkage(chrom,Conditional[][])
CHROMOSOME chrom;
double Conditional[][];
{
  int i,j,ln,cnt=0,Set[];
  double max,wt=0;

  ln = chrom.LinkageSet[0].Length();
  for (i=0;i<ln;i++) {
  //for each element in initial linkage
    Set=chrom.LinkageSet=NULL;
    max=Max(Conditional[i]);
    for (j=0;j<ln;j++) {
      if (Conditional[i][j]>
        max-EPSILON){
        //j in Linkage Set for i
        Set.Add(j);
        wt = wt+Conditional[i][j];
        cnt=cnt+1;
      }
      Set.Add(i); //Linkage set for i
      wt = wt/cnt ;
      if (wt>WEIGHT_THRESH)
      //Linkage thresholding criterion
        if (!PresentIn(Set,
        chrom.LinkageSet))
      chrom.LinkageSet.Add(Set,wt);
  }
}
```

Fig. 6. GetFinalLinkage operator using the Conditional Matrix to get final linkage sets of chrom. Add(Set,wt) adds 'Set' to the linkage set of chrom and sets its weight to 'wt'. The goodness and trial values are set to zeros.

ulation of chromosomes remains unchanged, except that the capacities of the genes change and the linkage sets get constructed. This is followed by the Recombination stage, in which the detected linkage sets are combined based on conditions of goodness, weight, trial, and fitness. Figure 8 shows the overall algorithm. The length of the transcription phase application is ℓ. In the PreRecombinationExpression phase, no function evaluation is performed. Since the population size is $O(|\Lambda|^k)$, the transcription phase is applied for ℓ generations and no function evaluation is performed in the PreRecombinationExpression phase, the overall sample complexity of the phases during which the linkages are learned is $SC = O(|\Lambda|^k(\ell))$.

The following section presents the test results.

```
Recombination(chrom1,chrom2)
CHROMOSOME chrom1, chrom2;
{
  //let chrom1 be the marked chromosome
  CHROMOSOME t1,t2;
  int maxtrial;
  double maxwt,maxgood;
  LinkageSet SelSet[];

  //make copies of the chromosomes
  t1=chrom1; t2=chrom2;
  //select linkage set from chrom1 based on
  //linear combination of goodness & weight
  SelSet=Select(chrom1.LinkageSet);
  //Get max weight, goodness & trial among
  //linkage sets disrupted in chrom2 due
  //to SelSet
  chrom2.Disrupted(SelSet,maxwt,
      maxgood,maxtrial);
  if (SelSet.goodness ≥ maxgood) {
    //the swap the genes in SelSet
    //Adjust linkage sets of chrom1 & chrom2
    //due to the swapping of SelSet
    AdjustLinkage(chrom1,chrom2,SelSet);
    chrom1.Fitness();
    chrom2.Fitness();
    if (chrom2.fitness ≥ t2.fitness)
      //SelSet good - increase its goodness
      //in t1 and chrom2.
    else
      //SelSet bad - decrease its goodness
      //in t1 and chrom2
    //Incr. trial of SelSet in t1 and chrom2.
    //Select 2 offspring among chrom1,
    //chrom2, t1 and t2 as follows
    if (goodness(chrom2,SelSet)) ≥
      maxgood AND maxtrial > 0)
      //More copies of SelSet -
      //Select t1 and chrom2
    if (chrom2.fitness == t2.fitness)
      //Retain t1 and t2, but with changed
      //goodness and trial of SelSet.
    if (chrom2.fitness < t2.fitness)
      //Keep chrom1 and t2.
  }
}
```

Fig. 7. Recombination operator of GEMGA.

III. Test Results

The following sections document the performance of the GEMGA for problems with massive multi-modality, bounded delineability (approximated by bounded deception [4]), and large number of optimization variables.

A. Experiment design

The performance of GEMGA is tested for five different problems, namely *i) Liepins-Vose deceptive (LVD), ii) Muehlenbein, iii) Goldberg-Wang function 1 (GW1), iv) Goldberg Wang function 2 (GW2)* and *v) Massively-Multimodal function.* Each of the functions is constructed by concatenating order-5 subfunctions, so that the order of delineability is bound by 5. The subfunctions are now described in details:

i) LVD The following defines the deceptive trap [1] function.

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

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

  // Initialize the population at random
  Initialize(Pop);
  j = 0;
  Repeat {
    // Identify better relations
    Transcription(Pop, j);
    // Increment generation counter
    j = j + 1;
  } Until(j == Problem_length)
  k = 0;
  Repeat {
    // Select better classes and
    do GEMGA recombination
    RecombinationExpression(Pop,k);
    // Increment generation counter
    k = k + 1;
  } Until ( k > k_max )
}
```

Fig. 8. Pseudo-code of GEMGA.

where u is the unitation variable, or the number of 1-s in the string x, and k is the length of the subfunction. 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 $\ell = 200$, and $k = 5$, 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.

ii) Muehlenbein Each order-5 subfunction is defined as follows :

$$\begin{aligned} f(x) &= 4 && \text{if } x = 00000 \\ f(x) &= 3 && \text{if } x = 00001 \\ f(x) &= 2 && \text{if } x = 00011 \\ f(x) &= 1 && \text{if } x = 00111 \\ f(x) &= 0 && \text{if } x = 01111 \\ f(x) &= 3.5 && \text{if } x = 11111 \\ &= 0 && \text{otherwise.} \end{aligned}$$

Thus the global optima is the string of all 0-s while all the strings having a number of trailing 1-s constitute the local optima. Unlike the case for *LVD*, here the building block corresponding to the global optima, has a significant amount of overlap with the local optimas.

iii) GW1 Each order-5 subfunction is defined as follows :

$$\begin{aligned} f(x) &= 4 && \text{if } x = 1\#1\#0 \\ f(x) &= 8 && \text{if } x = 1\#0\#0 \\ f(x) &= 10 && \text{if } x = 0\#1\#0 \\ &= 0 && \text{otherwise,} \end{aligned}$$

where # denotes the don't care position.

iv) GW2 Each order-5 subfunction is defined as follows :

$$\begin{aligned} f(x) &= 10 && \text{if } u = 0 \\ f(x) &= 8 && \text{if } u = k \\ f(x) &= 7 && \text{if } u = 1 \textit{ and } odd(0) \\ f(x) &= 2 && \text{if } u = 1 \textit{ and } even(0) \\ f(x) &= 4 && \text{if } u = k-1 \textit{ and } odd(1) \\ f(x) &= 3 && \text{if } u = k-1 \textit{ and } even(1) \\ &= 0 && \text{otherwise,} \end{aligned}$$

where odd(0) and even(0) return true if the number of 0-s in x are odd and even respectively. odd(1) and even(1) are analogously defined.

v) Massively multimodal This is a massively multimodal function of unitation where the global optima is a string of all 1's (assuming that length of the subfunction is odd). It is defined as follows :

$$f(x) = u + 2 \times f'(x).$$

$f'(x)$ is defined as follows :

$$\begin{aligned} f'(x) &= 1 \text{ if } odd(u) \\ &= 0 \text{ otherwise.} \end{aligned}$$

It can be observed that this function resembles a one-max function with "bumps". The following section presents the test results.

B. Results

Figures 9—13 show the average number of sample evaluations from five independent runs needed to find the globally optimal solution for problem sizes ranging from 100 to 500. The population size is 200, chosen as described earlier in this paper. It is kept constant for all the problem sizes. Selection probability is kept as zero. For the *LVD, Muehlenbein* and *GW1* problems, the NoOfLinkageExpt, WEIGHT_THRESHOLD and EPSILON values are kept as 150, 0.7 and 0.1, while for the remaining two these are 195, 0.4 and 0.1 respectively. The reason for this is that the nature of the first three problems results in the optimal buliding blocks being detected in the PreRecombinationExpresion with very high weights, close to 1, while those in the other two problems come up with low weights. It is only when the the sub optimal building blocks are tried, and they provide poor goodness values that the optimal building blocks get the opportunity to prove their efficacy. In each case we see that the sample complexity, or the number of function evaluations required for attaining the optimal value, linearly depends on the problem size.

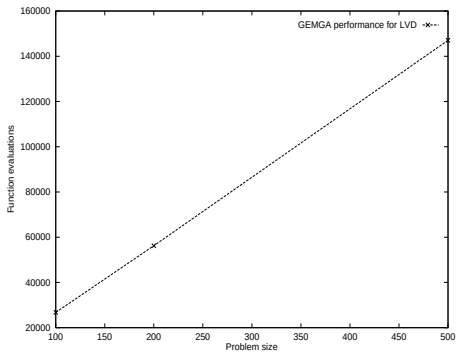

Fig. 9. Growth of the number of function evaluations to attain the optimum solution with problem size for *LVD*.

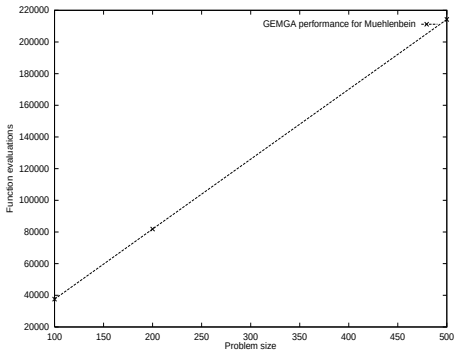

Fig. 10. Growth of the number of function evaluations to attain the optimum solution with problem size for *Muehlenbein*.

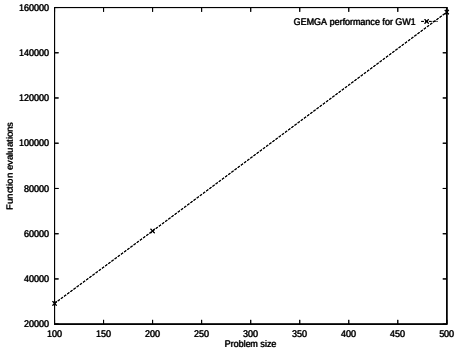

Fig. 11. Growth of the number of function evaluations to attain the optimum solution with problem size for *GW1*.

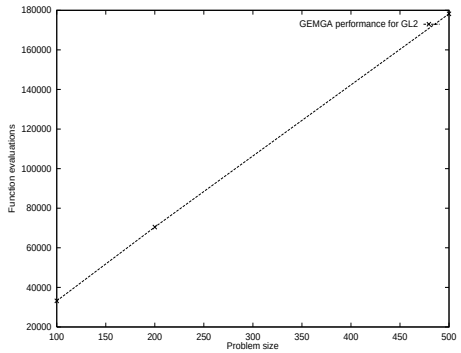

Fig. 12. Growth of the number of function evaluations to attain the optimum solution with problem size for *GW2*.

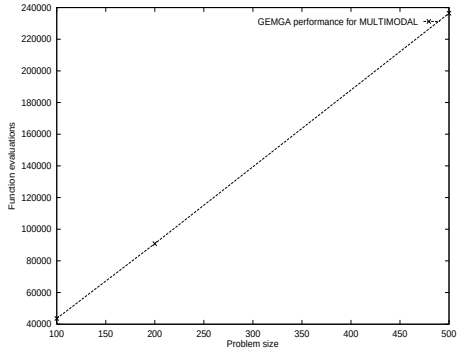

Fig. 13. Growth of the number of function evaluations to attain the optimum solution with problem size for *Massively Multimodal*.

IV. Conclusion

This paper introduces a significantly modified version of GEMGA and the test results for a large problem with millions of local optima and bounded inappropriateness of the representation. This version (v. 1.3) offers a linear sample complexity algorithm. The results of the investigation presented here indicate that the algorithm can detect appropriate relations efficiently for a large class of problems. Currently it is being tested for problems with real variables, larger problem size and variable fitness scaling. Problems lacking very crisp and well defined building blocks have been found to come up with very large number of elements in the linkage sets, thereby increasing the computation time of the algorithms. Improvements in this regard are also under investigation. The work on advancing the GEMGA to the second stage where it will construct new basis set is also underway; more details about that effort can be found elsewhere [10].

V. Acknowledgment

This work was supported by Los Alamos National Laboratory, United States Department of Energy. We also acknowledge many useful discussions with David E. Goldberg.

References

[1] D. H. Ackley. *A connectionist machine for genetic hill climbing*. Kluwer Academic, Boston, 1987.

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

[3] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimizaiton of difficult problems using fast messy genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, San Mateo, CA, 1993. Morgan Kaufmann.

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

[5] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[6] H. Kargupta. *SEARCH, Polynomial Complexity, and The Fast Messy Genetic Algorithm*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA, October 1995. Also available as IlliGAL Report 95008.

[7] H. Kargupta. Computational processes of evolution: The SEARCH perspective. Presented in SIAM Annual Meeting, 1996 as the winner of the 1996 SIAM Annual Best Student Paper Prize, July 1996.

[8] H. Kargupta. The gene expression messy genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 814–819. IEEE Press, 1996.

[9] H. Kargupta. Performance of the gene expression messy genetic algorithm on real test functions. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 631–636, 1996. IEEE Press.

[10] H. Kargupta. Gene Expression: The Missing Link Of Evolutionary Computation. In C. Poloni D. Quagliarella, J. Periaux and G. Winter, editors, *Genetic Algorithms in Engineering and Computer Science.*, page Chapter 4. John Wiley & Sons Ltd., 1997.

[11] H. Kargupta and D. E. Goldberg. SEARCH, blackbox optimization, and sample complexity. In R. Belew and M. Vose, editors, *Foundations of Genetic Algorithms*, pages 291–324, San Mateo, CA, 1996. Morgan Kaufmann.