A Method for Utilizing Diploid/Dominance in Genetic Search

World Conference on Computational Intelligence (1994) IEEE (pp. 439-444)

F. Greene
Department of Electrical Engineering
University of Washington
Seattle, WA 98195
bgreene@accessone.com

Abstract

A method is proposed for implementing diploid/dominance in genetic search. Both diploid strings, in a single gene example are evaluated to produce two intermediate phenotypes which then determine dominance. Experimental results indicate that changing global optima are accurately identified and followed. The method is problem independent and can be readily extended to a polyploid chromosome structure.

Introduction

Genetic algorithm research has largely involved the use of single chromosome *haploid* individuals in a population of individuals which are subjected to a variety of selection and crossover operations. In biology, however, most higher organisms utilize a *diploid* chromosome structure. This structure is sufficiently common that investigations into its use have been done by several investigators including Holland (1975) and Goldberg (1989).

Diploid/dominance expression in biology suggests that two homologous alleles are paired and, through some mechanism, the dominant gene is phenotypically expressed. If, and only if, both genes are *recessive* then the recessive phenotype is expressed. In Hollstien (1971), a genetic algorithm ("GA") was used to implement this logic by increasing the cardinality of the chromosome bit string from 2 to 3. Hollstien then added a "triallelic" third value ("2") to the usually 2-valued ("0" or "1"), at each binary position. If a "2" occurs, it is phenotypically interpreted as a "1", but only if paired with a "1". The result is that a diploid allele can act as either a dominant ("1") or as recessive ("0"). This approach made it possible to study dominance in multiple allele chromosomes (every bit position in the string is a separate gene with corresponding allele value).

Sub-Phenotype Interactions

In contrast, this paper considers the diploid chromosome as two chromosomes with one gene each, for simplicity, and one corresponding allele value each. The allele values are defined to be the fitness values of the two strings. These two *sub* fitness values are then utilized as if they were actually representing an intermediate phenotype-level between genotype and phenotype. This process, where two homologous alleles seemingly compete prior to manifestation in the observed phenotype, will be referred to as *sub-phenotype* interaction.

The two sub-phenotypes are obtained by evaluating both chromosomes using whatever fitness function is to be maximized. Dominance is implemented by mapping the two sub-phenotype fitness values to the scalar fitness value which will then be used for selection. The mapping function will be referred to as the *dominance map* or *dominance function*, and the fitnesses of the individual haploid chromosomes (homologues), will be referred to as *sub-fitness*. The approach used here is based on zoology and genetics, as will now be described. The actual algorithm will be described in the methods section.

Gene-Enzyme Dominance

In biology, it has been postulated Muller (1950) that homologous genes can interact in a non linear fashion to produce their associated phenotype. If the dominant form of an allele produces enough of the enzyme needed for, e.g., eye coloration (Muller, pg. 179) then a homozygous combination of dominants (i.e., both alleles are dominant) will produce only a barely noticeable increase over the heterozygous combination. This idea is summarized in Crow (1983) as follows. "...that genes produce enzymes provides an explanation of dominance." [Consider that C is the dominant allele and c is the recessive form of that allele] "Since only very small amounts of enzymes are needed to catalyze chemical reactions, one gene usually produces more than enough enzyme to convert all the substrate into product.... Since one C allele is sufficient, the genotypes CC and Cc both have the same amount of pigment even though CC produces twice as much enzyme as Cc." One consequence of the dominance mechanism occurring at a level between the string level (genotype) and fitness level (phenotype) is that homologous genes need not be explicitly linked (e.g. through gene position), they simply compete through the similarity of their intermediate (enzyme) phenotype.

Methods

Test Function

The experimental approach in this paper follows a portion of Smith (1992) wherein the Hollstein triallelic mechanism

was used to study the ability of diploid/dominance to improve GA performance with changing global optima. The function they attempted to optimize is known as a "non-stationary knapsack problem." This is an NP-complete problem with a search space of 2^L, for an L-bit chromosome. The present work uses this same approach, not only because it provides direct comparison with a similar method, but also because of the moderate difficulty of the problem. The 0-1 knapsack problem is defined as follows:

$$\text{maximize} \quad \sum_{i=1}^{L} v_i x_i = f(\mathbf{x}) \quad (1a)$$

$$\text{given} \quad \sum_{i=1}^{L} w_i x_i \le W \quad (1b)$$

where: $x_i \in [0,1]$ are the L chromosome bits

v_i and w_i are real, pre-defined weights

This function seeks to maximize the inner product of **v** with the binary chromosome, **x**, imposing the constraint (1b) that the inner product of **w** with the chromosome must be less than a fixed constant W. The above authors then modified the knapsack problem, in order to effectively induce a *shift* in the fitness environment, by periodically switching the value of W between two different values.

Environment Shift and Non-Stationarity

Once again referring to biology, diploidy is suspected of providing protection against low-fitness allele loss. "The preservation in diploid populations of alleles that may be selectively disadvantageous under some conditions provides a source of evolutionary variability not available to haploids Novikoff (1970)."

If a similar environment repeatedly arises, diploidy could presumably enable the species to rapidly re-express its former fitness for that environment. A shift in environment, or non-stationarity, was introduced into Smith's GA experiment by periodically switching the value of "W" every 15 generations. In that paper, complete specification of the **v** and **w** vectors, as well as other experimental details, were provided that are sufficient to duplicate their knapsack experiment using the approach about to be described.

Use and Modification of the Genitor Algorithm

The Genitor GA algorithm Whitley (1989) is used here as it has proven useful for on-going experiments and is easily modified. Genitor was modified to compile under C++ using Borland C++ 3.1 for DOS, in order to take advantage of object definition.

Genitor must ordinarily evaluate and replace only one individual per generation. This behavior was modified slightly, so that when W changes in eq. (1b), the *entire* population is re-evaluated. Such a step is unnecessary with more standard *generational* techniques Syswerda (1991), since these algorithms evaluate the entire population every generation. All remaining major modifications to Genitor are listed in Table 1, and will be discussed in context with their use.

Modifications to the Smith-Goldberg Experiment

In Goldberg (1991), it is suggested that a comparison of execution rates between Genitor and a simple, *generational*, GA calls for on the order of M generations of Genitor to get comparable computational power to 1 generation of a simple GA, where "M" is the population size. For this reason, the number of generations between environmental switching, as presented here, is increased somewhat from 15 (as used by Smith) to 200. Switching W every 15 generations also proved successful, as shown in the results section.

Genitor permits specifying a "b" parameter (for "bias", which is the fitness ratio between the most highly ranked individual and the median individual), instead of the usual crossover probability, p_C. In addition, crossover always occurs *exactly once* every generation with Genitor. Selection pressure can therefore be controlled by the value of "b". Smith used a fairly large crossover probability ("p_C") of .75). The bias value used in this paper is the Genitor default (maximum for linear ranking), of b= 2.0, except for cases where very low mutation rates are tried, where the value used is b= 1.2. The intention is to ensure a large amount of selection pressure where possible (comparable to p_C= .75), and to demonstrate that results obtained are not overly sensitive to the specific values used for fitness pressure and mutation rate (see figures 2-5, results section.)

Smith provides the optimum values of f(**x**) for two values of W. These were f(**x**)= 87 for W= 104, and f(**x**)= 71 for W= 60, and are the values used here as well.

Fitness Function Modification for Genitor

Genitor is designed so that fitness increases with *decreasing* objective function value, with a fitness of zero usually being interpreted as perfectly correct. In order to *maximize* f(**x**) (as opposed to training the system to produce a known correct value), and since the fitness function (1a) increases (from zero), the Genitor fitness function is redefined to be $f_S(\mathbf{x})$= 200 - f(**x**), where the "s" subscript stands for (Genitor) *sub-phenotype*. The f_S of the individual homologues are evaluated, and used to calculate diploid fitness by Genitor, as will be described. There are two optimum values of f_S= 113= 200-87, for W= 104, and f_S= 129= 200-71, for W= 60. The values 113 and 129 are therefore the two solutions that the GA should ideally converge to, remain at, and switch between. If f(**x**) is negative, then $f_S(\mathbf{x})$ is set to floating point infinity (giving it minimum possible fitness, since Genitor uses *rank* allocation of trials). As in Smith (1992), a string length of 17 is used, the elements of which define the values of the **x** vector. The result is a modestly difficult search space of 2^{17}.

Genitor	Here
Written in "C"	Converted to C++
Haploid chromosome structure	Diploid chromosome C++ object (compatible with Genitor)
A single individual is evaluated and replaced per generation	Same, except entire population is re-evaluated following an environment switch (every 200 generations)
2-point crossover -- only differing bits get crossed	Same, except as in simple GA, crossover occurs at any bit position
Copying one gene to another involves 1 chromosome string + fitness value, and is done with the Genitor function "ga_copy()"	The Genitor ga_copy() function has been augmented to copy both diploid strings, two sub-fitness values, and pointers to the dominant homologues
Real number capability, but usage left up to user.	Arithmetic crossover implemented (see methods section).

Table 1. Summary of Genitor modifications.

Diploid Implementation

Individual members of the (diploid) population are represented using a C++ object or *class*. The class is designed to be backward compatible with the Genitor structure that defines a haploid chromosome. Its associated "member" functions handle memory management and haploid and diploid fitness evaluation. The only other Genitor modification is the chromosome copying function (named "ga_copy"), which is augmented to handle both diploid strings. Initialization of the diploid population is done at the beginning of the GA run by randomly assigning values to the first homologue of each individual, and then replicating those values in the second homologue.

Diploid/Dominance Fitness Evaluation

The diploid evaluation function used in this paper to implement dominance mapping simply identifies the homologue with maximum fitness and uses that value for the diploid fitness. That is:

$$f = MAX(f_1, f_2) \qquad (2)$$

where: f is the individual's fitness (to be used for selection purposes -- in this case, eq. 1a),

$f_1 = f(\mathbf{x_1})$= fitness of the 1st homologue, $\mathbf{x_1}$, and

$f_2 = f(\mathbf{x_2})$= fitness of the 2nd homologue, $\mathbf{x_2}$.

Clearly other functions are possible. Function (2), however, has some properties that are useful:

- *dimensional consistency:* the diploid fitness has the same units as the homologue sub-fitnesses;
- *shielding of the recessive allele:* a reduction in fitness of the recessive homologue has no (immediate) effect on the diploid fitness value, thereby preventing such an event from affecting the individual's diploid fitness. This may be important if a previously optimal genotype is to be retained throughout a lengthy or radical, change in environment;
- *identification of global optima:* if either homologue has achieved the global fitness optimum, then the diploid fitness will also be at a global optimum.

Diploid Crossover

The method described here differs from the analogous process in biology, as it has thus far given much more consistent results. In biology, crossover occurs within the homologues of each parent *prior* to pairing a homologue from each parent to form the offspring. Here, crossover occurs *between* homologues that are randomly selected from the two parents to produce the two chromosomes that make up the child. Specifically:

1) Select the 1st homologue from parent 1 and cross it with a randomly selected homologue from parent 2,
2) Select the 2nd homologue from parent 1 and cross it with a randomly selected homologue from parent 2,
3) Create the child using the chromosomes produced by steps (1) and (2),
4) Apply random mutation to each of the diploid child chromosomes, according to the mutation rate, p_m,
5) Evaluate the fitness of each child chromosome,
6) Apply (eq. 2) to get the child's diploid fitness,
7) Insert the child into the population, according to its diploid fitness.

This algorithm uses two evaluations per generation, compared with standard Genitor, which uses one.

Arithmetic Crossover

The crossover method used in the test (below), involves real valued chromosomes as is described in Michalewicz (1992). Two crossover points are selected randomly, k and m, from a string of length L. The two offspring are then produced at time t+1 from two parents as follows:

if real chromosomes $\mathbf{V}^{\mathbf{t}}$ and $\mathbf{W}^{\mathbf{t}}$ are crossed at t and $0 \le a \le 1$, $a \in$ Reals, the resulting offspring are

$$\mathbf{V}^{\mathbf{t+1}} = \left\langle \begin{matrix} v_1 \ldots v_k, w_{k+1} \cdot a + v_{k+1} \cdot (1-a), \ldots, \\ w_m \cdot a + v_m \cdot (1-a), v_{m+1} \ldots v_L \end{matrix} \right\rangle$$

$$\mathbf{W}^{\mathbf{t+1}} = \left\langle \begin{matrix} w_1 \ldots w_k, v_{k+1} \cdot a + w_{k+1} \cdot (1-a), \ldots, \\ v_m \cdot a + w_m \cdot (1-a), w_{m+1} \ldots w_L \end{matrix} \right\rangle$$

As suggested by Wright (1991), $a > 1$ is permitted (i.e., the chromosome solution space is no longer strictly convex). The value of a is defined globally, and is swept in a triangular wave fashion between .5 and 1.5 at a rate of $^{+}/_{-}$ .01 per generation. The bits of the vector $\mathbf{x}$ in (1a) are set or cleared depending on the sign of the corresponding elements of the chosen offspring ($\mathbf{V^{t+1}}$ or $\mathbf{W^{t+1}}$). No mutation (p_m= 0.0) is used.

Results

The experiments described in this section use

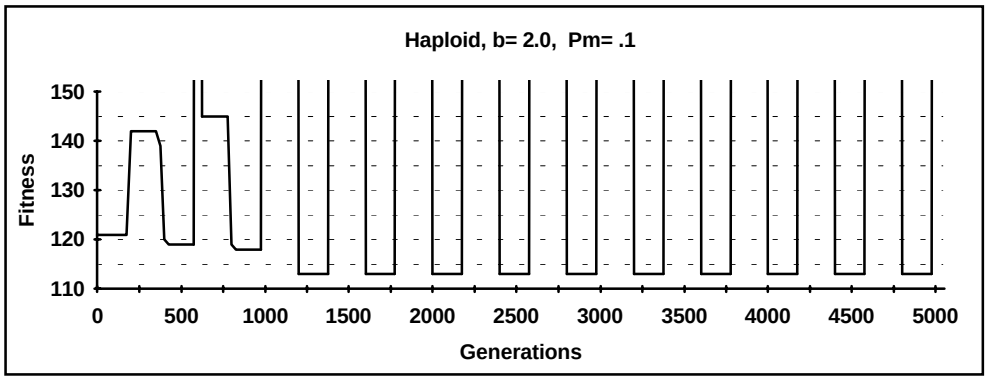

Figure 1. Haploid, non-stationary knapsack problem, W switched every 200 generations.

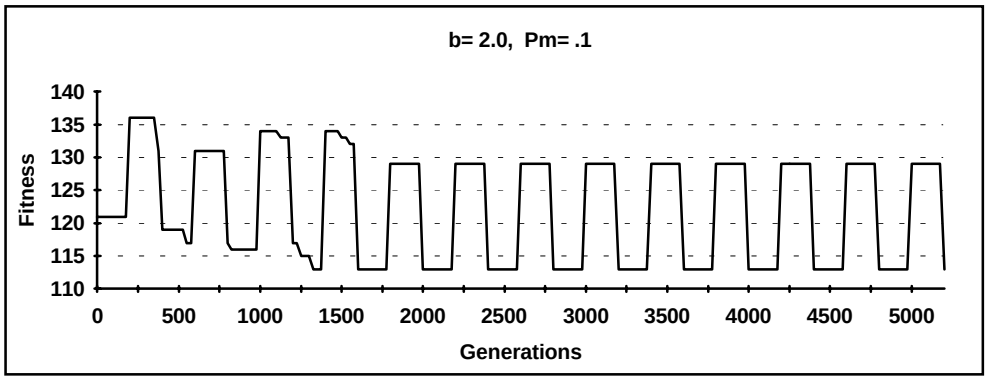

Figure 2. Diploid, Non-Stationary Knapsack Problem, W switched every 200 generations.

diploid/dominance as previously described. There is one figure per experiment. For each figure, the vertical scale is the Genitor fitness value, $f_s(\mathbf{x})$, which *increases* from zero with *decreasing* fitness value, $f(\mathbf{x})$. The *best* individual of each generation is plotted vs. generation. X-axis plotting resolution is 25 generations, except for the indicated portion of figure 4, which is one generation.

Figure 1 shows results for a conventional haploid chromosome implementation. By 1200 generations *one* environmental constraint, W= 104, was correctly identified, having fitness= f_s= 113. However the other state is never identified and the fitness value in the population past t= 1000, for W= 60 is equal to floating point $+\infty$ (which Genitor interprets as minimum fitness). The remaining figures in this section use the same values for W and (resulting f_s).

The remaining results utilize diploid/dominance, for which the GA readily converges to correct f_s values. Once convergence has occurred, subsequent switches in W always result in an immediate change in fitness to the appropriate f_s value within *one* generation, remaining constant until the next change in W.

Figure 2 uses the same values for b and p_m, as figure 1, but now a diploid chromosome is used and as can be seen, the GA converges to the two values of 113 and 129 by 1800 generations. Switching between fitness values occurs immediately to the correct fitness values and remains at those values until the next environment switch. This latter statement is true of for all remaining results. There *are* combinations of p_m and b for which complete convergence to both environments never occurs, such as b= 2.0 and p_m from 0.0 to 0.001, apparently due to premature convergence. Figure 3(a,b) shows results for differing values of bias and $p_m > .001$ that produce stable and accurate solutions. In fig. 3a), b=2.0 is utilized with a large mutation rate of p_m= .5. Convergence to the correct fitness values still occurs, but now requires 5000 generations. Figure 3 b) has minimal p_m and b is reduced to prevent premature convergence, converging in 4200 generations. Figure 4 uses a faster switching rate, with convergence time being comparable (t=1875) to the longer switching period of figure 2. In figure 5, the environment is alternately switched every 100 and 200 generations and the diploid GA converges to the correct solution in 2600 generations. In figure 6 arithmetic crossover, which uses floating point strings, was implemented. No mutation was necessary to achieve a stable and accurate solution in approximately 2600 generations.

Conclusions

The proposed algorithm for implementing dominance/diploidy easily succeeds in locating alternating

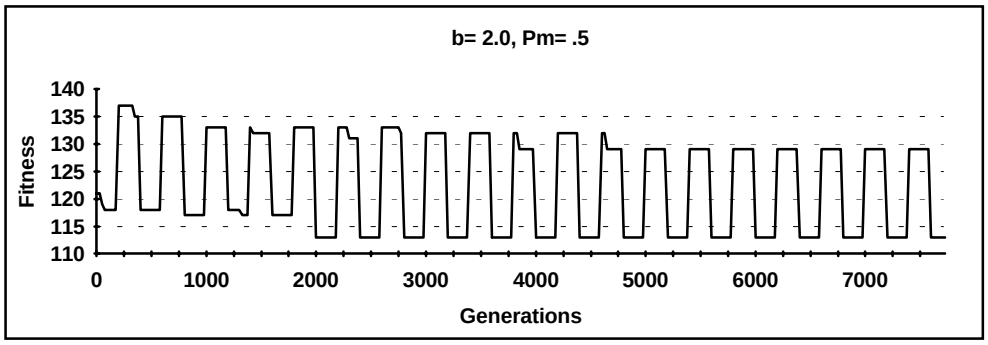

Figure 3a. Diploid, non-stationary knapsack problem, W switched every 200 generations

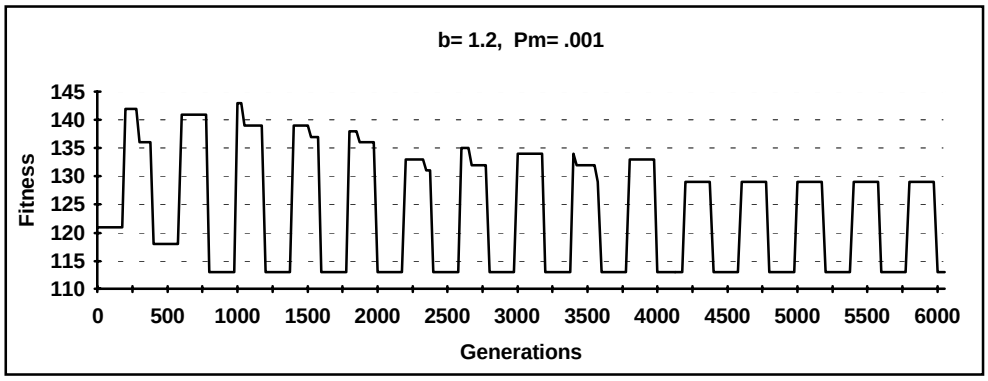

Figure 3b. Diploid, p_m= .001, b reduced to prevent premature convergence.

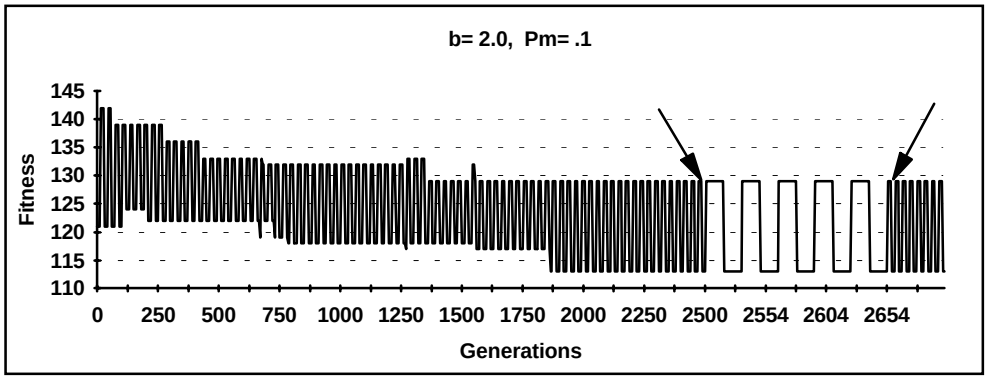

Figure 4. Diploid, W switches every 15 generations, arrows indicate magnified scale

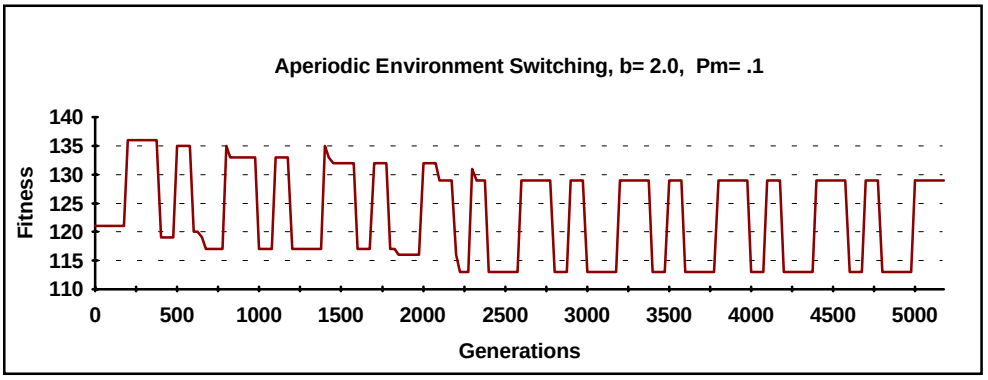

Figure 5. Diploid, W switches alternately every 100 and 200 generations.

global optima. Results with the non-stationary 0-1 knapsack problem, similar to Smith (1992), indicate that convergence occurs over a wide range of p_m, fitness pressure, and switching rates. Switching rise time and stability appear to be improved over those shown by Smith (pp. 265-280) using the Hollstein triallelic, multiple allele scheme. This result is due, in part, to using the (knapsack) problem's fitness function as input to the dominance function, which also makes the method fairly problem non-specific.

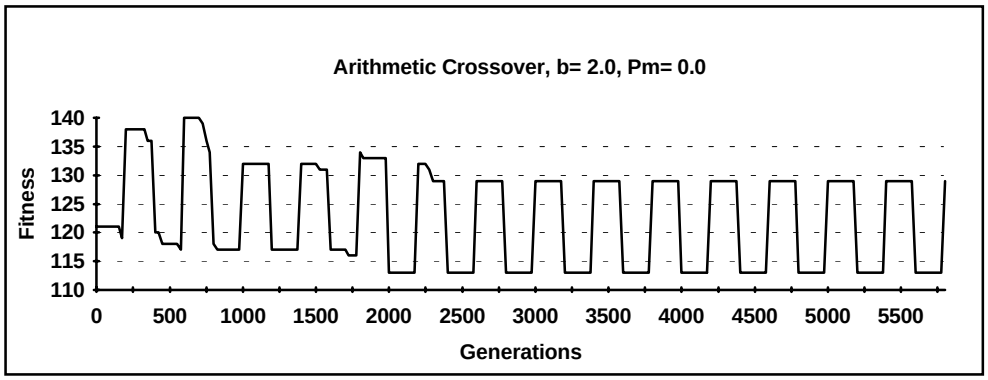

Figure 6. Diploid, W switched every 200 generations, arithmetic crossover

The proposed method is shown to function with a non binary, real encoded version of the 0-1 knapsack problem. Preliminary results indicate that the approach can be extended to polyploid/dominance (by repeating steps 1-3 under "diploid crossover".)

The approach is inspired by the interaction of genes and enzymes in biology. Intermediate or "sub" phenotype values for each diploid homologue are mapped to the individual's expressed phenotype using a straightforward evaluation function that emulates the relationship of gene/enzyme to phenotype. Extension to multiple alleles might be made, if needed, by mapping their sub fitnesses to separate phenotypes that would be subsequently mapped to a scalar fitness value according to the problem specification.

Smith (pg. 257) showed, in part, that diploid/dominance reduces the p_m needed to maintain a small, steady-state proportion of recessive alleles in a *stationary* fitness environment. This suggests that diploid/dominance might reduce premature convergence in stationary problems and should be the subject of further research (related experiments by the author are thus far inconclusive.) Another approach might be to induce non-stationarity in an ordinarily stationary problem, to guide or encourage building block growth through changes in the fitness environment.

Acknowledgments

My thanks to Dr. Joe Felsenstein for insightful information about biology and genetics.

References

Crow, J. F. (1983). Genetics Notes (8 ed.). Minneapolis: Burgess.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization & Machine Learning. Menlo Park: Addison-Wesley.

Goldberg, D. E., and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In G. J. E. Rawlins (Eds.), Foundations of Genetic Algorithms (pp. 69-93). San Mateo: Morgan-Kaufmann.

Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor: The University of Michigan Press.

Hollstien, R. B. (1971) Artificial genetic adaptation in computer control systems. Doctoral Dissertation, University of Michigan.

Michalewicz, Z. (1992). Genetic Algorithms + Data Structures = Evolution Programs. Berlin: Springer-Verlag.

Muller, H. J. (1950). Evidence of the precision of genetic adaptation. In C. C. Thomas (Eds.), The Harvey Lecture Series (pp. 165-229). Springfield, Illinois:

Novikoff, A. B., Hotzman, E. (1970). Cells and Organelles. New York: Holt, Rinehart, and Winston, Inc.

Smith, R. E., Goldberg, D.E. (1992). Diploidy and dominance in artificial genetic search. Complex Systems, 6, 21-285.

Syswerda, G. (1991). A study of reproduction in generational and steady-state genetic algorithms. In G. J. E. Rawlins (Eds.), Foundations of Genetic Algorithms (pp. 94-101). San Mateo: Morgan-Kaufmann.

Whitley, D. (1989). The Genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In Proceedings of the Third International Conference on Genetic Algorithms, (pp. 116-123). George Mason University: Morgan Kaufmann.

Wright, A. (1991). Genetic algorithms for real parameter optimization. In G. J. E. Rawlins (Eds.), Foundations of Genetic Algorithms (pp. 205-218). San Mateo: Morgan Kaufmann.