The Relevance of Genotypic Learning in the CLGA*

Terry P. Riopka and Peter Bock
George Washington University
Dept. of Computer Science, 801 22nd St. NW
Washington, DC 20052
Email: {riopka, pbock}@seas.gwu.edu

Abstract
A new recombination framework for genetic algorithms referred to as the Collective Learning Genetic Algorithm (CLGA) has been demonstrated which utilizes genotypic learning to do recombination based on a cooperative exchange of knowledge between interacting chromosomes. Although preliminary experiments suggest that the CLGA may be an effective algorithm for searching for solutions to highly epistatic, non-separable combinatorial optimization problems, whether or not the mechanism of genotypic learning is responsible for this apparent success is less clear. This paper presents results that show that genotypic learning does play a significant role in recombination, independent of all other policies of the CLGA. It also provides insight into the benefits of genotypic learning over other recombination mechanisms within the context of the CLGA framework.

1 INTRODUCTION
The Collective Learning Genetic Algorithm (CLGA) has recently demonstrated an interesting new paradigm for genetic algorithms utilizing genotypic learning to guide recombination deterministically in a distributed network of interacting agents [Riopka and Bock, 2000]. Individuals of the population collaborate and exchange knowledge instead of symbols in order to modify their own chromosome strings in a process referred to as intelligent recombination. Thus, random crossover is essentially replaced with a consensus of information based on individual experience and observation. In addition, individuals maintain their own strings throughout evolution, preserving a modified string only if it is the same or better than the last.

Although evidence for the utility of the CLGA is slowly accumulating [Riopka and Bock, 2000][Riopka, 2000], the relevance of genotypic learning itself is less clear. It is possible that given the object-oriented framework of the CLGA, any recombination method substituted for intelligent recombination might result in reasonable CLGA performance. Before devoting significant effort into testing the CLGA, it is important to establish the degree to which the central concept of the CLGA, specifically genotypic learning, is relevant to its successful operation. The objectives of the research presented in this paper are to show that genotypic learning does play a significant role in CLGA recombination, independent of all other CLGA policies, and to provide insight into the relative benefit of genotypic learning over other recombination mechanisms within the context of the CLGA framework.

The paper is organized as follows. Section 2 discusses the motivation behind the development of the CLGA. Section 3 reviews the main features of the algorithm and establishes the primary research hypothesis. Section 4 presents and discusses the results, and the paper is summarized and concluded in Section 5.

2 MOTIVATION
The linkage problem, the representation problem, building block disruption, deception, the reordering problem: all of these (at some level) can be looked at as different names for the same phenomenon, and that is, the detrimental effect of interdependent genes on GA performance. Although it is far from obvious what exactly makes a problem hard for GAs to solve, the success of a GA appears to be highly correlated with the degree of interdependency between features and their proximity. This characteristic is referred to as genetic linkage, and refers to the non-linear epistatic relationships between different variables of the solution. The problem is not new; even Holland and his colleagues were well aware of the difficulties associated with epistatic bit interactions and their deleterious effect on GA performance. As a result, a significant amount of work has been done in this area.

In general, most researchers have realized the limitations of static, non-adaptive approaches to recombination and have taken steps towards developing more advanced

*Published as a poster in Proc. of the Genetic and Evolutionary Comp. Conf. (GECCO-2001)
methods that attempt to either adapt to the problem representation at hand, or to more efficiently exploit global information in the evolving population. Several of these include fmGA [Goldberg, Deb, Kargupta, and Harik, 1993], GEMGA [Bandyopadhyay, Kargupta and Wang, 1998], LLGA [Harik, 1997], BOA [Pelikan, Goldberg and Cantu-Paz, 1999], as well as the adaptive poly-parental recombination strategy of [Smith and Fogarty, 1996], the automata-based approach of [White and Oppacher, 1994] and selective crossover [Vekaria and Clark, 1998], to name a few.

The CLGA framework combines the idea of adaptive recombination with the intuition behind solution modeling, i.e., that an evolving population contains a great deal of information that is not adequately being exploited. Given the task of optimization and an unknown problem representation, an intelligent algorithm should be able to observe the consequences of its actions and to learn from them to devise more effective ways of generating new solutions. The CLGA attempts to do this through a fusion of ontogenetic and phylogenetic adaptive processes as described below.

3 CLGA DESCRIPTION

The general structure of the CLGA is the same as that of most evolutionary algorithms and very similar to the SGA. Consequently, the underlying idea remains the same, i.e., a population of solutions (or individuals) represented as chromosome strings evolves through recombination, mutation, and selection according to fitness. The CLGA however, differs from the SGA in two major respects: it incorporates a learning mechanism at the genotypic level, and redistributes control of recombination, mutation, and selection to the individual chromosomes. The following section reviews the main features of the algorithm to provide the necessary background for understanding the goals of this paper. Note that all but mutation are implemented in exactly the same manner as described in [Riopka and Bock, 2000].

A CLGA consists of a population of adaptive learning agents called SmartChromosomes. Each SmartChromosome consists of two components: an instantiation of a collective learning automaton (CLA) and a chromosome string representing the best solution the SmartChromosome has found so far. The concept of a CLA is derived from collective learning systems theory [Bock, 1993] an adaptive approach to learning, closely related to the fields of reinforcement learning, connectionism and learning automata. In general, a CLA incorporates several components: a set of sensors for the perception of stimuli from an external environment, a network of adaptive processors for the synthesis of trial responses based on the perceived stimuli, an evaluation policy for measuring the effectiveness of the trial responses, and a compensation/ update policy for the dynamic modification of the knowledge stored in the memory of its processors.

In this application, the “sensors” of the CLA consist of a fixed number of feature detectors, each associated with a histogram that contains the accumulated knowledge of the fitness of schemata the SmartChromosome has “observed”. Each feature detector monitors a unique set of chromosome sites. Its corresponding histogram contains one bin for every possible permutation of symbols for the chromosome sites, which the feature detector monitors. The number of feature detectors in each SmartChromosome \((d)\) and the cardinality of the set of monitored sites \((k)\) are both parameters of the algorithm. For example, for binary encodings, a feature detector monitoring \(k\) chromosome sites will have \(2^k\) bins.

In each generation, SmartChromosomes interact, “inspecting” the strings of other SmartChromosomes and “interrogating” each other to obtain information which can help each improve its own string. Instead of exchanging schemata parts, each combines its own knowledge with others’ to determine the manner in which it will modify its string. If a modification results in a string superior to its previous string, the new string is retained, otherwise, the SmartChromosome reverts to the previous one. In either case, the SmartChromosome learns from the interaction and subsequently passes on what it learns to other SmartChromosomes it encounters. A flowchart depicting the main elements of the CLGA is shown in figure 1.

Given a feature detector cardinality of \(k\), the total number of possible combinations of monitored sites is \(N\) chromosome sites taken \(k\) at a time \(\binom{N}{k}\). The number of SmartChromosomes \((M)\) in the CLGA population is determined by the number of feature detectors per SmartChromosome and the combination ratio \(r_c\), the fraction of combinations actually incorporated into the population. Hence,

\[
M = \left\lfloor \frac{r_c \times \binom{N}{k}}{d} \right\rfloor
\]

Incorporating all combinations insures some degree of problem-representation independence with respect to the recombination process. A population is created by randomly selecting the required number of combinations (without replacement) from the total and distributing feature detectors as randomly as possible among the SmartChromosomes, taking care to insure a uniform cover for all chromosome sites. The cover of a chromosome site is the total number of feature detectors in the population whose monitored sites include that site. This insures that chromosome sites are all fairly represented by the distributed feature detectors.

Initial chromosome strings are generated randomly. The following paragraphs explain the main elements of the CLGA flowchart shown in figure 1.
**Mating.** Each SmartChromosome mates with \( m \) other SmartChromosomes selected randomly from the population. Mating merely determines which subset of SmartChromosomes will be inspected and interrogated by each SmartChromosome. Note that mating in the context of the CLGA is not reciprocal, *i.e.* a SmartChromosome inspects and interrogates its mates but not (necessarily) vice versa.

**Intelligent Recombination.** Modification of schemata occurs during a process referred to as **interrogation.** However, decisions made during interrogation are based on the knowledge accumulated by the individual SmartChromosomes over the course of evolution through a process referred to as **inspection.** Intelligent recombination therefore comprises two processes: an acquisition of knowledge (inspection) and application of that knowledge to direct recombination (interrogation).

A SmartChromosome **inspects** the strings of its mates by noting in each mate’s string the particular permutation of symbols at the location of the sites monitored by the SmartChromosome’s feature detectors. The particular permutation of symbols observed in the inspected string acts as a stimulus for each feature detector. The arrival of a stimulus causes each feature detector to generate a response, whose evaluation is the fitness of the inspected string. A compensation policy adjusts the final value in the corresponding feature detector bin by replacing the previous value with the average of all the evaluations (including the current one) to date. Note that this average is not weighted, so that all observations are given equal weight.

A SmartChromosome **interrogates** its mates by superposing the best states of its own feature detectors with those of its mates’ feature detectors in order to modify its string. Best states from each mate’s feature detectors function as stimuli causing the SmartChromosome to modify its string as follows. All best states are sorted in descending order by weight magnitude. The corresponding permutations of symbols are superposed sequentially state by state beginning with the best state with the largest magnitude, overwriting the relevant chromosome sites in the SmartChromosome’s string. In order for a state to be **consistent,** none of the monitored sites of the corresponding feature detector may overlap with those of any of the previously integrated states or, in case of overlap, the symbols in the overlapping sites must be identical. States, which are not consistent, are omitted. The fraction of a chromosome’s sites affected by the best state superposition is referred to as the **superposition fraction.**

**Directed Mutation.** Standard mutation is not applied in the CLGA. Many informal experiments have shown that standard mutation tends to degrade CLGA performance. Consequently, a **directed mutation** operator is applied as follows. Each SmartChromosome maintains a FIFO∗ list of the last \( h \) unique chromosome strings it has evaluated, where \( h \) is some small number (to limit time and memory resources) referred to as **history length.** A SmartChromosome first searches its memory to see if the current string has already been evaluated. This will occur only if interrogation results in the creation of a string

\* FIFO here means that the chromosome that hasn't been repeated in the longest time is replaced first.
already in memory. If the string has already been evaluated, a single bit is randomly selected in the string and complemented. A mutation template is maintained for each chromosome in memory to keep track of which bits have been changed, so that the next time that same string is obtained, a different bit can be changed. Only those bits that have not already been changed are eligible for mutation each time. Once all single bits are exhausted, combinations of two bits are complemented. Once those combinations are exhausted, combinations of three bits are complemented, etc.

Evaluation. The SmartChromosome evaluates its string using the fitness function.

Individual Selection. If a modification results in a string whose fitness is greater than the previous string, the new string is retained, otherwise, the SmartChromosome reverts to the previous one. In effect, the offspring competes with its parent. The SmartChromosome inspects its new string regardless of what its relative fitness is, thus learning from both its successes and its failures.

In summary, the relevant parameters of a CLGA are: the number of feature detectors ($d$), feature detector size ($k$), number of mates ($m$), history length ($h$), superposition fraction ($s_f$) and combination ratio ($r_c$) (or indirectly, population size). In the CLGA algorithm, population size is related to the other parameters by equation 3.1.

The effect of genotypic learning will be determined in two ways. First, since intelligent recombination is based on the superposition of best states as determined through genotypic learning, a method of recombination that involves the superposition of random states should result in significantly poorer CLGA performance. A significant difference in CLGA performance would suggest that genotypic learning is indeed a critical aspect of the CLGA. It would also validate the relevance of genotypic learning in promoting useful recombination.

The effect of genotypic learning will also be determined by comparing the performance of the CLGA with the following recombination mechanisms substituted for intelligent recombination: parameterized uniform crossover, two-point crossover and mutation only. The object-oriented framework of the CLGA was specifically developed to enable the mechanism of genotypic learning to function as a method of recombination. A very important question is to what degree the framework itself is responsible for improved CLGA performance on optimization problems. For example, it may be hypothesized that any form of recombination e.g., uniform crossover, may be just as effective given the structure of the CLGA and the highly conservative selection policy. The fact that genotypic learning may be a useful mechanism for recombination is important, but it is equally important to determine the extent to which it benefits the CLGA. Comparing the performance of intelligent recombination to other forms of recombination given the CLGA framework may provide insight into the benefits of the genotypic learning paradigm.

We therefore present the following hypothesis: a CLGA implementing intelligent recombination outperforms a CLGA that implements each of the following forms of recombination: random state selection, parameterized uniform crossover, two-point crossover, and mutation only over a range of problem epistasis.

4 EXPERIMENTS

A reasonable architecture for the CLGA was chosen and used to test the various methods of recombination. In all cases, the only aspect of the algorithm that was modified was the method of recombination. The modularity of the algorithm enabled the simple substitution of the different mechanisms of recombination, yielding a CLGA-type algorithm that performed identically except for the way in which schemata were modified.

4.1 FIXED PARAMETERS AND CONDITIONS

Several problem generators have been developed to facilitate the design of more controlled experiments for testing evolutionary algorithms [DeJong et al., 1997][Heckendorn et al., 1998]. Random problems generated by an NK-Landscape problem generator were used in each of the following experiments.

In the NK model of fitness landscapes [Kauffman, 1989], $N$ represents the number of genes in a chromosome, and $K$ represents the number of linkages each gene has to other genes within the chromosome. Chromosome fitness is computed by averaging the fitness contribution of each locus. The fitness of each locus is obtained by using the corresponding allele along with the other $K$ linked alleles as an index into a table of $2K+1$ randomly generated numbers uniformly distributed in the real interval [0, 1].

In the following experiments, a random model was used, meaning that for a given locus, the set of $K$ linked genes are located randomly throughout the chromosome. For the purpose of generating links, the chromosome is circular.

Problem size was fixed at $N=20$. A relatively small value of $N$ was chosen for two reasons. First, this enabled the normalization of fitness values, which removed the effect of optima variance and resulted in more accurate confidence intervals. Second, the smaller problem size resulted in faster processing which enabled a more thorough optimization of the competing "substituted-recombination" CLGAs.

The following fixed parameter values for the CLGA were used: $m=1$, $h=4$, $d=2$, $k=3$, $r_c=1.0$, and $s_f=0.4$, resulting in
a population of $M=572$ SmartChromosomes. Note that a thorough investigation of all of these parameters is ongoing.

Although these parameters were not optimized, they were selected based on heuristics derived from extensive informal experiments [Riopka, 2000]. Feature detector size, $k=3$ has been used extensively and found to be effective over a range of problem size and level of epistasis. History length, $h=4$ does not significantly affect the most active (and most interesting) stages of evolution, but does affect directed mutation in the latter stages of evolution to some degree. Although the range of the superposition fraction is in the interval $[k/N, 1.0]$, good results have been consistently obtained using values ranging between 0.4 and 0.6. Recall that a combination ratio of $r_e=1.0$ insures that all combinations of feature detectors are present in the population, a minimum requirement for representation-independence in the CLGA. The number of feature detectors per SmartChromosome, $d=2$, was chosen to result in a reasonable population size. Number of mates, $m=1$ is a standard setting that has not been explored.

Fixed parameters for all substituted-recombination CLGAs are shown in table 1. Each algorithm was optimized to obtain the best possible performance across the different levels of epistasis, given the particular substituted-recombination method.

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Algorithm Description</th>
<th>Relevant Parameter Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>m-CLGA</td>
<td>Mutation Only</td>
<td>$\mu = 0.1$</td>
</tr>
<tr>
<td>r-CLGA</td>
<td>Random State Selection</td>
<td>n/a</td>
</tr>
<tr>
<td>puc-CLGA</td>
<td>Parameterized Uniform Crossover</td>
<td>$p_c = 0.5$</td>
</tr>
<tr>
<td>tpc-CLGA</td>
<td>Two-Point Crossover</td>
<td>$p_c = 0.9$</td>
</tr>
</tbody>
</table>

**Table 1:** Optimal parameter values for substituted recombination algorithms.

### 4.2 FACTORS

The initial experiments varied only one factor: the degree of problem epistasis. $K$ set to 2 for low epistasis, 6 for medium epistasis, and 10 for high epistasis.

Each algorithm was run on 50 different problems of low, medium and high epistasis, generated randomly using the NK-Landscape problem generator.

### 4.3 PERFORMANCE METRICS

Performance is measured by the best-so-far string fitness, normalized by the problem optimum. Average performance over all random problems up to and including the current generation is computed and plotted for each generation. In addition, percent optima found is also plotted. All experiments were run for 250000 function evaluations.

### 4.4 RESULTS

Results for the "substituted recombination" experiments are shown in figure 2. The vertical hi-lo bars represent 95% confidence intervals based on results for the 50 random problems. Figure 2a shows normalized average best-so-far fitness (NABF) in the early part of evolution up to 50000 evaluations (approximately 90 generations). Figure 2b shows the percent optima found (POF) up to 250000 evaluations (approximately 450 generations).

![Figure 2a: Normalized Average Best-so-far Fitness (NABF) plots for various methods of recombination substituted for intelligent recombination, averaged over 50 random problems of size $N=20$ generated by an NK-Landscape problem generator for low, medium and high levels of epistasis. 95% confidence intervals are shown.](image-url)
The most important observation to make is that intelligent recombination consistently outperforms random state selection across all levels of epistasis by a significant margin. This is very important because this strongly supports the hypothesis that genotypic learning is a relevant mechanism for recombination. If genotypic learning were not providing useful guidance in making decisions for modifying schemata, random state selection would be just as useful.

Had this not been the case, one could have argued that intelligent recombination simply provides a convenient form of mutation. In fact, this may be a useful interpretation of what occurs when epistasis is high. Note that the performance of mutation-only is comparable to (though still inferior to) that of intelligent recombination for higher levels of epistasis. It is well known that as epistasis increases, the fitness landscape becomes more and more uncorrelated [De Jong, 1993]. It is not unreasonable to infer that as epistasis increases, the amount of useful information available for intelligent recombination decreases, due to smaller and smaller correlation between fitnesses of similar solutions in Hamming space. In other words, it becomes more and more difficult for the CLGA to learn consistent relationships between bits due to greater variance in solution fitness. Consequently, intelligent recombination may act like mutation at high levels of epistasis, because its decisions have greater variance and the source of the bits used to modify the schemata is the histogram of each feature detector, not the schemata. This last point needs some elaboration.

Recall that the schemata are modified using knowledge stored in the histograms of the feature detectors. It does not rely explicitly on the schemata for modifying bits as do regular recombination methods that exchange bits between chromosomes. Consequently, decreasing diversity in the population can reduce the effectiveness of standard recombination methods, requiring an explicit mutation operator to re-introduce diversity; whereas in the CLGA, decreasing diversity in the population has nothing to do with the diversity of states in the feature detector histograms. Some states may be preferred over others, but any permutation of bits is potentially possible. This may explain why the CLGA works so well without an explicit mutation operator and why it behaves similarly to mutation-only at high levels of epistasis. It is interesting to note that at low levels of epistasis ($K=2$), intelligent recombination performs significantly better than mutation-only, reaching better (and optimal) solutions in fewer effective evaluations. This strongly suggests that intelligent recombination can adapt to the level of epistasis to some degree.

Both two-point crossover and parameterized uniform crossover perform reasonably well at a low level of epistasis, although parameterized uniform crossover appears to perform slightly better than two-point crossover. This is not unexpected, because the representation independent nature of uniform crossover would be expected to perform better on problems where there is loose genetic linkage (as is the case with the random model NK functions used here). Uniform crossover does not contiguously. Recall that genetic linkage refers to the proximity of related genes on the chromosome. An operator like two-point crossover would be expected to work better than uniform crossover only if genes are proximally located on the chromosome (i.e. in case of tight genetic linkage), because two-point crossover has a bias for keeping contiguous genes together and uniform crossover does not.

Although both operators perform comparably to intelligent recombination at low levels of epistasis, for high levels of epistasis they perform significantly worse. Note that both uniform crossover and intelligent recombination are representation independent. The
difference is that intelligent recombination makes an explicit effort to learn linkage, whereas uniform crossover does not. These preliminary results strongly suggest that some linkage learning is taking place, yielding benefits at many levels of epistasis.

5 CONCLUSIONS AND FUTURE WORK

As the preliminary results show, intelligent recombination seems to have the best of both worlds. It performs as well or better than mutation-only at high levels of epistasis and as well or better than uniform crossover at low levels of epistasis, indicating at least some ability to adapt to the level of problem epistasis. The results presented are somewhat subtle, but the behavior of the CLGA suggests that the benefits of genotypic learning may translate to larger rewards for larger problem sizes. This, however, will need to be verified in future work. In the mean time, work is progressing to more thoroughly investigate the effect of various parameters on CLGA performance and to more accurately characterize the behavior of the CLGA on larger and more varied types of problems.

References


