The Gene Expression Messy Genetic Algorithm

Hillol Kargupta
Computational Science Methods Group
X Division, Los Alamos National Laboratory
Los Alamos, NM, 87545
hillol@lanl.gov

Published in the Proceedings Of The IEEE International Conference On Evolutionary Computation, Nagoya, Japan, 1996

ABSTRACT

This paper introduces the *gene expression messy genetic algorithm* (GEMGA)—a new generation of messy GAs that directly search for relations among the members of the search space. The GEMGA is an $O(\Lambda^k(\ell^2 + k))$ sample complexity algorithm for the class of order-k delineable problems [6] (problems that can be solved by considering no higher than order-k relations). The GEMGA is designed based on an alternate perspective of natural evolution proposed by the SEARCH framework [6] that emphasizes the role of gene expression. The GEMGA uses the *transcription* operator to search for relations. This paper also presents the test results of the GEMGA for large multimodal order-k delineable problems.

I. Introduction

The field of evolutionary computation is deluged with many algorithms. Introducing yet another evolutionary algorithm demands a strong justification. The SEARCH (Search Envisioned As Relation and Class Hierarchizing) framework, introduced elsewhere [6] offered an alternate perspective of blackbox optimization (BBO) in terms of relations, classes and partial ordering. SEARCH is primarily motivated by the observation that searching for optimal solution in a BBO is essentially an *inductive process* [8] and in absence of any relation among the members of the search space, induction is no better than enumeration [9]. SEARCH decomposed BBO into three spaces: (1) relation, (2) class, and (3) sample spaces. SEARCH also identified the importance of searching for appropriate relations in BBO. No BBO algorithm can efficiently solve a reasonably general class of problems unless it searches for relations. Kargupta [6] also showed that the class of *order-k delineable* (problems that can be solved by considering no higher than order-k relations) problems can be solved in SEARCH with sample complexity polynomial in problem size, desired quality and reliability of the solution.

Unfortunately, most of the existing evolutionary algorithms do not pay careful attention to the fundamental components of BBO. For example, the search for relations is often inadequate; the decision makings in relation and class spaces often interfere with each other. This paper introduces a new BBO algorithm called *gene expression messy GA* (GEMGA) that tries to transcend these bottlenecks. The GEMGA is a product of the development of the SEARCH framework and the series of efforts made by Goldberg and his students [3; 5; 4; 6] in designing BBO algorithms that work for reasonably general class of BBO problems. The GEMGA is an $O(\Lambda^k(\ell^2 + k))$ sample complexity algorithm for order-k delineable problems in sequence representation of length ℓ.

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

II. The Gene Expression Messy GA

This section introduces a modified version of the GEMGA and shows that the overall sample complexity is subquadratic. 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, selection, and recombination. 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*, *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 [3; 5] 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 [3; 5] no under or overspecifictions are allowed. A population in GEMGA is a collection of such chromosomes.

B. 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 [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, $k = \log(n/c)/log|\Lambda|$. If the problem is order-k delineable [6] 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.

C. 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.

A. 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 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

```
// 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);
}
```

Fig. 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.

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 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 op-

```
// 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);
  }
}
```

Fig. 2. Transcription Phase II operator for minimization problem.

erator. 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 [7].

B. 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 [2; 5] 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.

```
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]];
}
```

Fig. 3. Class selection 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.

The following section describes the recombination operator in GEMGA.

C. 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.

```
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;
}
}
```

Fig. 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.

D. 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,

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

```
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 )
}
```

Fig. 5. Pseudo-code of GEMGA. The constant C ¡ $|\Lambda|$, where $|\Lambda|$ is the cardinality of the alphabet set.

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.

The following section presents the test results.

III. 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.

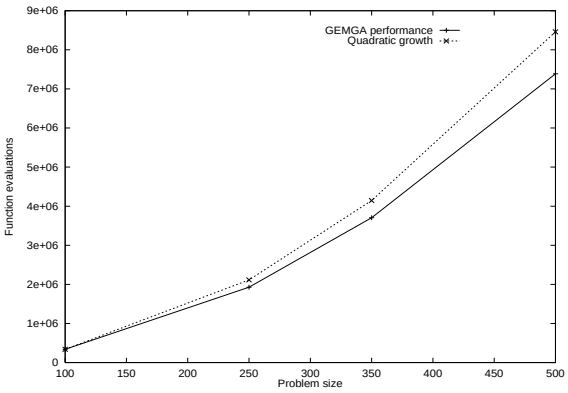

Fig. 6. Growth of the number of function evaluations with problem size.

A. Experimental design

A test function is constructed by concatenating multiple numbers of order-5 trap functions [1]. Each of these subfunctions is an order-5 trap function. The particular version of the deceptive trap function used can be defined as follows:

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

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.

B. 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.

IV. Conclusion

This paper presents a brief introduction of GEMGA and the test results for large problems with millions of local optima and bounded inappropriateness of the representation. 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) 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 a large class of problems. Unlike natural evolution, GEMGA does not construct new representation. That is the immediate future possibility. Currently GEMGA is designed for problems that are order-k delineable in the chosen representation. GEMGA has been tested for different class of problems producing very encouraging results. Those results are not included in this paper because of the restricted space.

V. Acknowledgment

The early stages of this work was supported by AFSOR Grant F49620-94-1-0103 and the Illinois genetic Algorithm Laboratory. The following stages of the design and experimentation have been performed at Los Alamos national Laboratory under the auspices of the US. Department of Energy. The author also acknowledges many useful discussions with Professor David E. Goldberg and Georges Harik.

References

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

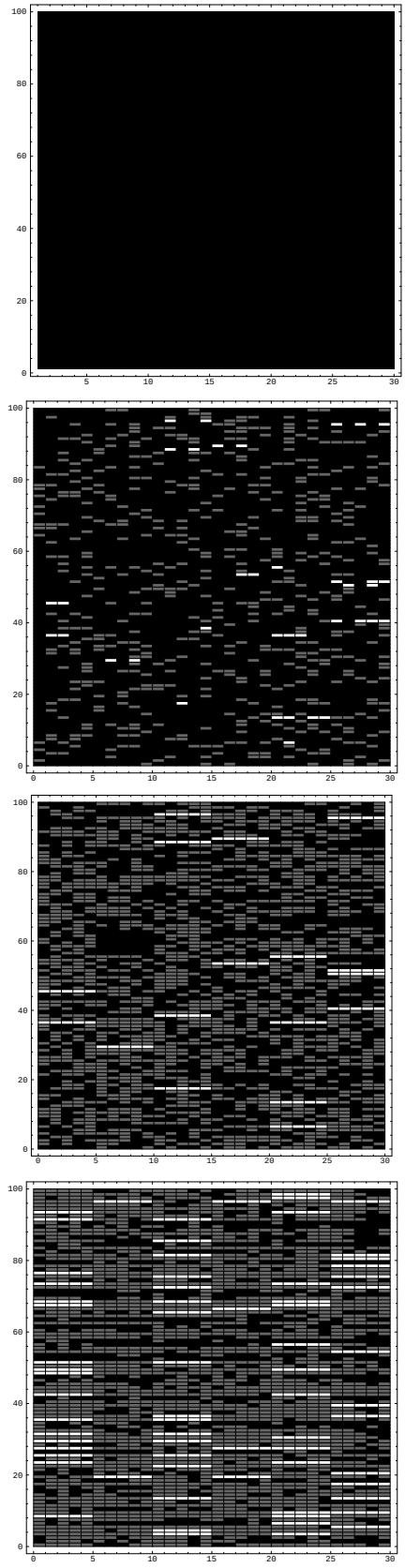

Fig. 7. The relation space during primordial generation 1, 10, and juxtapositional generations 1, 4 (from top to bottom).

[2] A. Brindle. *Genetic Algorithms for Function Optimization.* Unpublished doctoral dissertation, University of Alberta, Edmonton, Canada, 1981.

[3] K. Deb. Binary and floating-point function optimization using messy genetic algorithms. IlliGAL Report no. 91004 and doctoral dissertation, unversity of alabama, tuscaloosa, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, 1991.

[4] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimizaiton of difficult problems using fast messy genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, 1993.

[5] 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).

[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] R. S. Michalski. Theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine learning: An artificial intelligence approach*, pages 323–348. Tioga Publishing Co, 1983.

[9] S. Watanabe. *Knowing and guessing - A formal and quantitative study.* John Wiley & Sons, Inc., New York, 1969.