Intelligent Recombination Using Individual Learning in a Collective Learning Genetic Algorithm

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

This paper introduces a new *collective learning genetic algorithm* (CLGA) which employs individual learning to do intelligent recombination based on a cooperative exchange of knowledge between interacting chromosomes. Each individual in the population observes a unique set of features in the chromosomes with which it interacts in order to *explicitly* estimate the average fitnesses of schemata in the population, and to use that information to guide recombination. The stages of evolution are still controlled by a global algorithm, but much of the control in the CLGA is distributed among chromosomes that are individually responsible for recombination, mutation and selection. The effectiveness of the approach is demonstrated on random problems generated by an NK-Landscape problem generator. Preliminary results suggest that the CLGA may be especially effective for searching for solutions to highly epistatic, non-separable problems, a class of problems traditionally difficult for regular GAs.

1 INTRODUCTION

Theoretical understanding of the success of the simple genetic algorithm (SGA) has been largely based on Holland's (1975) notion of "schemata", a formal term for the more informal notion of "building blocks". Traditional theory assumes that GAs work by recombining and propagating smaller pieces of good solutions from one generation to the next to generate better solutions over time, *implicitly* estimating the average fitnesses of all schemata present in a population and increasing or decreasing their representation according to the Schema Theorem [Holland, 1975]. The performance of GAs in general, is largely dependent on representation. 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. Theoretical studies have shown that if an effective *linkage-learning* GA were developed, it would have significant advantages over the SGA [Thierens and Goldberg, 1993]. It would not only be less sensitive to the representation problem, but would also be able to solve a wider variety of problems faster, more reliably and more accurately than regular GAs.

The preservation and proliferation of good building blocks in GA problems becomes increasingly problematic as the epistatic relationships between variables grow. A standard genetic algorithm implicitly manipulates large numbers of building blocks using problem-independent recombination operators that can often disrupt relevant building blocks and lead to premature convergence to sub-optimal solutions. Three main approaches have been taken to try to prevent building block disruption, towards the development of a more robust GA capable of dealing with higher order feature interactions. Several researchers have focussed on evolving a problem's chromosome representation in conjunction with its solution [Goldberg, 1989; Harik, 1997; Kargupta, 1995]. Others have attempted to evolve recombination operators using self-adaptation mechanisms [Schaffer, 1987; Spears, 1995; Smith and Fogarty, 1996]. Still others have sought to replace the concept of recombination entirely by explicitly modeling the distribution of good solutions in a given problem in order to guide further search [Baluja, 1994; Harik, 1999; Muhlenbein & Paaß, 1996].

The role of individual learning in the context of evolution is ambiguous, but its effect can be conjectured at a number of different levels. Baldwin theorized that individual learning could guide the evolutionary process by rewarding partial genetic successes. Individuals with useful genetic variations could exploit them by learning, thereby surviving preferentially and eventually replacing abilities that previously required learning with genetically-determined sequences that predisposed them to those abilities [Baldwin, 1896]. At another level, individuals in human societies acquire life experience which they pass on through their offspring by sharing that life experience verbally either through oral or written exchange. In addition, superior intelligence and selective mating may result in genetically superior offspring potentially biasing the distribution of future offspring. Finally, at the highest level, subsequent generations benefit from the learned experiences of their ancestors, building on that knowledge and using it to advance civilization.

The research presented in this paper seeks to address the following question: can individual learning and exchange of knowledge help a genetic algorithm to search a solution space more effectively? In particular, can such an approach lead to a more robust GA capable of dealing with higher order feature interactions? **The objective of this research is to design and test an approach which employs individual learning to do intelligent recombination based on a cooperative exchange of knowledge between interacting chromosomes.** Each individual in a population "observes" a unique set of features in the chromosomes with which it interacts in order to *explicitly* estimate the average fitnesses of schemata in the population, and to use that information to guide recombination. Individuals collaborate and exchange knowledge to modify their chromosome strings, essentially replacing the random crossover operator with a consensus of information based on the permutations of observed symbols that yield the best string fitness.

The following section summarizes the proposed algorithm, giving a high-level pseudocode description of its main features. It describes the central element of the approach, specifically: the mechanism for acquiring, storing, and exchanging knowledge between individuals and then continues to elaborate on remaining details. Section 3 presents preliminary results, and section 4 presents conclusions and describes future work.

2 CLGA DESCRIPTION

A *Collective Learning Genetic Algorithm* (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. Figure 1 shows a graphical depiction of two SmartChromosomes, taken from a binary encoded population with d=2, k=3 and string length N=12.

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. CLGA Pseudocode is shown in Figure 2.

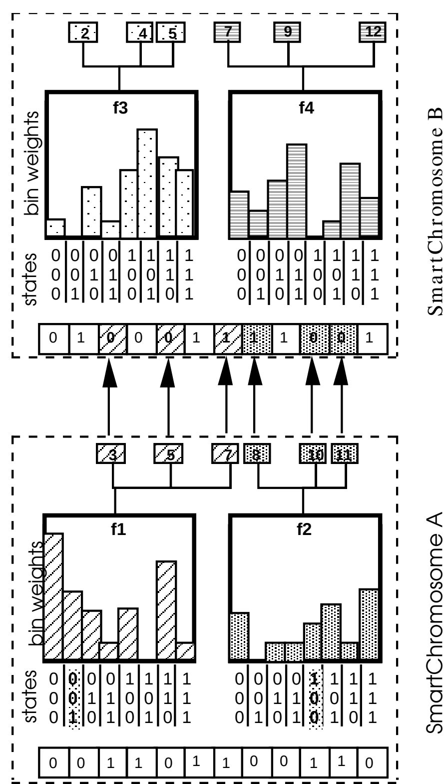

Figure 1: During inspection, the feature detectors of SmartChromosome A "observe" the shaded symbols pointed to by arrows. For feature detector f1, this causes the bin weight for state [001] to be replaced by the average of the fitness of SmartChromsome B's string and all other string fitnesses that contributed to that bin in the past. The same is done for bin weight f2[100].

```
Create population of P SmartChromosomes

Initialize and evaluate SmartChromosome strings

REPEAT (until termination criterion is met)

  FOR i=1 to P

    Mate:
    SmartChromosome i selects m mates
    randomly from the population.

    Learn by Inspection:
    SmartChromosome i inspects all m mates
    and updates its knowledge based on the
    observed schemata and other information
    the mates may have accumulated.

    Learn by Interrogation:
    SmartChromosome i uses knowledge encoded
    in the feature detectors of one or more
    of the m mates and its own to modify
    oldString.

    Mutate:
    SmartChromosome i applies mutation
    operator to oldString to get newString.

    Evaluate:
    SmartChromosome i evaluates newString.

    Learn from Experience:
    IF (newString is fitter than oldString)
       THEN SmartChromosome i updates its
              knowledge based on newString

       ELSE  SmartChromosome i reverts to
              oldString and updates knowledge
              based on the failed attempt
```

Figure 2: CLGA Pseudocode

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 ${}_N C_k$. The number of SmartChromosomes (P) 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,

$$P = \left\lceil \frac{r_c \; {}_N C_k}{d} \right\rceil \qquad 2.1$$

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. Note that feature detectors *within* a SmartChromosome (in this CLGA) do not overlap. Initial evaluated. The following paragraphs explain the chromosome strings are generated randomly and pseudocode headings in the FOR loop of figure 2.

Mate. 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*.

Learn by Inspection. 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. Figure 1 depicts the feature detectors and corresponding "observed" states for a case where one SmartChromosome(A) inspects another SmartChromosome(B).

Learn by Interrogation. A SmartChromosome **interrogates** its mates by combining 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 integrated 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. Figure 3 follows through an example where SmartChromosome(A) interrogates SmartChromosome(B).

Mutate. Once the string has been modified, a mutation operator is applied to the resultant chromosome, as an "insurance policy" against loss of diversity in any one of the symbol positions.

Evaluate. Finally, the SmartChromosome evaluates its string using the fitness function.

Learn from Experience. If the new string fitness is the same or higher than the previous, the SmartChromosome inspects the new string and updates its feature detectors accordingly. However, if the new string fitness is worse, the SmartChromosome restores its previous string and inspects *it*, thus reinforcing the symbols observed in the original string. In effect, the offspring competes with its parent.

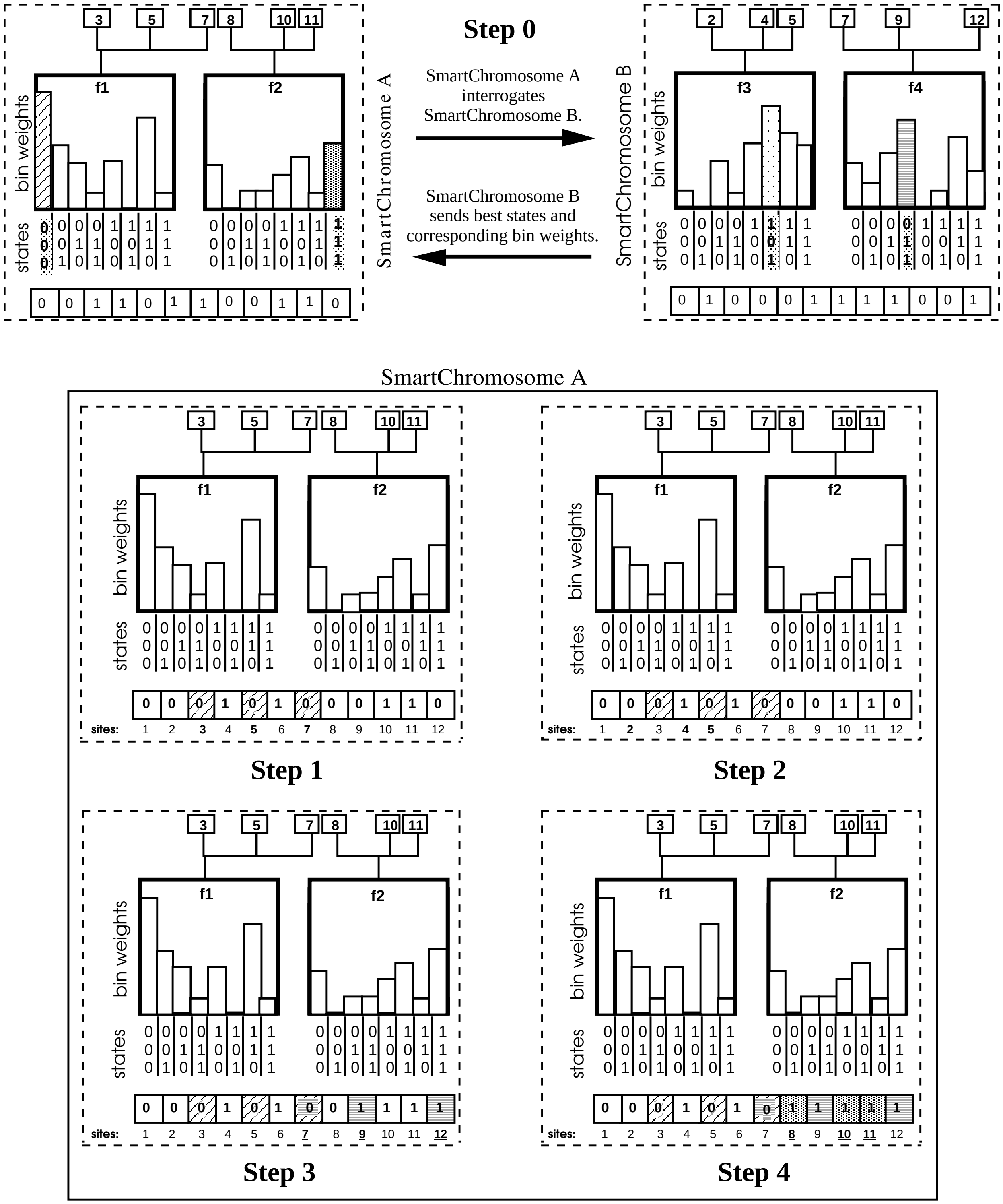

Figure 3: SmartChromosome A learns by interrogation. **[Step 0]** SmartChromosome A instructs SmartChromosome B to send the best state from each of its feature detectors (f3 and f4) and corresponding bin weights (shaded). SmartChromosome A combines its best states (shaded) with those from SmartChromosome B and sorts them in descending order by bin weight to obtain the following list: f1[000], f3[101], f4[011] and f2[111]. SmartChromosome A uses this list to modify its chromosome string through steps 1-4 as follows. **[Step 1]** The best state with the largest bin weight ([000] from f1) causes SmartChromosome A to replace the symbols at chromosome sites 3, 5 and 7 (the sites monitored by f1) with the symbols [000]. **[Step 2]** The best state with the next largest bin weight ([101] from f3) is omitted. The symbols [101] cannot replace those at chromosome sites 2, 4 and 5 (the sites monitored by f3) because the symbol for chromosome site 5 is different from the symbol already replaced at chromosome site 5 in step 1. **[Step 3]** SmartChromosome A replaces the symbols at chromosome sites 7, 9 and 12 with the symbols [011] (the best state from f4). The symbol to be replaced at chromosome site 7 is the same as the symbol already replaced there in step 1 so the replacement is allowed. **[Step 4]** Finally, SmartChromosome A replaces the symbols at chromosome sites 8, 10 and 11 with the symbols [111] (the best state from f2).

Relevant parameters of a CLGA are: the number of feature detectors (d), feature detector size (k), number of mates (m), mutation rate (μ) and combination ratio (r_c) (or indirectly, population size). In the CLGA algorithm, population size is related to the other parameters by equation 2.1.

The most important feature of the CLGA is believed to be its use of learned chromosome feature characteristics for guiding recombination, suggesting the capacity for tailoring recombination to the problem representation at hand. If this were so, its greatest advantage would be at solving highly epistatic, non-separable problems, where bits are highly interdependent. We therefore present the following hypothesis:

Primary Research Hypothesis: A CLGA will outperform an SGA on highly epistatic, non-separable problems, a class of problems traditionally difficult for regular genetic algorithms.

3 PRELIMINARY EXPERIMENTS

3.1 FIXED PARAMETERS AND CONDITIONS

Clearly, there are a number of critical parameters in this algorithm; a comprehensive investigation of them however, is beyond the scope of this paper. To introduce the algorithm we have fixed the values of the CLGA parameters to demonstrate the potential usefulness of this approach. These parameters were not optimized, but were selected for reasons made clear in the discussion in Section 4. For all experiments, the following fixed parameter values were used: d=4, k=3, r_c=1.0, m=1 and mutation rate μ=0.001, resulting in a population of 1015 SmartChromosomes. The CLGA was compared with a random CLGA with the same parameter values but with the best states selected randomly, and against a standard textbook GA (SGA). The SGA is a generational GA with scaled proportional selection using a two-point crossover and mutation. Fairly standard settings using a crossover rate of 0.6 and a mutation rate equal to the reciprocal of the chromosome length were used, along with population sizes of 50, 200 and 1016 (to match the CLGA).

3.2 FACTORS

The initial experiments varied only one factor: the degree of problem epistasis. Several problem generators have been developed to facilitate the design of more controlled experiments for testing evolutionary algorithms [DeJong et al., 1997]. For our preliminary experiments, we have selected an NK-Landscape problem generator. Each algorithm was run on 50 different problems of low, medium and high epistasis, generated randomly using an NK-Landscape problem generator, following an experiment methodology outlined in DeJong et al.(1997). In the NK model of fitness landscapes, 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, the neighborhood model was used, meaning that for a given locus, the set of K linked genes are immediately adjacent to each other, and for the purposes of generating links, the chromosome is circular. N was fixed at 30 with K set to 5 for low epistasis, 10 for medium epistasis, and 15 for high epistasis.

3.3 PERFORMANCE METRICS

Performance is measured by the best-so-far string fitness. Average performance over all random problems up to and including the current generation was computed and plotted for each generation. All experiments were run for 4000 generations (or the equivalent for smaller populations).

3.4 RESULTS

Results from the preliminary experiments are shown in figure 4. For clarity, SGA results are only shown for a population size of 1016, because there was no significant difference between the different population sizes. The vertical hi-lo bars represent 95% confidence intervals based on results for the 50 random problems.

Clearly, the CLGA performs better than the random CLGA for all levels of epistasis after approximately generation 200, suggesting a significant advantage of intelligent recombination over random exchange of chromosome bits. As expected, there is no significant difference between the CLGA and the SGA for low epistasis problems, because the low number of gene interactions corresponds to a relatively easy problem for an SGA. As epistasis increases, the difference between CLGA and SGA performance increases dramatically. Not only does the CLGA reach a higher best string fitness after 4000 generations, it also achieves higher string fitness in fewer generations. It is interesting to note that the difference between the SGA and the random CLGA for greater degrees of epistasis is slight, suggesting that the two-point crossover is just as disruptive as the random exchange of chromosome bits with respect to problems with high epistasis.

The relatively high standard deviation for K=15 after generation 2000 is actually very good. In several cases, the CLGA arrived at a best-so-far solution that was significantly better than any obtained by the SGA. The above-average outliers result in a much wider distribution.

Note that all 50 problems for each algorithm were initialized to the same initial population. To determine the effect of initial conditions on performance, an additional

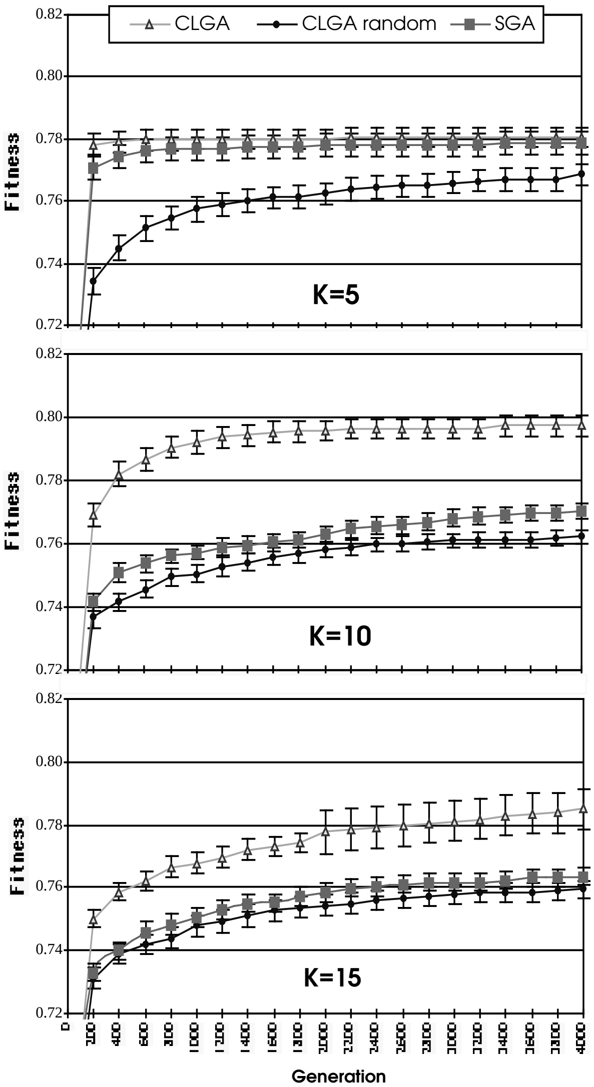

Figure 4: Average performance plots for the CLGA, the random CLGA and the SGA over 50 random problems for low, medium and high epistasis. 95% confidence intervals are shown.

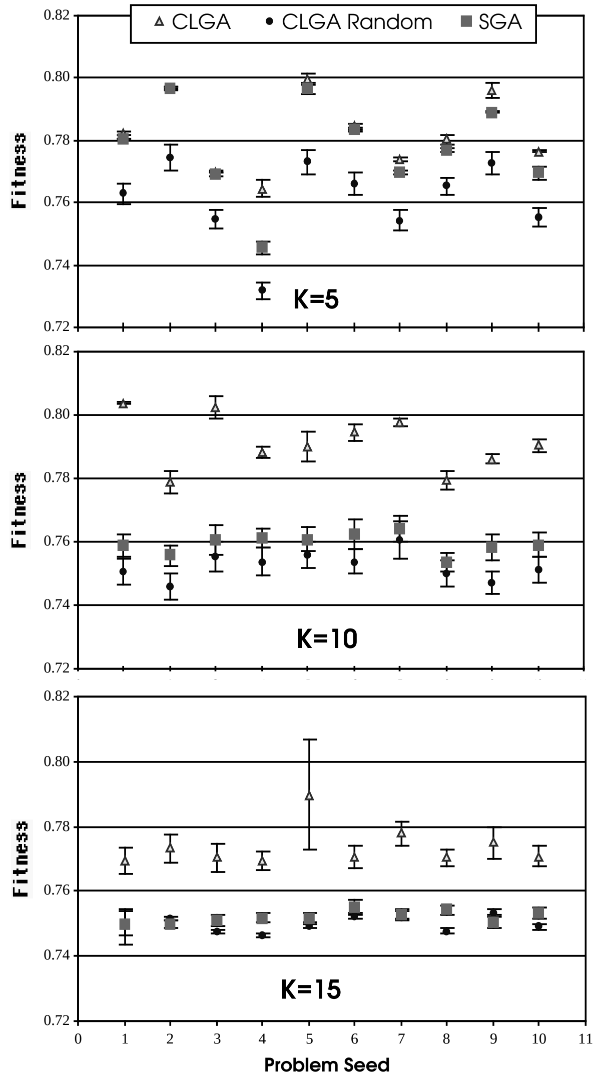

Figure 5: Average performance after 1200 different generations for 10 random problems using 30 random populations for low, medium and high epistasis. 95% confidence intervals are shown.

set of experiments was run for 10 random problems using 30 different initial populations. 1200 generations were judged to be sufficient for comparing results. Figure 5 shows results that are consistent with the conclusions drawn from the previous experiments. Again, the large standard deviation for problem seed 5 for K=15 is due to the CLGA discovering very good solutions (>0.93) early in the evolution in 3 of the 30 initial random populations.

4 CONCLUSIONS AND FUTURE WORK

Preliminary results are encouraging. Obviously, a great deal of work lies ahead in accurately evaluating the merits of the CLGA approach. Although initial results involved only a single set of parameter values for the algorithm, several points in regard to this initial choice must be made.

First, why should a set of feature detectors of size 3 be sufficient to capture higher order bit interdependencies? We don't contend that they are. Three was chosen initially because it resulted in manageable resource requirements and seemed like a reasonable place to start. However, our preliminary results are interesting enough to warrant further investigation of the conditions under which this may be a possibility. The fact that fairly good results were obtained over a large range of epistasis suggests that the order of the schemata monitored may not have to be nearly as large as the expected degree of epistasis in order for the CLGA to successfully *disentangle* the epistatic interactions.

Second, informal experiments with the CLGA suggested that only about 3/4 of each chromosome string should be modified using the combined knowledge of the interacting SmartChromsomes during interrogation. For example, in this paper, interrogation resulted in two SmartChromosomes combining the information of a total of eight feature detectors affecting a maximum of 24 chromosome sites (80% of the sites). Intuitively, this may be reasonable as there may need to be some balance between the amount of knowledge used to guide recombination versus the information kept from the currently held best string. This, however, will need to be further investigated.

The combination ratio for this experiment was chosen to be 1.0, meaning that all combinations of feature detectors were integrated into the population. Clearly, the choice to include all was a safe decision, enabling all order-3 schemata to be monitored. However, it would be interesting to see how small this ratio can get before significant degradation of performance occurs.

Only one mate was selected for each SmartChromosome. It is possible that inspection and interrogation of more than one mate may yield a faster dissemination of information, and a subsequent increase in convergence speed, but this has yet to be investigated.

Finally, a mutation rate of 0.001 was chosen, a value smaller than commonly used for chromosomes of this size. Given the specifics of the algorithm, it is clear that some method of "insurance" against a loss of diversity in the chromosome sites is necessary. Mutation happens to be a reasonable one. Again, informal experiments suggest that mutation needs to be rather low perhaps to avoid significant disruption of the benefits achieved by the more intelligent recombination.

The question of why the CLGA works is still a matter of speculation, but work is progressing. First, it is clear that some explicit estimation of average schema fitness is being done, guiding the population in a manner similar to standard GA selection. The difficulties associated with potentially high schema fitness variance still exists; however, the variances associated with these averages are now accessible, which gives us a valuable tool to use to interpret the relationship between these measurements and quality of a schema's fitness contribution. Second, because a SmartChromosome retains its best solution after each generation, it essentially conducts a search on its own, following an avenue of search biased weakly by the chromosome sites monitored by its feature detectors and ultimately determined by other SmartChromosomes it encounters. Since every SmartChromosome has a unique set of feature detectors, every SmartChromosome has a weak bias different from every other, exploring different subsets of hyperplanes in the fitness landscape. A SmartChromosome mixes and matches subsets of order-3 schemata to try and build a higher order schema, which, if it results in a superior string, is guaranteed to be preserved under the elitist selection policy. The new string is then observed by other SmartChromosomes during inspection which affects their "opinion" of what the best bits are for the schemata *they* are monitoring.

There are several issues of concern:

1. Of primary concern is the scalability of our approach. If the size of the feature detectors must increase significantly to deal with greater degrees of epistasis, the population size may become impractical even for modest sized problems. Though we are not entirely convinced this must be so,. one possible solution might be to decrease the combination ratio to reduce the total number of feature detector combinations. Another would be to allow feature detector overlap within SmartChromosomes. The former may degrade performance, while the latter would increase the computational burden on each SmartChromosome. Both, however, would have the effect of decreasing population size.
2. The computational resources required for this algorithm are significantly larger than for a regular GA, since what is a simple structure in a regular GA is now (at the software level) an entire object, with its own functions and memory. However, the algorithm can be implemented entirely in parallel, which may make this concern a moot point as parallel processing systems become more readily available. Conversely, a serial implementation may still be competitive if the bottle neck is the evaluation of the fitness function, or if, as our preliminary experiments suggest, the algorithm is general enough to effectively deal with both epistatic and non-epistatic problems.
3. The problem of premature convergence for the population is now shifted in emphasis from the population strings to the CLAs. The CLAs of each SmartChromosome become entrenched in their "opinion" on what are the best bit configurations given the observations they have made. The full ramifications of this are not yet clear.
4. It is also not yet clear how sensitive the CLGA is to its parameters, or how the parameters directly relate to its performance. At first glance, the number of parameters may seem large, but constraints and relationships between parameters limit the number of practical combinations.

Work is continuing with experiments using other problem generators and a more thorough investigation of CLGA parameters. Several interesting avenues of research are also being investigated. Because each SmartChromosome is capable of remembering its interactions, it can adapt more readily to the fitness landscape as it discovers and explores the space in conjunction with the other members of the population. For example, Hamming distance and fitness differential might be correlated over its interactions to give the SmartChromosome some idea of the ruggedness of the fitness landscape. In this way, it may be able to say, increase or decrease its own mutation depending on whether it estimates the fitness landscape to be rough or smooth. Mutation may also be made "smart" by limiting it only to chromosome sites for which the SmartChromosome has not obtained information via interrogation.

The CLGA may provide an additional way of attacking deceptive problems. Every SmartChromosome inspects schemata in the same way. Observing a "bad" schema is no worse than observing a "good" schema because all information is good information: what matters is how the information accumulated in the feature detectors is used. Consequently, the goals of SmartChromosomes need not be the same, *i.e.*, a sub-population of SmartChromosomes whose goal is to discover the worst solution can co-exist with a sub-population of SmartChromosomes whose goal is to discover the best solution! As a result, SmartChromosomes could be made to search both "ends" of the solution space to increase the probability of discovering good solutions to deceptive problems. A SmartChromosome looking for a very bad solution, may, in the case of deceptive problem, stumble upon a very good solution and then switch its allegiance to look for good solutions.

The nature of the algorithm also suggests that the architecture of the CLGA *i.e.* the placement and structure of the feature detectors may be tailored to the fitness function. For example, for real-valued parameters, feature detectors may be designed to monitor only similarly significant bits, enabling a multi-scaled problem solving approach. Clearly, much work remains to be done, but results are very encouraging.

Acknowledgements

The authors would like to acknowledge the use of Mitchell A. Potter's code for the NK-Landscape Model. We would also like to thank Mona Diab for valuable discussions in the early stages of this work.

References

Baluja, S. (1994).Population-based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Tech. Report No. CMU-CS-94-163. Pittsburgh, PA: Carnegie Mellon University.

Baldwin, J.M. (1896). A New Factor in Evolution. *Amer. Naturalist*, 30:441-451.

Bock, P. (1993). *The Emergence of Artificial Cognition*. World Scientific Publishing Co.

Cobb H. G. (1993). Is the Genetic Algorithm a Cooperative Learner? In L.D. Whitley, ed., *Foundations of Genetic Algorithms 2*. Morgan Kaufmann.

DeJong, K.A., Potter, M.A., and Spears, W.M. (1997). Using Problem Generators to Explore the Effects of Epistasis. In T. Back ed., *Proc. 7th Int. Conf. on Genetic Algorithms*. Morgan Kaufmann.

Goldberg, D.E., Korb, B., and Deb, K. (1989). Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems*, 3:493-530.

Harik, G. (1997). Learning Linkage to Efficiently Solve Problems of Bounded Difficulty Using Genetic Algorithms. IlliGAL Technical Report No. 97005. Urbana, IL: University of Illinois at Urbana-Champaign.

Harik, G. (1999). Linkage Learning via Probabilistic Modeling in the ECGA. IlliGAL Report No. 99010. Urbana, IL: University of Illinois at Urbana-Champaign.

Hinton, G.E., and Nowlan, S.J. (1987). How Learning Can Guide Evolution. *Complex Systems* 1:495-502.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press. (Second edition: MIT Press, 1992).

Kargupta, H. (1995). SEARCH, Polynomial Complexity and the Fast Messy Genetic Algorithm. IlliGAL Technical Report No. 95008. Urbana, IL: University of Illinois at Urbana-Champaign.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.

Muhlenbein, H. and Paaß, G. (1996). From Recombination of Genes to the Estimation of Distributions. I. Binary Parameters. *Parallel Problem Solving from Nature IV*: 178-187.

Pelikan, M., Goldberg, D.E. and Cantu-Paz, E. (1999). BOA: The Bayesian Optimization Algorithm. IlliGAL Report No. 99003. Urbana, IL: University of Illinois at Urbana-Champaign.

Reynolds, R.G. and Chung, C. (1997). Regulating the Amount of Information Used for Self-Adaptation in Cultural Algorithms. In T. Back, ed., *Proc. 7th Int. Conf. on Genetic Algorithms*. Morgan Kaufmann.

Riopka, T.P., Diab, M., and Bock, P. (1997). Quantifying and Interpreting the Effect of Intelligent Information Exchange Between Chromosomes in a Human Simulation of a Genetic Algorithm. Unpublished GWU Technical Report. Washington, DC: George Washington University.

Schaffer, J.D. and Morishima, A. (1987). An Adaptive Crossover Distribution Mechanism for Genetic Algorithms. In J. J. Grefenstette, ed., *Proc. 2nd Int. Conf. on Genetic Algorithms and Their Applications*. Erlbaum.

Smith, J. and Fogarty, T.C. (1996). Self Adaptation of Mutation Rates in a Steady State Genetic Algorithm. In *Proc. 3rd IEEE Conf. on Evolutionary Comp*. IEEE Press.

Spears, W.M. (1995). Adapting Crossover in Evolutionary Algorithms. In *Proc. 4th Annual Conf. on Evolutionary Programming*. MIT Press.

Thierens, D. and Goldberg, D.E. (1993). Mixing in Genetic Algorithms. In S. Forrest, ed., *Proc. 5th Int. Conf. on Genetic Algorithms*. Morgan Kaufmann.