Linkage Learning via Probabilistic Modeling in the ECGA

Georges Harik

IlliGAL Technical Report 99010
January 1999

Illinois Genetic Algorithms Laboratory
Department of General Engineering
117 Transportation Building
104 S. Mathews Ave
Urbana, IL 61801
Phone: (217) 333-2346
Fax: (217) 244-5705

Linkage Learning via Probabilistic Modeling in the ECGA

Georges Harik
gharik@earthlink.net

ABSTRACT

The goal of linkage learning, or building block identification, is the creation of a more effective genetic algorithm (GA). This paper explores the relationship between the linkage-learning problem and that of learning probability distributions over multi-variate spaces. Herein, it is argued that these problems are equivalent. Using a simple but effective approach to learning distributions, and by implication linkage, this paper reveals the existence of GA-like algorithms that are potentially orders of magnitude faster and more accurate than the simple GA.

I. Introduction

Linkage learning in genetic algorithms (GAs) is the identification of building blocks to be conserved under crossover. Theoretical studies have shown that if an effective linkage-learning GA were developed, it would hold significant advantages over the simple GA (2). Therefore, the task of developing such an algorithm has drawn significant attention. Past approaches to developing such an algorithm have focused on evolving a problem's chromosomal representation along with its solution (3, 4, 5). This has proven to be a difficult undertaking. This paper reinterprets and solves the linkage-learning problem in the context of probabilistic optimization.

Recently, a number of algorithms have been developed that replace the GA's population and crossover operator with a probabilistic representation and generation method (6, 8, 9, 11). Studies have shown a close correspondence between these algorithms and equivalent simple GAs (4). This paper shows how a variant of these algorithms, that pays close attention to the probabilistic modeling of the population successfully tackles the linkage-learning problem.

We will begin by briefly reviewing the workings of the simple GA, as well as those of the related probability-based algorithms. We will then explore the close relationship between these two approaches to optimization. The remainder of this paper is then concerned with the ultimate consequences of this relationship. The argument to be presented will consist of two separate assertions:

- That learning a 'good' probability distribution is equivalent to learning linkage.
- That one 'good' distribution can be found by searching for a jointly small representation of two components: 1) the compressed representation of the

population under the given distribution and 2) the distribution's representation given the problem encoding.

Ultimately, this argument must stand on the legs of empirical observations, as in its essence it is but the application of Occam's Razor. The last part of this paper presents a probabilistic algorithm, the extended compact GA (ECGA), designed to learn linkage through learning good probability distributions. It then demonstrates the advantage that this approach provides over the simple GA, on a theoretical problem that has traditionally been used to test other linkage-learning approaches. Finally, this paper explores the consequences of the proposed probabilistic algorithm.

II. The Simplified Simple GA

A GA (12, 13) is a simulation of the genetic state of a population of individuals --- their genetic state being their combined chromosomes. It typically includes those forces of genetics deemed most influential in nature, such as natural selection, mutation, and crossover (mating). In this paper, we will restrict ourselves to one facet of the GA: its use as a problem solver, or optimization algorithm. Natural evolution typically leads to a set of individuals that are well suited to their environment. By controlling the computational nature of such an environment, the GA can be made to evolve chromosomes (structures) that are well suited to any given task.

An optimization is a search over a set of structures, to find the "best" structure under some given criteria. This paradigm maps over readily to implementation in a GA. Each structure is represented by its blueprint, or chromosome, in the GA's population. The GA's population is thus the current set of structures the algorithm has found to be most interesting or useful. At each point in time, it represents the current 'state' of the search. The genetic operators of natural selection, crossover, and mutation then generate the next state of the search from the current one. The GA's goal is to reach a final state (population) that contains a good solution (structure) to the problem at hand.

In order to simplify this exposition, we will assume that the structures the GA will optimize over are the set of binary strings of fixed length L. A binary string is simply a consecutive sequence of characters each of which is a 0 or a 1. This restriction makes it easier to visualize and understand the GA and its operators. However, the theory developed in this paper will remain applicable to the much wider domain of optimization over finite-dimensional spaces. We will for the same reason also consider only selecto-recombinative GAs, thus ignoring for the moment the effects of mutation.

In the course of optimization, the GA's population repeatedly undergoes processing by the two genetic operators of crossover and selection, until it converges. Convergence here means that only one type of chromosome remains in the population --- hopefully the best or a good solution. The two operators have orthogonal goals. The crossover operator generates new chromosomes by mixing parts from other pairs of chromosomes. It roughly corresponds to mating and reproduction in nature.

The selection operator weeds out those chromosomes that are unsuited to their environment --- that is, those that have a poor score under the current optimization. Again, for the purpose of simplicity, we focus on block selection (14) and uniform crossover (15) as representatives of possibly more general selection and crossover operators.

In block selection, a large fraction of the weakest chromosomes in the population are thrown out, and the stronger chromosomes are given their place. Strength here is measured according to the chosen optimization problem. Operationally, the optimization problem is represented by a *fitness function* that maps structures over to real numbers. The strongest structures are then those with the highest fitness score. Block selection is controlled by one parameter, *S*, which specifies that only the best fraction 1/*S* of the population is to be retained after the action of selection. Figure 1 shows the effects of selection with *S*=2 on a population of size 8.

Figure 1: With S=2, each chromosome in the top half of the population gets 2 copies in the next generation.

Under uniform crossover, the population is paired up, and each pair of chromosomes generates two children, which replace their parents in the population. A child is created from the two parents by randomly inheriting the value of each position (dimension) from one of its two parents, while its sibling gets the value at that position from the other parent. In the parlance of GAs, each (position, value) combination is referred to as a *gene*. Figure 2 shows the possible effects of uniform crossover on two very dissimilar chromosomes.

11111111	10011000
00000000	01100111
Parents before crossover	Children after crossover

Figure 2: Note how all the genes are conserved in a crossover. Each parental gene ends up in one of the two children.

Optimization by selecto-recombinative GAs thus consists of a random initialization of the population, followed by repeated applications of crossover and selection. This optimization is typically stopped when the population has converged, although a number of other stopping criteria are also possible. Figure 3 shows one particular population that has converged to the value 01110111. In a number of problems, GAs have been shown to consistently outperform standard optimization techniques. The reason why the GA does well is widely agreed to be a consequence of its effective propagation of sub-structures that are correlated with high fitnesses.

01110111
01110111
01110111
01110111
01110111
01110111
01110111
01110111

Figure 3: A converged population only has one type of chromosome in it, and can therefore not search for more structures.

As an example, let us consider simple optimization problem of maximizing 1s (one-max), where the fitness of each string is the number of 1s it contains. Figure 4 shows the possible evolution of a population under one generation of selection and crossover. Note how the ratio of 1s in the new population is higher than in the old, and that the 1s are well distributed. This is because selection increases the number of 1s, and crossover mixes them together in an attempt to combine all the 1s into a single chromosome. In this case, each 1 gene is correlated with a high fitness, and the GA has successfully exploited this information.

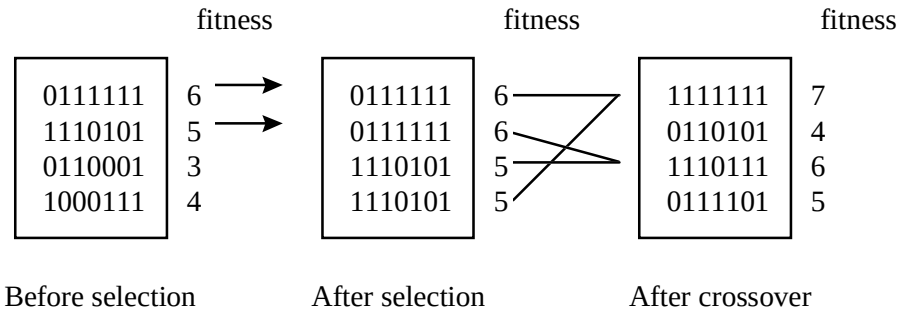

Figure 4: Selection followed by crossover leads to a new chromosome (the first) that is more fit (7) than any previous one.

Although the reason for the GA's success is widely agreed upon, the nature of the structures it exchanges, and whose correlation with fitness it maintains, is under vigorous debate. The crux of the problem is the complexity of these structures. Two mutually exclusive possibilities vie for the explanation of the GA's success: 1) that single genes are the only structures that the GA can effectively deal with; and 2) that the GA can process more complex structures, consisting of several genes, which are referred to as *building blocks*.

The study of linkage learning is the study of how to make the GA process structures more complex than single genes, in the absence of information about which genes are related. In this context, related roughly means that the genes would have to exist in tandem to provide a fitness boost, but each gene alone would not provide a detectable fitness signal. As of yet, the most advanced of such methods have only been marginally successful in justifying the computational effort necessary to undertake them (4). The remainder of this paper addresses this issue by developing a computationally justifiable algorithm that learns linkage. First, however, we take a necessary detour to explore a set of probabilistic algorithms that are closely related to the GA.

III. Order-1 Probabilistic Optimization Algorithms

The population of the GA represents information about the parts of the search space that the GA has seen before. The crossover and selection operators tell the GA how to exploit this information to generate new, and potentially good, solutions. Along the course of time, researchers noticed that crossover tended to decorrelate the individual dimensions (or genes) in the solution structures, while selection tended to change the makeup of the population by rewarding the more successful genes. Thus were born a number of algorithms that replaced the population, crossover, and selection with a number of actions on marginal probability distributions on each of the representation's genes.

The idea behind these algorithms rested on representing the current state of the search as the fraction of each dimension (or gene) in the population that had a value of 1. Using only this information, a new population could be generated that mimicked the effect of many consecutive crossovers. By altering these probabilities according to how well

certain genes did against the competition, these algorithms could also mimic the effect of selection. The compact GA (cGA) (9) and PBIL (8) are two examples of these simplistic (but effective) algorithms. We will restrict ourselves here to looking at the cGA as it is slightly simpler than PBIL.

The cGA begins by initializing an *L*-dimensional probability vector *P*[] (one for each gene position) to 0.5. This phase corresponds to the random initialization phase of the simple GA. *S* solutions are then generated by polling this vector, i.e., selecting the *K*th dimension (or gene) to be 1 if a unit uniform random variable was less than the *K*th dimention of the probability vector, *P*[*K*]. The gene positions of the fittest of these *S* solutions are rewarded in pairwise competitions with each of the less fit solutions. *P*[*K*] is increased if the fittest has a 1 in the *K*th position, and the less fit solution does not. *P*[*K*] is likewise decreased if the fittest has a 0 in the *K*th gene, and the less fit solution does not. The amount of increase or decrease in parametrized by a value *E*.

For instance, take the maximizing 1s problem, and assume *L*=4, *S*=2 and *E* = 0.25. Figure 5 shows one evaluation taking place under this algorithm. Of the two generated chromosomes 0111 (with a fitness of 3) is fitter than 1010 (with a fitness of 2). The original probability vector is random, *P*[] = [0.5 0.5 0.5 0.5]. In the first gene, the 0 is part of a fitter chromosome than the 1. Therefore *P*[0] is decreased by 0.25. Note that the index of the first gene in the vector is taken to be 0, not 1 due to a programming convention. In the second gene, the opposite is true, therefore *P*[1] is increased by 0.25. The third gene is the same in both chromosomes, so *P*[2] is unchanged. *P*[3] is again increased by 0.25. This leaves us with the new probability vector *P*[] = [0.25 0.75 0.5 0.75]. This process continues until the P[] vector implies a single solution, that is all its values are zeroes or ones. At this point, the cGA has converged.

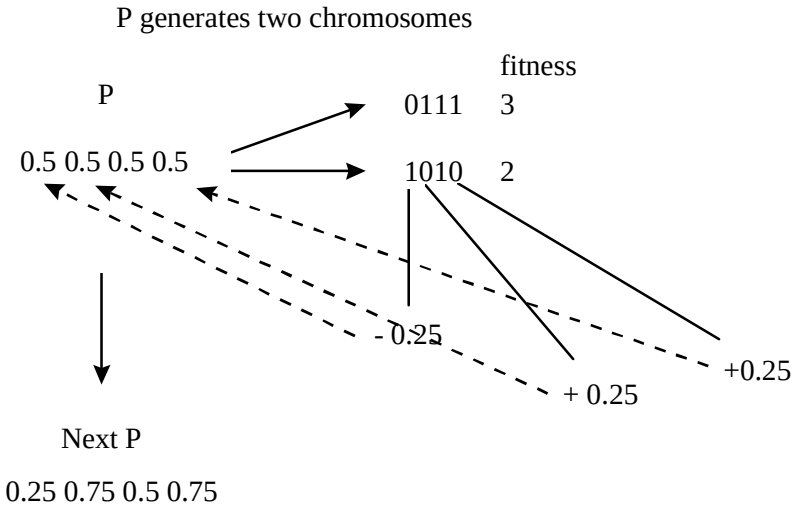

Figure 5: The cGA evaluation step consits of generation, followed by an examination that changes the probability distribution.

One might see a close correlation between this algorithm and the simple GA, but still find it difficult to guess at the extent of this relationship. It has been shown that the simple GA using a population size N and a selection rate of S under tournament selection (which is a close cousin to block selection), and uniform crossover, can be mimicked very closely by the cGA generating S solutions and using $E = 1/N$. These algorithms are referred to as order-1 probabilistic algorithms as they maintain the population's distribution as a product of the marginal distributions of each of the separate genes; genes being considered order-1 or trivial building blocks.

IV. Probabilistic Optimization and Linkage Learning

The correspondence between the operation of cGA and the simple GA hints at a deeper connection. This connection is that the GA's population can be interpreted as representing a probability distribution over the set of future solutions to be explored. The GA's population consists of chromosomes that have been favored by evolution and are thus in some sense good. The distribution that this population represents tells the algorithm where to find other good solutions.

In that sense, the role of crossover is to generate new chromosomes that are very much like the ones found in the current population. This role can also be played by a more direct representation of the distribution itself. This is precisely what the cGA and PBIL do. Similarly, changes in the makeup of the population due to selection can be reflected in alterations of the probability distribution itself.

The probability distribution chosen to model the population can be crucial to the algorithm's success. In fact**, the choice of a good distribution is equivalent to linkage learning**. We take a moment now to explore this statement in the context of a problem that is difficult for the simple GA to solve without proper linkage learning.

IV.1 Linkage Learning and Deceptive Problems

We begin by defining a "deceptive" version of the counting ones problem. Here the fitness of a string is the number of 1s it contains, unless it is all 0s, in which case its fitness is L+1 (L recall is the problem length). The reason this is called a deceptive problem is that the GA gets rewarded incrementally for each 1 it adds to the problem, but the best solution consists of all 0s.

The initial conception of this problem is a needle in a haystack, which no optimization algorithm can be reasonably expected to solve. To transform it into one that requires linkage learning, we combine multiple copies of deceptive subproblems into one larger problem. For example, a 40 dimensional problem can be formed by grouping together each 4 dimensions into a deceptive subproblem. This problem will thus utilize 10 of the deceptive subproblems defined above. The fitness of a string will be the sum of the subproblem fitnesses, where each subproblem is defined over a seperate group of 4 dimensions. Figure 6 shows how a sample 40 bit string is evaluated in this problem. This

problem is an order-4 deceptive problem, and is typical of the kinds of problems used to test linkage learning algorithms.

```
1111 0100 1110 1011 0110 0101 1011 1111 0000 0000
|    |    |    |    |    |    |    |    |    |
fitness = 4  + 1  + 3  + 3  + 2 + 2 +  3+  4 + 5 +  5  = 32
```

Figure 6: A large partially deceptive problem can be formed by concatenating a number of smaller fully deceptive subproblems.

A GA that learns linkage will operate by recombining each of the optimal solutions to the 4 bit subproblems into one optimal 40 bit string consisting of all 0s (Q mGA paper, LLGA). A GA not learning linkage will tend to gravitate towards a suboptimal solution consisting of some 0000s, and some 1111s. This is what the simple GA will do [2]. In swapping genes between parents, it will often break up good combinations, such as 0000, by crossing them over with slightly worse combinations, such as 1111. The difficulty in learning linkage in this situation is that the four genes defining each subproblem don't have to be adjacent. To learn linkage, a GA must correctly pick out each set of four related genes. Even in this small problem, the number of such combinations is astronomical.

Figure 7 shows a possible size 8 population representing a set of solutions to this partially deceptive problem. This illustration depicts only one subproblem (four genes) of each chromosome. In figure 7 the GA has found several good solutions with marginal fitness contributions of 4 and 5 over these four genes. The uniform crossover shown destroys the correlations among the genes that lead to a high fitness; and the average fitness in the population decreases after crossover! Similarly, the order-1 probability representing this population is $P[] = [0.5\ 0.5\ 0.5\ 0.5]$. Yet, generating new solutions using this distribution leads to poor solutions.

```
marginal fitness

+4   1111……...   →   0011……...   +2
+5   0000……...   →   1100……...   +2
+4   1111……...       1111……...   +4
+4   1111……...       1111……...   +4
+5   1111……...       0001……...   +1
+5   0000……...       1110……...   +3
+4   1111……...       1101……...   +3
+5   0000……...       0010……...   +1
```

Figure 7: The first four genes of a population before and after a crossover that does not recognize building block boundaries.

The correspondence that holds between the cGA and the simple GA rears an ugly side to its head here. Both are unable to deal with this partially deceptive problem, in which linkage learning is crucial. That is, both uniform crossover, and order-1 probabilistic generation, fail to produce new chromosomes that are as good as the ones already in the population! Similarly, the solution to this problem holds dually in the realm of GAs and probabilistic algorithms.

In the GA, the crossover operator needs to understand that these four genes are related, and not break up the combinations they represent. A building block crossover can be developed for this purpose that only swaps whole solutions to subproblems, instead of single genes. In order to do this, however, the algorithm must guess correctly at which genes are related --- it must learn linkage.

In probabilistic algorithms, the probability distribution needs to recognize that these four genes are related, and represent the joint probability of these four genes having the 16 possible configurations they can hold; as opposed to the marginal distributions over each of the four genes independently. Such an algorithm would model the original population in figure 7 using $P[0000] = 0.5$ and $P[1111] = 0.5$, and each set of four genes would maintain their correlation from one generation to the next.

What we have just seen is that linkage learning is a skill that is easily transferrable into the domain of probabilistic algorithms. The remainder of this paper shows that the reverse is also true: that an operational and computationally feasible search for good distributions can fulfill the traditional task of linkage learning.

IV.2 What Makes a Good Probability Model?

The cGA and PBIL define what it means to generate new solutions that are like the current one. It is to poll the marginal distributions of each dimension or gene position, considering each of the gene positions to be independent. More complex algorithms have been developed that match some of the order-2 behavior of the population (11). These algorithms act by investigating pairwise inter-gene correlations and generating a distribution that is very close to polling from the population. The closeness measure most easily used is an information-theoretic measure of probability distribution distances (1). Modeling more complex, and more precise, higher-order behavior has been suggested, but the validity of doing so has been questioned (6).

Pursuing this last train of thought to its ultimate conclusion reveals the flaw in its prescription. We can directly model the order-L behavior of polling the population, by only generating new members through random selection of chromosomes that exist in the population already. This behavior will rapidly lead to the algorithm's convergence, while exploring no new structures. Thus, more accurate modeling of the population's distribution is not always a desirable course of action.

Probabilistic algorithms that use order-2 correlations have sometimes been found to be vastly superior to those using order-1 probabilities. Yet the argument above indicates that this trend cannot continue indefinitely up to order-*L* modeling of the population. At some point, this progression must stop. This puzzling combination seems to imply that more complicated models of the population are useful, but only up to a point.

These ruminations hint at a resolution to the problem of picking an appropriate distribution to model the population. The solution comes from realizing that the probability distribution to be used will represent a model of what makes the current population good; and that the population is simply a finite sample from this distribution. Fundamentally, the task of identifying a probability model to be used is then the induction of models that are likely to have generated the observed population.

It is well known that unbiased search for such models is futile. Thus we have no choice but to select a bias in this search space. The one we choose is that **given all other things are equal, simpler distributions are better than complex ones**. Simplicity here can be defined in terms of the representational complexity of the distribution, given the original problem encoding. All things are, however, rarely equal, and there remains a tradeoff between simplicity and accuracy. Our aim will therefore be to find a simple model that nonetheless is good at explaining the current population.

IV.3 Minimum Description Length Models

Motivated by the above requirement, we venture forth a hypothesis on the nature of good distributions:

> By reliance on Occam's Razor, good distributions are those under which the representation of the distribution using the current encoding, along with the representation of the population compressed under that distribution, is minimal.

This definition is a minimum description length bias on the model search for distributions (10). It directly penalizes complex models by minimizing over model size. In addition to doing so, it penalizes inaccurate models, because information theory tells us that these are unlikely to be of much use in the compression of the population (1).

IV.4 MDL Restrictions on Marginal Product Models

We take a moment now to explore this hypothesis. The basis for compression is the availability of a probability distribution over the space of structures to be compressed. Given any particular distribution, we can calculate how many bits it takes to represent a given message. In our case, the message is the population, and the distribution is the one to be evaluated. This hypothesis reformulates the problem of finding a good distribution as a new optimization problem --- that of finding the distribution model that minimizes the combined model and population representation.

For the remainder of this paper we focus on a simple class of probability models: those formed as the product of marginal distributions on a partition of the genes --- marginal product models (MPMs). These models are similar to those of the cGA and PBIL, excepting for the fact that they can represent probability distributions over more than one gene at a time. We choose these models for two reasons: 1) they make the exposition simpler; and 2) the structure of such a model can directly be translated into a linkage map, with the partition used defining precisely which genes should be tightly linked.

[0,3]		[1]		[2]	
00	0.5	0	0.5	0	0.6
01	0	1	0.5	1	0.4
10	0				
11	0.5				

Table 1: A marginal probability model over four genes.

To make MPMs more concrete, figure 9 shows one possible model over a four dimensional problem. The partition chosen is [0,3][1][2], which means the distribution represents genes 1 and 2 independently, but genes 0 and 3 jointly. Thus the probability distribution over [0,3] has a positive distribution for only the values 00 and 11. This means that at no time can the population generated by this distribution contain a 1 in the first position and a 0 in the fourth position. So, chromosomes of 1001 and 0100 are legal, but 0001 and 1010 are not! This form of restriction is not possible if the probabilities of genes 0 and 3 are represented independently. Obviously, this form of distribution is more powerful than that allowed by the cGA (or PBIL).

Let us now try to represent a population of N chromosomes, half of which are 0000, and half of which are 1111. If the populations represented as is (simple bit listing), this population will require $4N$ bits of storage. On the other hand, an MPM of genes [0,1,2,3] could first represent the probability of a structure being any of the 16 possible binary structures over those four positions. This probability distribution would indicate that only 1111 and 0000 have a positive probability of being represented. Subsequently, the representation of each structure can be 0 for 0000 and 1 for 1111. This encoding uses only one bit per structure, and thus only N bits for the whole population.

By recognizing that these four bits are correlated, and representing their distribution in an MPM, we have cut down to a fourth the amount of space that storing the population requires! Even when the probabilities are not so abrupt, and every string has a positive probability of being represented, the entropy of a distribution $E(P)$ gives us the average number of bits it takes to represent structures randomly pulled from this distribution. By calibrating an MPM's probabilities to match those of the population, this number can be used to estimate that distribution's compression of the population.

Furthermore, the calibrated MPM can easily be seen to be the one that compresses the population the most --- that is, incorrectly modeling the population cannot possibly help. In fact, since the order of chromosomes is deemed unimportant in the representation, a randomization of the ordering, followed by sampling a chromosome then projecting onto any gene subset from the MPM will be identical to polling the MPM at that subset. Thus, no distribution over that MPM subset can do better than that incorporating the population's frequency counts (1).

The use of an MPM to represent the population consists of two parts: 1) choosing the partition structure of the MPM; and 2) calibrating the MPM by pulling the frequency counts of each of the subsets of the MPM directly from the population. The efficacy of a particular distribution is defined as the sum representation size of the model itself and the population compressed under the model. At this point, there is little recourse but to explore the actual equations defining this criterion.

IV.5 The Combined Complexity Criterion

Let the Ith partition subset of an MPM be of size $S[I]$, where the sum of the S_I is L. Each subset of size S requires $2^{S[I]}$ frequency counts to define its marginal distribution - one for each possible configuration over those genes. Each of the frequency counts is of size $log\ N$, where N is the population size. Therefore the total model representation size, or complexity is:

Model Complexity = $log\ N\ \ \Sigma_I\, 2^{S[I]}$

Now, the Ith subset represents $S[I]$ genes. Let M_I be the marginal distribution over this subset. The entropy of this distribution is then $E(M_I)$. This number is the average number of bits it takes to represent these $S[I]$ genes in the population. This number will never be greater than $S[I]$. This is how the population is compressed, by representing the Ith subset's genes only after the Ith marginal distribution has been represented! Therefore, the total compressed population complexity is:

Compressed Population Complexity = $N\ \Sigma\ E(M_I)$

Armed with these two definitions, we can evaluate the efficacy of any given MPM structure. By MPM structure, we mean the MPM partitioning, without the actual probabilities for each of the marginal distributions. These probabilities are determined by the population, and the required condition that the compression be optimal. First, we calibrate the MPM structure using the population's frequency counts to form a full MPM model. Second, we add the model complexity and compressed population complexity to get a combined complexity number.

CC(MPM) = Model Complexity + Compressed Population Complexity

The next section describes a simple algorithm for searching for partitions of the gene space for a good MPM distribution over the population, given this operational criterion of finding a distribution such that its compressed complexity is suitably small.

V. The ECGA

In this section, we combine the above heuristic with a greedy search algorithm to invent an efficient probabilistic optimization algorithm. The proposed algorithm is very simple:

1. Generate a random population of size *N*.
2. Undergo tournament selection at a rate *S*.
3. Model the population using a greedy MPM search.
4. If the model has converged, stop.
5. Generate a new population using the given model.
6. Return to step 2.

This algorithm may also be stopped at any time, using the best found solution so far as its result. Most of the algorithm is self-explanatory, but we focus on two of its features. First, the algorithm requires both a population, and selection. Because we are remodeling the population at each generation, the structure of the models may not be stable. Therefore, selection cannot be replaced by a simple update as in the cGA. For the same reason, a concrete population is required also. Only the crossover step is replaced by probabilistic polling in this algorithm. Second, we have yet to describe the greedy MPM search.

The greedy MPM search begins each generation by postulating that all of the genes are independent --- that is, that the MPM [0][2]...[L-2][L-1] is best. What it will then do is perform a steepest ascent search, where at each step, the algorithm attempts to merge all pairs of subsets into larger subsets. It judges such mergers again on the basis of their combined complexity. If the best such combination leads to a decrease in combined complexity, then that merger is carried out. This process continues until no further pairs of subsets can be merged. The resulting MPM is then the one that is used for that generation. A new MPM search is thus carried out each generation. Significant optimizations can and have been taken in the implementation here, such as caching delta values for all pair combinations at each step.

This combined greedy search algorithm along with the minimum description length search criteria, applied to the task of optimization, will henceforth be referred to as the extended compact GA (ECGA).

V.1 Experimental Results

Most of this paper has concerned itself with the theoretical justification of the ECGA. This section shows how the ECGA can significantly speed the solution of

problems that are partially deceptive. In the creation of partially deceptive functions, we will rely on the composition of small deceptive problems, like the 4-bit problem defined above. The subproblems we will use are trap functions, whose fitness relies solely on the number of 1s present in a chromosome. These functions have been used extensively in the testing of linkage learning algorithms, and solving them has proven to be quite challenging in the absence of prior linkage information.

We will begin by exploring the relationship between the population size used and the proportion of subproblems solved correctly by both the ECGA and the simple GA using uniform crossover. By adding in information about the algorithms' running time, we can show comparisons of the number of function evaluations both algorithms need to achieve a comparable level of optimization. Without further ado then, we proceed to the experimental results.

V.2 Deceptive Trap Functions

In this section, ten copies of the four-bit trap subproblem, are concatenated to form a difficult 40-bit problem, as in figure. This is the problem used to compare the ECGA with the simple GA. Each set of four neighboring genes [0-3][4-7] thus formed one subfunction to be optimized. But neither the simple GA nor the ECGA were told which genes were related, or that the related groups were contiguous, or for that matter the size of the subproblems.

Both the ECGA and the simple GA with uniform crossover were run on this problem 10 times, with a selection rate of 16 (which is higher than the ECGA needs, but which the simple GA requires), gathering the average number of subfunctions solved per population size. This is measure is especially significant, as the performance of GAs and other optimization algorithms is typically judged by the number of objective function evaluations they undertake - and this number is the population size times the number of generations.

Table 2 shows the population size versus the average number of subfunctions solved for the simple GA, and the average number of function evaluations taken to do so. Table 3 does the same for the ECGA.

Population size	Subfunctions Solved	Objective Evaluations
100	3.9	740
500	5.2	5100
1000	6.1	15600
5000	6.8	100000
10000	7.3	248000
20000	8.0	614000
50000	7.9	1560000
100000	8.8	3790000

Table 2: Simple GA complexity on deceptive subproblems.

Population size	Subfunctions Solved	Objective Evaluations
100	4.0	750
200	5.2	1460
300	7.1	2610
500	9.3	4000
600	9.9	5040
1000	10.0	7300

Table 3: ECGA complexity on deceptive subproblems.

The differences between the simple GA and the ECGA are large, and in the favor of the ECGA. To consistently solve 9 building blocks, the simple GA needs a population size of 100 thousand and over 3.8 million function evaluations! To do the same, the ECGA needs a population size of 500 and 4 thousand function evaluations. On this small 40-bit problem, the ECGA is 1000 times faster than the simple GA. This speedup is due to the careful attention paid to probabilistic modeling in the ECGA. This speedup should theoretically also become much greater when solving larger problems. The following shows the successive MPM structures used in one successful run of the ECGA:

GENERATION 0

[0-3][4 7 27][5-6 15][8-11][12-14 24][16-19][20-23][25][26 32-35][28-31][36-39]

GENERATION 1

[0-3 25][4-7 15][8-11][12-14][16-19][20-23][24 27][26 32-35][28-31][36-39]

GENERATION 2

[0-3][4-7][8-11][12-15][16-19][20-23][24-27][28-31][32-35][36-39]

Note that this structure changes from generation to generation. The ECGA makes a few mistakes in the first pair of generations. The final result arrived at in generation 2, however, completely discerns the subfunction structure of the given problem --- without being given this information ahead of time. By simply searching for MPM-structures that optimally compress the population, the ECGA has completely dissected the subproblem structure of the 40-bit problem! It is no surprise that armed with this information, the ECGA proceeded to optimize this problem much faster then the simple GA.

V.3 The Role of Selection

The role of selection is a curious one in the ECGA and not at all immediately clear. If the proper level of selection is not maintained in the above runs however, the ECGA fails, and never models the problem structure correctly (the same is true but an often

ignored aspect of the simple GA). Taking a second, we consider the dual roles of probabilistic generation and selection in the ECGA. The role of generation (the ECGA's crossover equivalent) is to create more chromosomes that are like the ones in the present population. These chromosomes will have no correlations across the MPM structure boundaries. That is, if the MPM structure says that genes 1 and 2 are independent, they will actually be independent in the generated population. Given that the algorithm begins by assuming that all genes are independent, one might wonder where dependencies come from at all.

The answer to that question is that selection recorrelates genes if a specific combination of theirs is correlated with high fitness. In the partially deceptive problems we have experimented on, selection correlates the group of 0s defined over a common subproblem. This correlation is what the ECGA detects. If the level of selection is very low, this correlation will never be generated, and the ECGA will never detect that the genes are related. Thus, a low level of selection can cause the ECGA to fail. This point is especially important in considering the future of research on such probabilistic algorithms. It also points to a related class of algorithms that learn the proper MPM structures by exploring intergene correlations.

VI. Summary, Conclusions and Future Work

This paper began by reviewing the simple GA and a related set of probabilistic algorithms. Past work has equated some of these algorithms with the simple GA, and an exploration of this relationship has pointed to the existence of more general probabilistically-based GA-like algorithms. That this class exists has been pointed out before (6). However, the benefits of paying close attention to the probabilistic modeling of the a GA-like population have not previously been thoroughly explored.

This paper demonstrated that proper probabilistic modeling in these algorithms is in effect the long-sought solution to the linkage-learning problem. It has also introduced an operational complexity criterion for distinguishing between good models and bad models. Experimental results have shown that by focusing on learning marginal probability models, the ECGA can solve some difficult problems orders of magnitude faster than simple GAs not using linkage information.

This paper has revealed a strong connection between linkage learning and proper probabilistic modeling. This is however, only the tip of the iceberg inasmuch as effective optimization is concerned. The goal of linkage learning, while ambitious, is only a small part of the more general goal of representation learning. In representation learning, the actual optimization problem is transformed into a different space, and optimized in that new space. The optimization and design of biological entities --- which transforms fitness functions defined on three dimensional molecules into ones over a genetic representation --- is proof that such techniques can be effective. It is this author's belief that even such a search for transformations can be placed into the framework of complexity based modeling.

Several questions and difficulties remain to be addressed by future work on probability based optimization in general, and MPM-like approaches in particular. A few of the more promising or addressable issues in this area are:

- The ECGA is simple to parallelize, by replacing the migration step of standard parallel GAs by one of probability model exchange. This has the potential to greatly reduce the amount of communication and synchronization needed over that of the parallel simple GA.
- The MPM model search is potentially computationally expensive. In some problems, this poses the risk of overwhelming the function evaluation time with the search's own computational overhead. One recourse in these cases is to use simpler probability models, such as those used by the cGA or MIMIC. Another possible alternative is to implement this fixed search algorithm in hardware, or to look for heuristic approximations to the MPM algorithm. For example, the MPM search could be biased somewhat to the encoder's original linkage specification.
- Another approach to reducing the complexity of the MPM model search is to sample from the population when building the correct MPM structure.
- On the other spectrum of function evaluation complexity, it is possible that more time for population analysis might be available. In these cases, bayesian network learning, although more complex, is likely to yield even more powerful algorithms than the one described in this paper. In some cases, such as optimization in euclidean spaces, direct probabilistic modeling might also offer more accurate methodologies than modeling using arbitrary binary encodings over the selfsame space.
- The probability modeling framework suggested here deals particularly well with multidimensional data, but does not trivially extend to optimization over more complex structures such as permutations, or programs. This issue deserves serious consideration.
- Algorithmic complexity analysis of the ECGA is likely to be difficult. However, it is quite likely that simple estimates of its overhead costs over the simple GA on non-deceptive problems would be straightforward to calculate.
- Even when using the most complex probability models, it is quite likely that high levels of selection will be required to solve certain problems. The role of selection there will be to correlate the features of the population, so that they may be detected in the probabilistic model. To date, few theories explain this dependence of GA-like algorithms on selection. The SEARCH framework seems likely to offer at least a partial answer to this puzzle.
- It is quite possible that by searching for mixtures of different models, one might be able to optimally decide between the application of different GA operators. That is, the probabilistic approach could settle once and for all, on a problem by problem basis, the efficacy question of crossover versus mutation.
- Finally, while in theory the ECGA seems to provide a huge advantage over optimizing with the simple GA, this result has yet to be extended to the solution of a practical, real-world problem.

References

1. Cover, T.M. and Thomas, J.A., *Elements of Information Theory*. John Wiley & Sons, 1991.
2. Thierens, D. and Goldberg, D.E., "Mixing in genetic algorithms." *Proceedings of the Fifth International Conference on Genetic Algorithms* pp. 38-45, 1993.
3. Goldberg, D.E., Korb, B. and Deb, K., "Messy genetic algorithms: Motivation, analysis and first results." *Complex Systems*, 1989, vol. 3, no. 5, pp. 493-530.
4. Harik, G., "Learning Linkage to Efficiently Solve Problems of Bounded Difficulty Using Genetic Algorithms." IlliGAL Technical Report No. 97005, University of Illinois at Urbana-Champaign, Urbana, IL, 1997.
5. Kargupta, H., "SEARCH, polynomial complexity, and the fast messy genetic algorithm." IlliGAL Technical Report No. 95008, University of Illinois at Urbana-Champaign, Urbana, IL, 1995.
6. Muehlenbein, H., "Schemata, Distributions and Graphical Models in Evolutionary Optimization." Unpublished manuscript. Http://set.gmd.de/AS/ga/publi-neu.html
7. Baluja, S. and Caruana, R., "Removing the Genetics from the Standard Genetic Algorithm.", CMU Technical Report CMU-CS-95-141. Http://www.cs.cmu.edu/afs/cs/user/baluja/www/techreps.html, 1995.
8. Baluja, S., "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning.", CMU Technical Report CMU-CS-94-163. Http://www.cs.cmu.edu/afs/cs/user/baluja/www/techreps.html, 1994.
9. Harik, G., Lobo, F. and Goldberg, D.E., "The Compact Genetic Algorithm." *Proceedings of the 1998 IEEE Conference on Evolutionary Computation*, pp. 523-528, 1998.
10. Mitchell, T.M., *Machine Learning*. McGraw Hill Text, 1997.
11. De Bonet, J.S., Isbell, C.L, and Viola P., "MIMIC: Finding Optima by Estimating Probability Densities. " *Advances in Neural Information Processing Systems*, 1996, vol. 9.
12. Goldberg, D.E., *Genetic Algorithms in Search Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
13. Holland, J.H., *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan, 1975.
14. Muhlenbein, H. and Schlierkamp-Voosen, D., "Predictive Models for the Breeder Genetic Algorithm: I. Continuous Parameter Optimization. " *Evolutionary Computation*, 1993, vol. 1, num. 1.
15. Syswerda, G., "Uniform Crossover in Genetic Algorithms." *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 2-9, Morgan Kaufmann, 1989.