Modeling Hybrid Genetic Algorithms

Darrell Whitley

Computer Science Department, Colorado State University, Fort Collins, CO 80523
whitley@cs.colostate.edu

1 INTRODUCTION

A "hybrid genetic algorithm" combines local search with a more traditional genetic algorithm. The most common form of hybrid genetic algorithm uses local search to improve the initial population as well as the strings produced by genetic recombination. The resulting improvements are then coded onto the strings processed by the genetic algorithm (e.g. Mühlenbein, 1991; Davis, 1991). These hybrid genetic algorithms have also been called "Memetic Algorithms". The term Lamarckian Evolution is also sometimes used, where local search in this context is considered to be analogous to acquired or learned behavior that is subsequently coded back onto the "genotype" representation and passed on to offspring during reproduction.

This paper looks at how one form of hybrid genetic algorithm can be modeled in the context of the existing models for the simple genetic algorithm; it should be possible to model the integration of other types of local search with genetic algorithms using the same basic approach. A secondary goal of this paper is to review the existing models for finite and infinite population genetic algorithms and to show the relationship between the two models. The "Lamarckian" updates associated with a hybrid genetic algorithm can be represented as a matrix update to the infinite population vector used by the infinite population model of the simple genetic algorithm. In turn, the infinite population vector is also equivalent to the sampling distribution probabilities used by finite Markov models of the simple genetic algorithm. Thus, the extension of existing models to hybrid genetic algorithms is simple and straight forward.

This paper also builds on earlier work by Whitley, Gordon and Mathias (1994) by exploring the notion that the use of a hybrid genetic algorithm induces equivalence classes over the set of all possible discrete functions, such that functions in an equivalence class are in expectation processed in an identical fashion under hybrid genetic search.

The results presented here specifically focus on parameter optimization problems that have a binary encoding. The models discussed in this paper can be extended to other representations, and in particular, Whitley and Yoo (1995) show how models of simple genetic algorithms can be adapted for permutation representations.

For binary encodings, local search involves changing each of the L bits in a string encoding; if an improvement is found, the search is said to "move" to the improved string. Initially local search will be limited to one step in the search space. The arguments presented in this paper assume that each step of local search results in a deterministic choice of moves in the search space; for modeling hybrid genetic algorithms this requirement is not rigid however and the results can be reformulated to consider probabilistic choices of moves under local search. In order to make moves deterministic we use steepest ascent as our local search. For steepest ascent, all L bit changes are tested and the best improvement is taken. An alternative is to use next ascent, where the first improvement found among the L neighbors is taken. Note that if an improvement is found before checking all L neighbors then next ascent has less evaluation cost per move than steepest ascent. Also note that if next ascent checks neighbors in a fixed order, the moves it generates are also deterministic; however, in practice it is often effective to test the neighbors of a string in random order, thus making the next move a probabilistic decision.

1.1 Equivalent Classes Under Hybrid Genetic Search

Consider the following Function One:

```
e(00000) = 105   e(01000) = 73    e(10000) = 36    e(11000) = 60
e(00001) = 69    e(01001) = 71    e(10001) = 93    e(11001) = 64
e(00010) = 43    e(01010) = 89    e(10010) = 37    e(11010) = 96
e(00011) = 47    e(01011) = 43    e(10011) = 25    e(11011) = 19

e(00100) = 60    e(01100) = 44    e(10100) = 15    e(11100) = 54
e(00101) = 40    e(01101) = 29    e(10101) = 68    e(11101) = 38
e(00110) = 18    e(01110) = 48    e(10110) = 44    e(11110) = 31
e(00111) = 65    e(01111) = 91    e(10111) = 23    e(11111) = 100
```

where $e(x)$ is the evaluation of string x. This function was created by assigning the evaluation 100 to point 11111 and evaluation 105 to the point 00000 and by assigning a random value between 10 and 99 to all other points in the space. The problem is posed as a maximization problem. The basins of attraction for this function under steepest ascent are illustrate in Figure 10.1.

Note that there are five local optima under steepest ascent: 00000, 11111, 11010, 10001, and 10110.

In the current paper, we assume that 1-step of steepest ascent is applied to the initial population processed by the genetic algorithm, and that all offspring produced by recombination and/or mutation are also improved with 1-step of steepest ascent. As already noted, each improvement will change one bit in the string being processed. The first observation which can be made is that the genetic algorithm (i.e., not considering the steepest ascent component) will never process any of the saddle points in the search space. This is true in the sense that

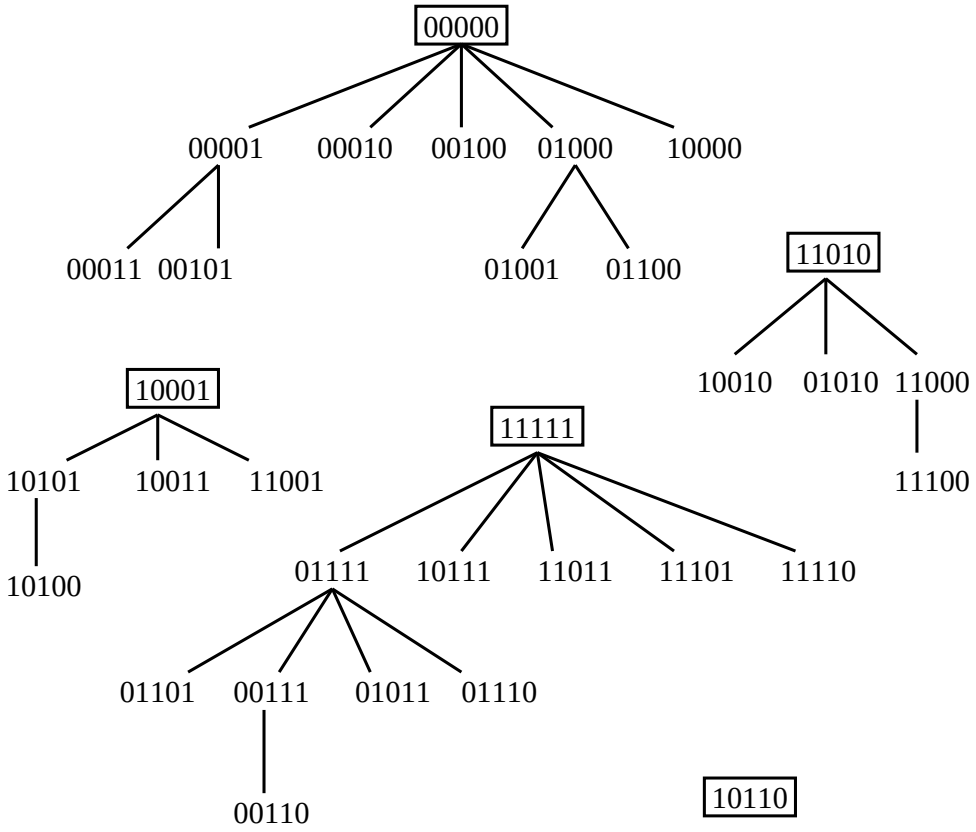


Figure 1: Basins of attraction under steepest ascent. The "boxed" strings are local optima.

local search could be built into the evaluation function: the evaluation function is passed a string, evaluates the string, does 1-step of steepest ascent, returns the improved string (assuming the current string is not an optima) and also returns the evaluation of the improved string. In this sense the genetic algorithm can be separated from the Lamarckian updates. As will be shown, this separation carries through to the infinite and finite population models of genetic search.

A related observation is that the actual values of the saddle points are irrelevant to the genetic search process except in so much as the evaluations of the saddle points conforms to the tree structures associated with the various basins of attraction. Since the saddle point are not actually processed during genetic search except inside the evaluation function, the actual evaluations of the saddle have no impact on the search process. For Function One 22 of the 32 points are saddle points. Since Function One is posed as a maximization problem, it follows that decreasing the evaluation associated with any of the 22 saddle points would not change the tree structures of the basins of attraction in any

way. The evaluations of the saddle points can also be increased as long as the tree structure remains unaltered. It thus follows that there are many functions which have exactly the same basins of attraction as Function One and which in expectation would be processed in an identical fashion by a hybrid genetic algorithm using 1 step of steepest ascent.

Figure 10.2 shows the strings in Function One that are actually evaluated by the simple genetic algorithm after 1 and 2 steps of steepest ascent. Also note that for two functions to be processed in an identical fashion by a genetic algorithm after applying steepest ascent to all members of the population every generation, all that is required is that the reduced tree structures be equivalent. It is not required that the strings which have been removed from the full function space map onto the remaining strings in the same way, or even that strings be in the same basin of attraction in the full function space. All that matters is that the reduces tree structures be identical with the same number of strings moving to the remaining nodes.

1.2 Models of Simple Genetic Algorithms

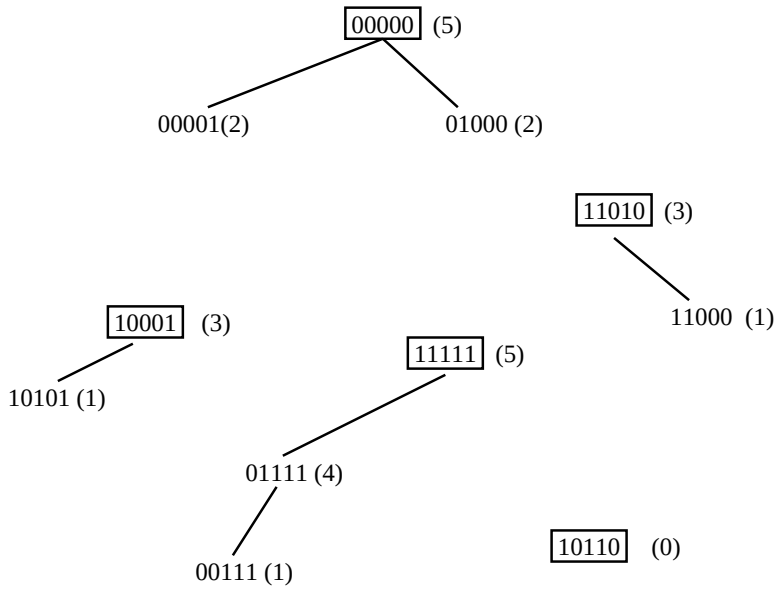
Goldberg (1987, 1989) and Bridges and Goldberg (1987) were the first to model critical details of how the genetic algorithm processes infinitely large populations under recombination. Vose (1990; Vose and Liepins 1991) extended and generalized this model. Whitley et al. (1992; Whitley 1993) introduce another version of the infinite population model that relates the work of Goldberg and Vose. Here, the Vose model is reviewed and it is shown how the effect of various operators, including local search, fit into this model.

The following notation is based on Vose and Liepins. The vector $p^t \epsilon \Re$ is such that the k^{th} component of the vector is equal to the proportional representation of string k at generation t. It is assumed that $n = 2^L$ is the number of points in the search space defined over strings of length L and that the vector p is indexed 0 to $n-1$. The vector $s^t \epsilon \Re$ represents the t^{th} generation of the genetic algorithm after selection and the i^{th} component of s^t is proportional representation of string i in the population after selection but before any operators (e.g. recombination, mutation, local search) are applied. Likewise, p_i^t represents the proportional representation of string i at generation t *before* selection occurs.

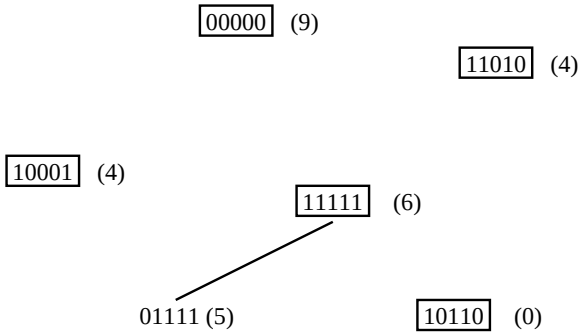
The function $r_{i,j}(k)$ yields the probability that string k results from the recombination of strings i and j. (For now assume that r only yields the results for recombination; the effect of other operators could also be included in r.) Now, using $\mathcal{E}$ to denote expectation,

$$\mathcal{E}\ p_k^{t+1} = \sum_{i,j} \ s_i^t\ s_j^t\ r_{i,j}(k) \tag{1}$$

To begin the construction of a general model, we first consider how to calculate the proportional representation of string 0 (i.e., the string composed of all zeros) at generation $t+1$; in other words, we compute p_0^{t+1}. A mixing matrix



Basins after 1 step of Steepest Ascent



Basins after 2 steps of Steepest Ascent

Figure 2: Basins of attraction for the same problem as they appear to a Hybrid Genetic Algorithm using 1 and 2 steps of steepest ascent. The numbers in parentheses beside each string indicates the number of strings that move to that position after steepest ascent.

M is constructed where the $(i,j)^{th}$ entry $m_{i,j} = r_{i,j}(0)$. Here M is built by assuming that each recombination generates a single offspring. The calculation of the change in representation for string $k = 0$ is now given by

$$\mathcal{E}\ p_0^{t+1} \quad = \quad \sum_{i,j} s_i^t s_j^t r_{i,j}(0) \quad = \quad s^T M s \tag{2}$$

where T denotes transpose. Note that this computation gives the expected representation of a single string, 0, in the next genetic population. Vose and Liepins formalize the notion that bitwise exclusive-or can be used to access various probabilities from the recombination function r. Specifically,

$$r_{i,j}(k) = r_{i,j}(k \oplus 0) = r_{i \oplus k, j \oplus k}(0). \tag{3}$$

This implies that the mixing matrix M, which was defined such that entry $m_{i,j} = r_{i,j}(0)$, can provide mixing information for any string k just by changing how M is accessed. By reorganizing the components of the vector s the mixing matrix M can yield information about the probability $r_{i,j}(k)$. A permutation function, σ, is defined as follows:

$$\sigma_j < s_0, ..., s_{n-1} >^T = < s_{j \oplus 0}, ..., s_{j \oplus (n-1)} >^T \tag{4}$$

where the vectors are treated as columns and n is the size of the search space. The computation

$$(\sigma_q\ s^t)^T M (\sigma_q\ s^t) = p_q^{t+1} \tag{5}$$

thus reorganizes s with respect to string q and produces the expected representation of string q at generation $t+1$. A general operator $\mathcal{M}$ can now be defined over s which remaps $s^T M s$ to cover all strings in the search space.

$$\mathcal{M}(s) = < (\sigma_0\ s)^T M (\sigma_0\ s), ..., (\sigma_{n-1}\ s)^T M (\sigma_{n-1}\ s) >^T \tag{6}$$

This model has not yet addressed how to generate the vector s^t given p^t. A fitness matrix F is defined such that fitness information is stored along the diagonal; the $(i,i)^{th}$ element is given by $f(i)$ where f is the fitness function. Following Vose and Wright (1994)

$$s^t = Fp^t / 1^T F p^t \tag{7}$$

since $Fp^t =< f_0 p_0^t, f_1 p_1^t, \ldots, f_{n-1}, p_{n-1}^t >$ and the population average is given by $1^T F p^t = f_0 p_0^t + f_1 p_1^t + \ldots + f_{n-1}, p_{n-1}^t$.

Vose (1993) refers to this complete model as the $\mathcal{G}$ function. Given any population distribution p, $\mathcal{G}(p)$ can be interpreted in two way. If the population is infinitely large, then $\mathcal{G}(p)$ is the next exact distribution of the next population. However given any (finite or infinite) population p, if strings in the next population are chosen one at a time, then $G(p)$ defines a vector such that element i

is chosen for the next generation with probability $\mathcal{G}(p)_i$. This is because $\mathcal{G}(p)$ defines the exact sampling distribution for the next generation. This is very useful when constructing finite Markov models of the simple genetic algorithm as well as the hybrid genetic algorithm as defined in this paper.

To motivate the inclusion of local search updates into the $\mathcal{G}$ function, we first look at how mutation can be added to this model in a modular fashion. Recall that M is the recombination matrix, which we initially defined to cover only crossover. Define $\mathcal{Q}$ as the mutation matrix. Assuming mutation is independently applied to each bit with the same probability, $\mathcal{Q}$ can be constructed by defining a mutation vector Γ such that component Γ_i is the probability of mutating string i and producing string 0. The vector Γ is the first column of the mutation matrix and in general Γ can be reordered to yield column j of the matrix by reordering such that element $q_{i,j} = \Gamma_{i \oplus j}$.

Having defined a mutation matrix, mutation can now be applied after recombination in the following fashion:

$$p^{t+1,m} = (p^{t+1})^T \mathcal{Q},$$

where $p^{t+1,m}$ is just the p vector at time t+1 after mutation. Mutation also can be done before crossover; the effect of mutation on the vector s immediately after selection produces the following change: $s^T \mathcal{Q}$, or equivalently, $\mathcal{Q}^T s$.

We now drop the $p^{t+1,m}$ notation and assume the original p^{t+1} vector is defined to include mutation, such that

$$p_0{}^{t+1} = (\mathcal{Q}^T s)^T M (\mathcal{Q}^T s)$$

$$p_0{}^{t+1} = s^T (\mathcal{Q} M \mathcal{Q}^T) s$$

$$p_0{}^{t+1} = s^T (\mathcal{Q} M \mathcal{Q}^T) s$$

$$\text{thus} \quad p_0{}^{t+1} = s^T M_2 s \quad \text{where} \quad M_2 = (\mathcal{Q} M \mathcal{Q}^T)$$

The matrix M as used here included only recombination, while M in the Vose model is a "mixing matrix" that includes both mutation and recombination. As long as the mutation rate is independently applied to each bit in the string, it makes no difference whether mutation is applied before or after recombination. Also, this view of mutation makes it clear how the general mixing matrix can be build by combining matrices for mutation and crossover.

1.3 Modeling Hybrid Simple Genetic Algorithms

The following function will be denoted Function Two:

```
e(0000) = 28     e(0100) = 20     e(1000) = 12     e(1100) =  4
e(0001) = 26     e(0101) = 18     e(1001) = 10     e(1101) =  2
e(0010) = 24     e(0110) = 16     e(1010) =  8     e(1110) =  0
e(0011) = 22     e(0111) = 14     e(1011) =  6     e(1111) = 30
```

where the function is posed as a maximization problem.

Recall that p_i^t, is the representation of string i in the population at time t. Let $p_i^{t,k}$ be the representation of string i in the population at time t after some k steps of local search. Function Two has two basins of attraction at 0000 and 1111. We now calculate how local search changes the distributions in the p vector. Note that the redistribution of string representations will be different for functions that are in different equivalence classes. For Function Two, the redistribution of points in the space occurs as follows. (The binary representation of the indices are used here to aid the reader in following the update process due to local search.)

$$
\begin{aligned}
p_{0000}^{t,k} &= p_{0000}^t + p_{0001}^t + p_{0010}^t + p_{0100}^t + p_{1000}^t \\
p_{0001}^{t,k} &= p_{0011}^t + p_{0101}^t + p_{1001}^t \\
p_{0010}^{t,k} &= p_{0110}^t + p_{1010}^t \\
p_{0100}^{t,k} &= p_{1100}^t \\
p_{1111}^{t,k} &= p_{1111}^t + p_{1110}^t + p_{1101}^t + p_{1011}^t + p_{0111}^{t,l} \\
p_{0011}^{t,k} &= 0 \\
p_{0101}^{t,k} &= 0 \\
p_{0110}^{t,k} &= 0 \\
p_{0111}^{t,k} &= 0 \\
p_{1000}^{t,k} &= 0 \\
p_{1001}^{t,k} &= 0 \\
p_{1010}^{t,k} &= 0 \\
p_{1011}^{t,k} &= 0 \\
p_{1100}^{t,k} &= 0 \\
p_{1101}^{t,k} &= 0 \\
p_{1110}^{t,k} &= 0
\end{aligned}
\tag{8}
$$

In this case, k = 1. Note that the local search updates to the p vector can also be expressed in the form of a matrix update. For the same single step of steepest ascent, those updates are as follows:

$$p^{t,k} = (p^t)^T \begin{bmatrix} 1\,0\,0\,0\,0 & & 0 \\ 1\,0\,0\,0\,0 & & 0 \\ 1\,0\,0\,0\,0 & & 0 \\ 0\,1\,0\,0\,0 & & 0 \\ 1\,0\,0\,0\,0 & & 0 \\ 0\,1\,0\,0\,0 & & 0 \\ 0\,0\,1\,0\,0 & & 0 \\ 0\,0\,0\,0\,0 & ... & 0 \\ 1\,0\,0\,0\,0 & & 0 \\ 0\,1\,0\,0\,0 & & 0 \\ 0\,0\,1\,0\,0 & & 0 \\ 0\,0\,0\,0\,0 & & 0 \\ 0\,0\,0\,0\,1 & & 0 \\ 0\,0\,0\,0\,0 & & 0 \\ 0\,0\,0\,0\,0 & & 0 \\ 0\,0\,0\,0\,0 & & 0 \end{bmatrix}$$

Note that the first column of the matrix has a 1 bit in those positions that directly "move" to string 0000 and in general column j of the matrix flags those strings that directly move to string j under 1 step of steepest ascent. Let L denote the update matrix for 1 step of local search; the matrix notation also has the advantage that for for k steps of local search $p^{t,k} = (p^t)^T L^k$. Thus, define $\mathcal{G}^k(p)$ to be the simple genetic algorithm function where k steps of local search are applied to each population (i.e., each generation).

2 THE MARKOV MODEL

Nix and Vose (1992) show how to structure the finite population for a simple genetic algorithm. Briefly, the Markov model is an N X N transition matrix Q, where N is the number of finite populations of K strings and $Q_{i,j}$ is the probability that the k^{th} generation will be population $\mathcal{P}_j$ given that the $(k-1)^{th}$ population is $\mathcal{P}_i$.

Let

$$< Z_{0,j}, Z_{1,j}, Z_{2,j}, ..., Z_{r-1,j} >$$

represent a population, where $Z_{x,j}$ represents the number of copies of string x in population j, and $r = 2^L$.

We will build the population incrementally. How may ways are there to place the $Z_{0,j}$ copies of string 0 in the population?

$$\binom{K}{Z_{0,j}}$$

How many ways can $Z_{1,j}$ now be placed in the population.

$$\binom{K - Z_{0,j}}{Z_{1,j}}$$

Continuing for all strings

$$\binom{K}{Z_{0,j}}\binom{K - Z_{0,j}}{Z_{1,j}}\binom{K - Z_{0,j} - Z_{1,j}}{Z_{2,j}}...\binom{K - Z_{0,j} - Z_{1,j} - ... - Z_{r-2,j}}{Z_{r-1,j}}$$

which yields

$$\frac{K!}{(K - Z_{0,j})!Z_{0,j}!} \quad \frac{(K - Z_{0,j})!}{(K - Z_{0,j} - Z_{1,j})!Z_{1,j}!} \quad \frac{(K - Z_{0,j} - Z_{1,j})!}{(K - Z_{0,j} - Z_{1,j} - Z_{2,j})!Z_{2,j}!}$$

$$...\frac{(K - Z_{0,j} - Z_{1,j} - ... - Z_{r-2,j})!}{Z_{r-1,j}!}$$

which in turn reduces to

$$\frac{K!}{Z_{0,j}!Z_{1,j}!Z_{2,j}!...Z_{r-1,j}!}$$

Let $C_i(y)$ be the probability of generating string y from the finite population P_i. Then

$$Q_{i,j} = \frac{K!}{Z_{0,j}!Z_{1,j}!Z_{2,j}!...Z_{r-1,j}!}\prod_{y=0}^{r-1} Ci(y)^{Z_{y,j}}$$

and

$$Q_{i,j} = K!\prod_{y=0}^{r-1}\frac{Ci(y)^{Z_{y,j}}}{Z_{y,j}!}$$

So, how do we compute $Ci(y)$? Note that the finite population P_i can be described by a vector p. Also note that the sampling distribution from which P_j is constructed is given by the infinite population model $\mathcal{G}(p)$. Thus, replacing $Ci(y)$ by $\mathcal{G}(p)_y$ yields (Vose, In Press):

$$Q_{i,j} = K!\prod_{y=0}^{r-1}\frac{(\mathcal{G}(p)_y)^{Z_{y,j}}}{Z_{y,j}!}$$

To build the Markov model for the hybrid genetic algorithm simply replace $Ci(y)$ by $\mathcal{G}^k(p)_y$.

3 CONCLUSIONS

It has been shown that a simple matrix representation can be used to characterize how local search updates can be integrated into the existing models of simple genetic algorithms. The local search updates can be modeled as simple updates to the p vector representing the proportional distribution of strings in an infinitely large population. In the current paper, the use of binary representations has been assumed, but models for the simple genetic algorithm have been generalized to include other representations by Vose (Vose, In Press; Whitley and Yoo, 1995) and generalizations to other forms of hybrid genetic algorithms should be straight forward in most cases.

The vector p not only tracks the representation of strings in a infinitely large population, but can also be used to represented the sampling distribution probabilities of a finite population genetic algorithm. Thus it follows that the use that the local search updates to the p vector carry over to the use of the p vector in the construction of finite Markov models for simple genetic algorithms.

References

[1] Bridges, C. and Goldberg, D. (1987) An analysis of reproduction and crossover in a binary-coded genetic Algorithm. *Proc. 2nd International Conf. on Genetic Algorithms and Their Applications.* J. Grefenstette, ed. Lawrence Erlbaum.

[2] Davis L. (1991) *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, NY.

[3] Goldberg, D. (1987) Simple Genetic Algorithms and the Minimal, Deceptive Problem. In, *Genetic Algorithms and Simulated Annealing,* L. Davis, ed., Pitman.

[4] Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison-Wesley.

[5] Mühlenbein, H. (1991) Evolution in Time and Space - The Parallel Genetic Algorithm. *Foundations of Genetic Algorithms*, G. Rawlins, ed. Morgan-Kaufmann. pp 316-337.

[6] Vose M. (1990) Formalizing Genetic Algorithms. In *Proc. of Genetic Algorithms, Neural Networks and Simulating Annealing Applied to Problems in Signal Processing and Image Processing*, IEEE. Glasgow, U.K.

[7] Vose, M. and Liepins, G., (1991) Punctuated Equilibria in Genetic Search. *Complex Systems* 5:31-44.

[8] Nix, A. and Vose, M. (1992) Modeling Genetic Algorithms with Markov Chains. *Annals of Mathematics and Artificial Intelligence.* 5:79-88.

[9] Vose, M., (1993) Modeling Simple Genetic Algorithms. *Foundations of Genetic Algorithms -2-,* D. Whitley, ed., Morgan Kaufmann.

[10] Vose, M. and Wright, G., (1993) Simple Genetic Algorithms with Linear Fitness. Unpublished Manuscript.

[11] Vose M. (In Press) The Simple Genetic Algorithms: Foundations and Theory. Cambridge, MA: MIT Press.

[12] Vose M. and Wright G. (1994) Simple Genetic Algorithms with Linear Fitness. *Evolutionary Computation*, 2(4): 347–368.

[13] Whitley, D., (1993) An Executable Model of a Simple Genetic Algorithm. *Foundations of Genetic Algorithms -2-.* D. Whitley, ed. Morgan Kaufmann.

[14] Whitley, D., Das, R., and Crabb, C. (1992) Tracking Primary Hyperplane Competitors During Genetic Search. *Annals of Mathematics and Artificial Intelligence.* 6:367-388.

[15] Whitley D., Gordon V.S. and Mathias, K.E. (1994) Lamarckian Evolution, The Baldwin Effect and Function Optimization, In *International Conference on Evolutionary Computation: PPSN 3* H-P Schwefel and R. Maenner, eds. Springer-Verlag.

[16] Whitley D. and Yoo N.W. (1995) Modeling Simple Genetic Algorithms for Permutation Problems. *Foundations of Genetic Algorithms -3-,* D. Whitley and M. Vose, eds., Morgan Kaufmann.