LA-UR- 96-429

CONF-960820--2

Title: COMPUTATIONAL PROCESSES OF EVOLUTION AND THE GENE EXPRESSION MESSY GENETIC ALGORITHM

Author(s): H. Kargupta

Submitted to: Foundations of genetic algorithms (FOGA4)
San Diego, CA
August 1996

Los Alamos
NATIONAL LABORATORY

Computational Processes Of Evolution And The Gene Expression Messy Genetic Algorithm

Hillol Kargupta[*]
Computational Science Methods division
Los Alamos National Laboratory
Los Alamos, NM, USA.

Abstract

This paper makes an effort to project the theoretical lessons of the SEARCH (Search Envisioned As Relation and Class Hierarchizing) framework introduced elsewhere (Kargupta, 1995b) in the context of natural evolution and introduce 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 $O(|\Lambda|^k(\ell+k))$ sample complexity algorithm for the class of order-k delineable problems (Kargupta, 1995a) (problems that can be solved by considering no higher than order-k relations) in sequence representation of length ℓ and alphabet set Λ. Unlike the traditional evolutionary search algorithms, the GEMGA emphasizes the computational role of gene expression and uses a *transcription* operator to detect appropriate relations. Theoretical conclusions are also substantiated by experimental results for large multimodal problems with bounded inappropriatness of representation.

1 Introduction

The SEARCH (Search Envisioned As Relation and Class Hierarchizing) framework introduced elsewhere (Kargupta, 1995a) offered an alternate perspective of blackbox search (BBS) in terms of relations, classes and partial ordering. SEARCH is primarily motivated by the observation that searching for optimal solution in a BBS is essentially an *inductive process* (Michalski, 1983) and in absence of any relation among the members of the search space, induction is no better than enumeration (Watanabe, 1969). SEARCH decomposed BBS into three spaces: (1) relation, (2) class, and (3) sample spaces. SEARCH also identified the importance of searching for appropriate relations in BBS. No BBS algorithm can efficiently solve a reasonably general class of problems unless it searches for relations. Kargupta (1995a) also showed that the class of *order-k delineable* problems can be solved in SEARCH with sample complexity polynomial in problem size, desired quality and reliability of the solution.

In this paper, I use the SEARCH framework to propose an alternate computational perspective of natural evolution. This perspective emphasizes the role of *gene expression* (DNA→RNA→Protein) in natural evolution and identifies gene regulatory mechanism, proteins, and DNA in terms of relation, class, and sample space respectively. This decomposition of evolutionary search process, backed by the SEARCH framework leads to the development of a new $O(|\Lambda|^k(\ell+k))$ BBS algorithm called *gene expression messy GA* (GEMGA) for order-k delineable problems in sequence representation of length ℓ with alphabet Λ.

Long before the development SEARCH framework, Goldberg and his students (Deb, 1991; Goldberg, Korb, & Deb, 1989; Goldberg, Deb, Kargupta, & Harik, 1993; Kargupta, 1995a) realized the importance of detecting appropriate relations and proposed a unique class of algorithms known as messy genetic algorithms. Different versions of messy GAs studied different aspects of BBS by decomposing blackbox search along different dimensions. These investigations have directly influenced the development of SEARCH and the design of the GEMGA. Undoubtedly, the authors of the original version of messy GA (Goldberg, Korb, & Deb, 1989) deserve the credit for first realizing the importance of detecting appropriate relations among the members of the search space.

Section 2 briefly reviews the SEARCH framework. Section 3 discusses the information flow in natural evolution from the SEARCH perspective. Section 4 combines the ideas developed in previous sections and introduces GEMGA. This is followed by Section 5 which presents the test results for large multimodal, order-k delineable problems. Finally, Section ?? concludes

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

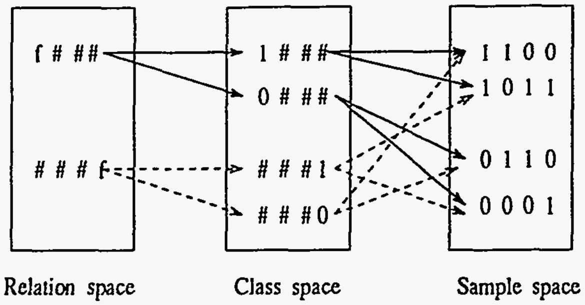

Figure 1: Decomposition of blackbox optimization in SEARCH.

this paper.

2 SEARCH: A Brief Review

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

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

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

A BBS algorithm in SEARCH cannot be efficient if it needs to consider relations that divide the search space in classes, with the total number of classes growing exponentially with the problem dimension. For

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

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

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

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

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

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

This sets the stage for launching an algorithm for solving the k-delineable problems. However, we shall take a detour and first establish the physical validity of the analytical findings in the context of a classical BBS algorithm of nature—the evolution of life on earth. In the following section we shall examine the computational processes in natural evolution and demonstrate that the lessons of SEARCH remains valid and even opens up some new dimensions in the biological context.

3 SEARCH And Natural Evolution

Natural evolution has evolved fitter organisms during the course of time; some species became extinct and some flourished. The development of functionally complex but efficient organisms like human beings has taken place in about 2 billion years. Human genome is comprised of around 2.9×10^8 base pairs, which essentially means that search space is extremely large and it is very unlikely that at the beginning of evolution there existed any prior domain knowledge about this search space. Clearly we evolved in a relatively shorter period of time and that really makes the evolutionary search very impressive.

The objective of this section is to develop a comprehensive perspective toward evolutionary computation using the lessons from the SEARCH framework. In order to do that we first need to establish a computational perspective of gene expression in evolution, largely ignored by the existing computational models of evolution. Once we do that, the arguments of SEARCH can be easily interpreted in the biological context.

Section 3.1 briefly discusses the flow of information in natural evolution. Section 3.2 points out the main problem of the existing models of evolutionary computation—lack of emphasis on gene expression. Section 3.3 raises some additional questions about the natural evolutionary search. Finally, Section 3.4 draws the correspondence between SEARCH and evolution and presents an alternate perspective. Section 3.5 revisits the questions raised in Section 3.3 and proposes some answers. The possibility of constructing richer representation transformation in eukaryotes is pointed out in Section 3.6.

3.1 Information flow in evolution

Information flow in evolution is primarily divided into two kinds:

- **extra-cellular flow**: storage, exploration, and transmission of genetic information from generation to generation;
- **intra-cellular flow**: expression of genetic information within the body of an organism.

Each of these will be discussed in the following two paragraphs.

The extra-cellular flow involves replication, mutation, recombination, and transmission of DNA (deoxyribonucleic acid) from parents to offspring. A DNA molecule consists of two long complementary chains held together by base pairs. DNA consists of four kinds of bases joined to a sugar-phosphate backbone. The four bases in DNA are *adenine* (A), *guanine* (G), *thymine* (T) and *cytosine* (C). Chromosomes are made of DNA double helices. For more detailed description, the reader should refer to Alberts, Bray, Lewis, Raff, Roberts, and Watson (1994, Stryer (1988). *Eukaryotes* (most of the developed organisms) have two chromosomes in their cell nucleus, and thus called *diploid* organisms. On the other hand, in *prokaryotes*, such as single-celled bacteria, only one chromosome is present. These are called *haploid* organisms. The DNA sequence is changed by mutation. Crossing over and subsequent recombination result in exchange of base pairs between the parent DNA sequences. These processes result in generation of new DNA sequences. DNA is then transmitted from the parents to the offspring. The DNA is responsible for defining the phenotype of organism and thereby controls the suitability of the organism in the environment. This suitability determines the selective pressure on the organism. Fitter organisms survive, and the rest do not. However, the computation of the phenotype from the DNA—gene expression—is an interesting process in itself. The following paragraph briefly describes the main steps of gene expression, that define the intra-cellular flow of evolutionary information.

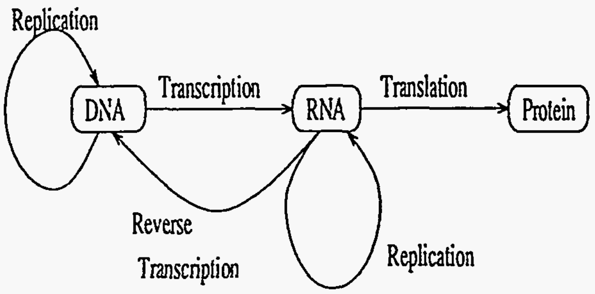

Figure 2: Intra-cellular flow of genetic information.

Expression of genetic information coded in DNA into the proteins is called the *gene expression*. Expression of genetic information takes place through several complicated steps. However, the major distinct phases are identified as

- transcription: formation of mRNA (ribonucleic acid) from DNA
- translation: formation of protein from mRNA

Figure 2 shows the different steps of gene expression. Each of them is briefly described in the following.

Transcription synthesizes messenger RNA (mRNA) from part of the DNA. RNA (ribonucleic acid) consists of four types of bases joined to a ribose-sugar-phosphodiester backbone. Transcription basically constructs a sequence of bases from another sequence of bases—the DNA. Transcription is initiated by some particular sequences of bases in DNA. They are known as *promoter regions*. For example, in many prokaryotes, the *Pribnow box* sequence TATAAT is a common promoter region. Transcription continues until it reaches some particular kind of sequences of bases, known as a *terminator region*. RNA polymerase transcribes the portion of DNA between the promoter and terminator regions. Regulatory proteins of a cell can directly control the transcription of DNA sequences. There are two kinds of regulatory proteins:

- gene activator protein, which enhances transcription of a gene, wherever it binds.
- gene repressor protein, which inhibits transcription of a gene.

These proteins usually bind to specific sequences of DNA and determine whether or not the corresponding gene will be transcribed. Translation synthesizes proteins from the mRNA sequences. Proteins are again sequence of *amino acids*, joined by peptide bonds.

Most of the existing models of evolutionary computation do not provide any understanding about the computational role of the intra-cellular flow of genetic information. The following section gives an account of that.

3.2 A major problem of existing models of evolutionary computation

Unfortunately, many of the existing computational models of evolution address only the extracellular flow of genetic information. Simple genetic algorithms (De Jong, 1975; Goldberg, 1989b; 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.

Although gene expression is not emphasized very much in most of the popular models of evolutionary computation, several researchers realized its importance. The importance of the computational role of gene expression was first realized by Holland. He described (Holland, 1975) the dominance operator as a possible way to model the effect of gene expression in diploid chromosomes. He also noted the importance of the process of protein synthesis from DNA in the computational model of evolution. Despite the fact that traditionally dominance maps are explained from the Mendelian perspective, Holland made an interesting leap by connecting it to the synthesis of protein by gene signals, which today is universally recognized as gene expression. He realized the relation between the dominance operator with the "operon" model of the functioning of the chromosome (Jacob & Monod, 1961) in evolution and pointed out the possible computational role of gene signaling in evolution (Holland, 1975).

Several other efforts have been made to model some aspects of gene expression. Diploidy and dominance have also been used elsewhere (Bagley, 1967; Brindle, 1981; Hollstien, 1971; Rosenberg, 1967; Smith, 1988). Most of them took their inspiration from the Mendelian view of genetics. The underspecification and over-specification decoding operator of messy GA has been viewed as a mechanism similar to gene signaling in Goldberg, Korb, and Deb (1989). Dasgupta and McGregor (1992) proposed the so-called structured genetic algorithm, which uses a structured hierarchical representation in which genes are collectively switched on and off. This implementation also gathered its primary motivation from gene expression. An interesting effort was made by Ackley (1987). He proposed a connectionist paradigm for *iterative genetic hillclimbing* (SIGH) and introduced a relation space through weights. Although, SIGH was not really motivated by gene expression, rather by connectionist computation, the computational objectives of SIGH shared similar philosophy.

Kauffman (1993) offered an interesting perspective of the natural evolution that realizes the importance for gene expression. However, Kauffman's work does not explain the process in basic computational terms on analytical grounds and does not relate the issue to the complexity of search process.

As we see, the computational role of gene expression has mostly been unrecognized. Even when duly recognized, little argument has been made to explain its role in making search efficient. In the coming sections an effort will be made to fill in this lacuna using the lessons from the SEARCH framework in pure computational terms. However, first let us set the premise properly by asking some relevant questions.

3.3 Evolution of life: Some questions

The problems of our existing computational understanding of evolutionary search will become more clear when we ask some hard questions and demand answers in rigorous computational terms. The objective of this section is to do so and demonstrate the need for the alternate perspective of evolution that SEARCH offers.

First in Section 3.3.1 I discuss the issue of "adequate time" (Kauffman, 1993) in evolution and argue that evolution is efficient because it directly searches for relations during *gene expression*. Section 3.3.2 investigates the possible computational role of genetic recombination.

3.3.1 The problem of "adequate time"

The evolution of living organisms, comprised of a large number of mutually interacting components with amazing degree of coordination is undoubtedly impressive. This naturally lead us to think about the time that might have been needed to evolve such organisms from primitive ingredients. Some biologists think that there was enough time for evolution to succeed (Kauffman, 1993) and some of them (Shapiro, 1986; Hoyle & Wickramasinghe, 1981) do not. This is the classical question of "adequate time" in evolution.

Holland (1976) came to an interesting conclusion. Using the so called α-universe model, he argued that emergence of life on earth in such a short time is only possible if evolutionary search could detect the appropriate equivalence classes or schemata. This was an important argument; unfortunately, this line of argument was largely neglected by the biologists and the debate continued primarily because of the lack of analytical result supporting Holland's argument.

Those who believe in inadequate time theory, depend on the observation that the search space is too large to deal with by random enumeration. Two billion years may be a "long time" compared to our lifetime, it may not be quite so compared to the size of the evolutionary search space. Wald (1954) conjectured that 2 billion years of time is sufficient and sampling different organisms during the course of this "long" time made evolution successful. He even concluded "Time is in fact the hero of the plot". Shapiro (1986) criticized Wald's perspective and presented convincing argument demonstrating that there was not sufficient time for evolution to succeed. Shapiro estimated the total number of samples that evolution could have taken during the last two billion years. He then computed a conservative bound on the joint probability of finding the set of functional enzymes of a primitive bacterium from this set of samples and showed that success probability is extremely low. The joint success probability is so low that it has been compared elsewhere Hoyle and Wickramasinghe (1981) with the chance of "a tornado sweeping through a junk yard might assemble a Boeing 747 from the materials therein". The foundation of this line of argument is based on the assumption that evolution searches by random enumeration. Once we accept this premise their argument makes sense.

Goldberg (1989a) indirectly addressed this question on computational grounds following Holland's idea of schema processing. Although his arguments were primarily directed toward computational limitations of evolutionary algorithms such as GAs, their implications on biological context were equally important. He introduced (Goldberg, 1987) order-k deceptive functions which essentially admitted that a blackbox search algorithm can only efficiently solve problems with certain degree of decomposability.

Kauffman (1993) offered a different way of answering this question. He argued against the idea of computing joint success probability for all the different enzymes of living beings. He writes "We should instead be concerned with the probability of finding any one of possibly very many properly coupled sets of enzymatic activities which might constitute a living proto-organism". He proposes that the development of the individual components in the RNA and protein spaces lead to the emergence of the whole in a time shorter than that corresponding to the joint probability computed by Shapiro (1986). This is an interesting break. Although Kauffman presents his arguments in terms of phase transitions, autocatalysis, and percolation principles, in my opinion his arguments against computing the joint probability make sense only when the protein space is decomposable. Although the search problem in DNA space may not be decomposable, the DNA→RNA→Protein transformation may convert the problem into a decomposable one.

As we see, the arguments in favor of adequate time are three folds: (1) Holland's idea of equivalence class processing (2) Goldberg's argument about problem decomposability, and (3) Kauffman's emphasis on gene expression. However, none of them alone describes the complete picture about the computational processes in natural evolution. In the coming sections we shall put them together in the light of SEARCH to offer a more complete picture of the efficiency in evolutionary search. Before that, let us consider another important factor in evolution—the natural selection and see whether or not the existing models of evolutionary computation does capture the complete picture.

3.3.2 Natural selection: Some questions

The role of natural selection in evolution is almost universally acknowledged. Natural selection has been identified as one of the main factor defining the evolution and self-organization in many complex systems.

An immediate question that may come to our mind is—What does natural selection select? Clearly living organisms have DNA space and the protein space. The DNA sequence defines the set of proteins in an organism. The proteins are in turn responsible for the phenotypic features of the organism. The performance of an organism in its environment may act as an index of the selective pressure. However, the question is how does the selective pressure effect the organism? As we know, both DNA space and the gene regulatory mechanisms evolved during the course of evolution (Alberts, Bray, Lewis, Raff, Roberts, & Watson, 1994). In order to take place that, there must be some distribution of selective pressure in each of these spaces.

Unfortunately, existing evolutionary algorithms do not consider the apportionment of selective pressure in these two different spaces. As we saw earlier, evolutionary search algorithms remains contend with selection in the sample space, corresponding to the effect of natural selection in the DNA space. Clearly, the lack of consideration of the selective pressure in gene expression is a missing feature from the modeling perspective. The question is whether it affects even the computational modeling of evolutionary search? The answer is yes. However, let us again resist ourselves from explaining the answer until we discuss another puzzle of natural evolution that appears from the role of genetic recombination and crossing-over.

3.3.3 Recombination of what?

Recombination among homologous pair of chromosomes results in exchanging set of genes among the parent chromosomes and produces offspring with new chromosomes. A good deal of controversy exists about the utility of recombination. In fact, the field of evolutionary computation appears to be divided into two camps one supporting the utility of recombination and other dismissing that. The basic question that we need to ask first is that recombine what? If we consider the parent chromosome together as a tuple, then recombination is nothing but a permutation operator among the 2ℓ genes. There are $(2\ell)!$ ways to permute that tuple of 2ℓ genes. Therefore, searching using recombination is no more efficient than mutative search.

However, recombination is good if we know what to exchange. If we know what relations are good then we only need to exchange the classes that belong to those relations. In an order-k delineable representation recombination can be used to combine the classes to produce classes of higher order relations.

In natural evolution, recombination process is controlled by different proteins. For example in *E. coli*, recombination is mediated by products of *rec* genes (Stryer, 1988). After the single-stranded DNA is created by *recBCD* protein, the *recA* protein directly controls the process of binding the duplex DNA, base pairing, and the exchange of strands. Clearly the working of recombination depends on these proteins and recombination will reduce to be a random permutation operator in absence of these proteins. Therefore, the evolution of right proteins appears to be important for the efficient working of recombination.

Unfortunately, most of the existing evolutionary algorithms do not recognize this. One and multi-point crossover (De Jong, 1975; Holland, 1975), uniform crossover (Syswerda, 1989) are some examples of artificial crossovers widely used in genetic algorithms. In one and multi-point crossovers the point of crossing-overs are randomly chosen. In uniform crossover individual gene swapping is decided randomly. Clearly none of them has any controlling feature. An interesting efforts was made elsewhere (Schaffer & Morishima, 1987). They suggested the use of adaptive crossover that gradually biases toward better classes. Maini, Mehrotra, Mohan, and Ranka (1994) proposed using domain knowledge-based nonuniform crossover. Other efforts on adaptive crossovers can be found elsewhere (Jog, Suh, & Van Gucht, 1989; White & Oppacher, 1994).

3.4 Evolutionary computation: The SEARCH perspective

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

- **Sample space:** DNA constitute the sample space. Crossover and mutation generate new samples of DNA. A population of organisms defines the sample space for the evolutionary search.
- **Class space:** Base sequences of mRNA transcribed in a cell correspond to only a part of the complete DNA. The sequence of amino acids in protein in turn correspond to base sequence in mRNA. The genetic code tells us that there is a unique relationship between the nucleotide triplets of the DNA and the amino acids in the protein. Therefore, if we consider the DNA as a representation defined over the evolutionary search space for life and different forms of life, then the amino acid sequence of a protein corresponds to a class of different DNA; every DNA in this class must have a certain sequence of nucleotides that can be transcribed to that particular sequence of amino acids. Since the genetic code is unique, a particular sequence of amino acids can only be produced by a certain sequence of nucleotides. In other words, the sequence of amino acids in a protein defines an equivalence class over the DNA space.
- **Relation space:** Recall that amino acid sequences in protein are translated from the nucleotide sequences of mRNA. The construction of mRNA is basically controlled by the transcription process. Since an equivalence relation is an entity that defines the equivalence classes, the transcription regulatory mechanism can be viewed as the relation space that defines classes in terms of the nucleotide sequences in mRNA and finally in terms of the amino acid sequences in proteins. Among the different components of this regulatory mechanism, regulatory proteins, promoter and terminator regions play a major role. Regulatory proteins exist as a separate entity from the DNA, but the promoter and terminator regions are defined on the DNA. It appears that

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

> there is a distinct relation space comprised of the different regulatory agents, such as activator and inhibitor proteins. However, it is quite interesting to note that this space also directly makes use of information from the sample space—the DNA. Expression of genetic information in eukaryotic organisms is more interesting than that in prokaryotes.

These possible relationships between the different spaces of SEARCH and natural evolution are summarized in Table 1. Now that, we have drawn a correspondence between the different components of natural evolution and the SEARCH framework, we are ready for answering the questions raised earlier in section 3.3.

3.5 Evolution of life: Some answers

In this section we shall revisit the questions raised in section 3.3 in the light of SEARCH and present some possible explanations.

3.5.1 The issue of adequate time

As we saw earlier, the arguments favoring the adequate time theory are three folds: (1) Holland's idea of equivalence class processing (2) Goldberg's argument about problem decomposability, and (3) Kauffman's argument favoring the importance on evolution in the protein space. Now if we look at the adequate time problem and these arguments in the light of SEARCH we can come to an interesting conclusion—all of them are correct when we put them together. When we do so, the hypothesis appears as follows: *evolution can be successful in such a short period of time if and only if it searches for appropriate equivalence classes defined by the representation and the search problem was either originally decomposable or transformed to a decomposable one in the protein space.* The following part of this section corroborates this hypothesis on the analytical grounds offered by SEARCH.

First of all, SEARCH proved that polynomial complexity blackbox search is not possible unless some relations among the members of the search space is exploited. Relations define classes and exploiting relations requires processing classes. Therefore, evolutionary search cannot be of polynomial complexity (i.e. efficient) unless it processes classes defined over the genetic representation. The second point is about decomposability. Was the evolutionary search problem decomposable in the initial genetic representation. No one knows, but it is unlikely. The genetic representation and the expression of genetic information (a representational transformation) evolved during the course of evolution. If the evolutionary search landscape were really decomposable and solvable in an efficient manner even at the early stage, such evolution and transformation of representation was not required. It seems that nature had to search for an appropriate transformation which expressed the genetic expression in such a way that the search problem becomes decomposable at a certain level. The need for problem decomposability and the possible mechanism of gene expression for accomplishing such decomposability can again be corroborated using SEARCH. The SEARCH framework proved that a blackbox search algorithm can only solve problems that need considerations of relations up to a bounded order—the class of order-k delineable problems. In other words there must be some degree of decomposability in the relation space. If the given relation space is not order-em k delineable, the relation space, Ψ_r must be transformed to introduce delineability. Evolutionary search in nature uses a sequence representation. DNA sequence defines the primary representation. Expression of this information using the DNA→RNA→Protein defines a new relation space. Searching for appropriate regulatory mechanism can be viewed as the search for the right relation space that makes the problem order-k delineable. Clearly all the three components of hypothesis supporting adequate time theory can be put in proper perspective in the light of the analytical foundation offered by SEARCH.

The following section revisits the issue regarding the computational need for accounting the effect of natural selection in the DNA, protein, and the regulatory mechanism spaces.

3.5.2 Apportionment of selection pressure

Section 3.4 identified the DNA space as the sample space, the protein space as the explicit class space and the gene regulatory control mechanisms as the relation space. SEARCH clearly points out that no algorithm can surpass the limits of random enumerative search if it guides itself by applying selection in the sample space. Therefore, evolutionary search in nature cannot surpass this computational limit by simply applying the selective pressure in the DNA space. Effect of natural selection must also be distributed in the relation and class spaces of evolutionary search. In other words the effect of natural selection must show up in the evaluation of parts of DNA sequences into proteins in different cells of an organisms and also the gene regulatory mechanism space.

The following section revisits the computational aspects of genetic recombination in the light of SEARCH.

3.5.3 Recombination of classes

From the SEARCH perspective, recombination serves the purpose of *resolution*. Once the right relations are detected the corresponding better classes can be *resolved* using a recombination like operator. Therefore, the purpose of recombination in natural evolution is not at all clear unless we introduce the relations as possible controlling agents.

The following section describes an interesting possibility—construction of new representation in natural evolution.

3.6 Representation construction

Evolution of gene regulatory mechanism means construction of new representation. Eukaryotic organisms have a richer way to construct new representation. Most of the eukaryotic organisms are diploid. At a particular gene one allele is recessive and the other is dominant. The expression of a dominant gene takes place during transcription and translation. When a diploid chromosome is viewed as a sequence of dominant and recessive tuples, the set of dominant alleles can be interpreted as a new representation for the set of recessive alleles. The gene regulatory control mechanism determines what gets expressed in a particular cell. The evolution of this regulatory control mechanism is computationally equivalent to construction of new relation space. There is existing biological evidence that these settings for the intra-cellular expression of genetic information evolved during the course of evolution (Alberts, Bray, Lewis, Raff, Roberts, & Watson, 1994). As noted in SEARCH, such transformation is needed when the original relation space is not order-k delineable. Therefore, one of the reason that eukaryotic organisms became more successful in the evolutionary race could be the ability to construct new representation. On the other hand prokaryotes are primarily haploid (i.e. one chromosome only) and are deprived of this capability. The following section presents the GEMGA. The following section identifies an one to one correspondence between different components of natural evolution and SEARCH.

4 The Gene Expression Messy GA

In the earlier sections of this paper, we noted the essential ingradients of efficient, general BBS algorithms and identified a class of BBS problems that can be solved efficiently. We also observed these conclusions in the light of natural evolutionary search. Now it is the time to put them together and propose a realization of the theoretical observations along with biological plausibility.

In this section I introduce Gene Expression Messy GA (GEMGA)—an $O(|\Lambda|^k(\ell+k))$ sample complexity algorithm for order-k delineable problems in sequence representation of length ℓ and alphabet Λ. Design of GEMGA is based on the alternate perspective of evolution, developed by SEARCH that emphasize the computational role of gene expression.

Section 4.1 discusses the representation in GEMGA. Section 4.2 explains the population sizing in GEMGA. This is followed by Section 4.3 that describes the main operators, transcription, selection, and recombination. Section 4.4 presents of the overall mechanisms.

4.1 Representation

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

4.2 Population sizing

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

4.3 Operators

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

4.3.1 Transcription

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

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

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

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

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

  for(i=0; i<Problem_length; i++) {
    if(chrom1[i].Weight() >
        chrom2[i].Weight() )
      chrom2[i] = chrom1[i];
    else if(chrom2[i].Weight() >
        chrom1[i].Weight() )
      chrom1[i] = chrom2[i];
  }
}
```

Figure 4: 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.

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.

4.3.2 Selection

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

- **Class Selection:** The class selection operator is responsible for selecting individual classes from the chromosomes. Better classes detected by the transcription operator are explicitly chosen and given more copies at the expense of bad classes in other chromosomes. Figure 4 describes the operator. Two chromosomes are randomly picked; the weights of the genes are compared and the gene with higher weight overwrites the corresponding gene in other chromosome with lower weight.

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

The following section describes the recombination operator in GEMGA.

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

  for(i=0; i<Problem_length; i++) {
    if(chrom1[i].Weight() >=
        chrom2[i].Weight()
        AND Random() < pcg) {
      dummy = chrom1[i];
      chrom1[i] = chrom2[i];
      chrom2[i] = dummy;
    }
  }
}
```

Figure 5: 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. `Random()` generates a random number in between 0 and 1; `pcg` is a number between 0 and 1.

4.3.3 Recombination

Figure 5 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. If the weight of a gene from the marked chromosome is greater than that of the corresponding gene from the other chromosome then it swaps the genes.

The following section describes the overall mechanism of the algorithm.

4.4 The Algorithm

GEMGA has two distinct phases: (1) primordial stage and (2) juxtapositional stage. The primordial stage simply applies transcription operator for ℓ generations, deterministically considering every gene in each generation. During this stage the population of chromosomes remains unchanged, except that the weights of the genes change. This is followed by the juxtapositional stage, in which the selection and recombination operators are applied iteratively. Figure 6 shows the overall algorithm. The length of the juxtapositional stage can be roughly estimated as follows. If t be the total number of generations in juxtapositional stage, then for binary tournament selection, every chromosome of the population will converge to same

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

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

Figure 6: Pseudo-code of GEMGA. The constant C ¡ $|\Lambda|$, where $|\Lambda|$ is the cardinality of the alphabet set.

instance of classes when $2^t = n$, i.e. $t = \log n / \log 2$. Substituting $n = c|\Lambda|^k$, we get $t = \frac{\log c + k \log |\Lambda|}{\log 2}$. A constant factor of t is recommended for actual practice. Clearly the number of generations in juxtapositional stage is $O(k)$. Let us now compute the overall sample complexity of GEMGA. Since the population size is $O(|\Lambda|^k)$ and the primordial stage continues for $C\ell = O(\ell)$ generations, the overall sample complexity,

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

GEMGA is a direct realization of the lessons from the SEARCH framework. Following SEARCH, it can be recognized that the sample complexity is also a function of the desired quality of the solution and the reliability of the process. However, the implementation of GEMGA through distributed local evaluation of relations and classes outweighs the satisfaction of quantifying the success probability that is straight forward in case of centralized comparison (as it was in SEARCH) from the practical perspective. Therefore, the reader must realize the dependence of the sample complexity on the desired accuracy of the solution and reliability, implicit in the above arguments. The following section 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 multimodality, bounded inappropriateness of relation space (BIRS), problem size, scaling, noise. The GEMGA has been tested against problems with all of these dimensions of difficulties (Kargupta, 1996). However, because of limited space, in this section, we present the performance of GEMGA for problems with only massive multimodality and controlled amount of BIRS.

For all functions the average number of function evaluations per success (AFPS) is measured. For every function we choose the desired solution value (DSV) a priori. We say the algorithm was successfull if it reaches the DSV.

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

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

where u is the number of 1-s in the string x and ℓ' is the length of the sequence used for representing the variable x. Goldberg, Deb, and Clark (1992) showed that such deceptive problems can be used to design problems of bounded difficulty. In a trap function defined over a sequence of length ℓ' the order of delineability is ℓ' with respect to the class average comparison statistics. Although GEMGA does not work using the class average comparison statistic (i.e. when classes are compared with respect to the distribution means) this gives us a simple way to capture the main essence. When multiple number of such functions are concatenated with each other a problem defined over a sequence of length ℓ with order-ℓ' delineability can be designed. Since the order of delineability directly controls the BIRS, such concatenated functions can be effectively used for designing problems with BIRS by controlling the ℓ'. Such functions have only ℓ/ℓ' proper relations among the $\binom{\ell}{\ell'}$ order-5 relations that must be detected in order to find the global solution. Therefore, searching for the appropriate relations is not a

Recombination probability	0.0
Gene exchange probability (pcg)	0.0
Class selection probability	1.0
String selection probability	0.0

Table 2: GEMGA parameters for TF1.

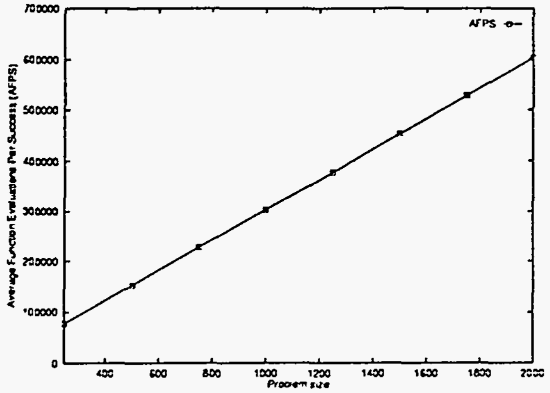

Figure 7: Growth of the number of function evaluations with problem size. The population size for all problem sizes was 300. All the results are average of *five independent runs.*

trivial job in this class of problems. Apart from BIRS, such functions also offer multimodality. If we carefully observe, we shall note that a trap function has two peaks. One of them corresponds to the string with all 1-s and the other is the string with all 0-s. If we design a problem by concatenating m such functions, it *will have a total of* 2^m *local optima and among them* only one will be the globally optimal solution.Clearly this class of problems are massively multimodal and has bounded inappropriateness of the relation space, defined by the representation.

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

The GEMGA is tested against order-5 deceptive problems of different sizes. Table 2 shows the GEMGA parameters used for all of them. Figure 7 shows the average number of function evaluations from five independent runs needed to find the DSV for different problem sizes. For all problems, the DSV is set to the globally optimal solution, which is equal to problem size; ℓ. The population size is chosen as described earlier in this paper. The chosen population size for all the problems was 300. The sample complexity clearly grows linearly and the population size is constant.

Figure 8 and 9 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 real valued 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. Figure 8 show that these relations are gradually detected in different chromosomes that contain good classes from those relations. Finally, at the end of primordial stage (Figure 9 (top)) all the relations are detected. Figure 9(middle), (bottom) shows the processing of classes during juxtapositional stage. More instances of good classes are produced by selection and they are exchanged among diffrent strings to create higher order relations that finally lead to the optimal solution. The following section concludes this paper.

6 Conclusion

This paper makes an effort to design BBS algorithms in a constructive manner following the lessons of SEARCH. It identified different decision makings in BBS and realized the class of problems efficiently. After verifying the arguments in the the light of natural evolution, the GEMGA is introduced. GEMGA does not construct new representation, although it is one among the immediate future possibilities. If the problem is order-k delineable with respect to the representation and class comparison statistic GEMGA will solve the problem in polynomial sample complexity. Test results for large problems with milions of local optima and bounded inappropriateness of the representation confirms this conclusion.

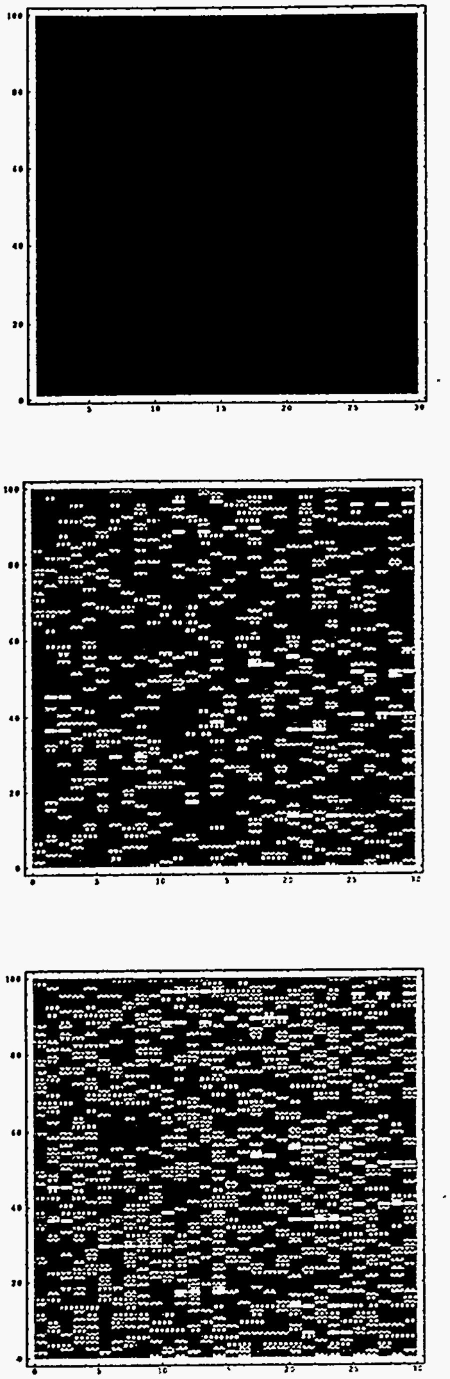

Figure 8: The relation space during primordial geneation 1 (top), 10 (middle), and 20 (bottom).

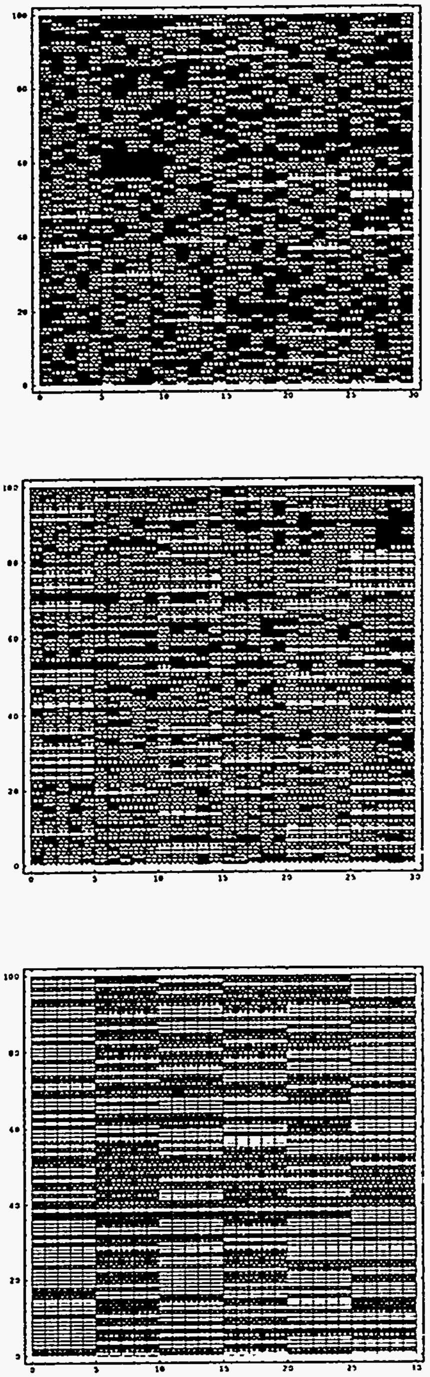

Figure 9: The relation space at the end of primordial geneation (top), juxtapositional generations 1 (middle) and 4 (bottom).

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

7 Acknowledgement

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, Kakali Sarkar, and Georges Harik.

References

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

Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., & Watson, J. D. (1994). *Molecular biology of the cell.* New York: Garland Publishing Inc.

Bagley, J. D. (1967). The behavior of adaptive systems which employ genetic and correlation algorithms. *Dissertation Abstracts International*, *28*(12), 5106B. (University Microfilms No. 68-7556).

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

Dasgupta, D., & McGregor, D. R. (1992). Designing neural networks using the structured genetic algorithm. *Artifical Neural Networks*, *2*, 263–268.

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.

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. In Davis, L. (Ed.), *Genetic Algorithms and Simulated Annealing* (pp. 74–88). San Mateo, CA: Morgan Kaufmann. (Also TCGA Report 86003).

Goldberg, D. E. (1989a). Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems*, *3*(2), 129–152. (Also TCGA Report 88006).

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

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

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

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

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

Holland, J. H. (1976). Studies of the spontaneous emergence of self-replicating systems using cellular automata and formal grammars. In Lindenmayer, A., & Rozenberg, G. (Eds.), *Automata, Languages, Development* (pp. 385–404). New York: North-Holland.

Hollstien, R. B. (1971). Artificial genetic adaptation in computer control systems. *Dissertation Abstracts International*, *32*(3), 1510B. (University Microfilms No. 71-23,773).

Hoyle, F., & Wickramasinghe, N. C. (1981). *Evolution from space.* London: Dent.

Jacob, F., & Monod, J. (1961). Genetic regulatory mechanisms in the synthesis of proteins. *Molecular Biology*, *3*, 318–356.

Jog, P., Suh, J. Y., & Van Gucht, D. (1989). The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. In Schaffer,

J. D. (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 110–115).

Kargupta, H. (1995a, 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. (1995b). Signal-to-noise, crosstalk and long range problem difficulty in genetic algorithms. In Eshelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 193–200). San Mateo, CA: Morgan Kaufmann.

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

Kauffman, S. (1993). *The origins of order.* New York: Oxford University Press.

Maini, H., Mehrotra, K., Mohan, C., & Ranka, S. (1994). Knowledge-based nonuniform crossover. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, Volume 1 (pp. 22–27). Piscataway, NJ: IEEE Service Center.

Michalski, R. S. (1983). Theory and methodology of inductive learning. In Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.), *Machine learning: An artificial intelligence approach* (pp. 323–348). Tioga Publishing Co.

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

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

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

Rosenberg, R. S. (1967). Simulation of genetic populations with biochemical properties. *Dissertation Abstracts International*, *28*(7). 2732B. (University Microfilms No. 67-17,836).

Schaffer, J. D., & Morishima, A. (1987). An adaptive crossover distribution mechanism for genetic algorithms. In Grefenstette, J. J. (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 36–40).

Shapiro, R. (1986). *Origins: A skeptic's guide to the creation of life.* New York: Summit Books.

Smith, R. E. (1988). *An investigation of diploid genetic algorithms for adaptive search of nonstationary functions* (TCGA Report No. 88001). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.

Stryer, L. (1988). *Biochemistry.* New York: W. H. Freeman Co.

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).

Wald, G. (1954, August). The origin of life. *Scientific American*.

Watanabe, S. (1969). *Knowing and guessing - A formal and quantative study.* New York: John Wiley & Sons, Inc.

White, T., & Oppacher, F. (1994). Adaptive crossover using automata. In Davidor, Y., Schwefel, H.-P., & Männer, R. (Eds.), *Parallel Problem Solving from Nature- PPSN III* (pp. 229–238). Berlin: Springer-Verlag.