Binary and Gray Encoding in Univariate Marginal Distribution Algorithm, Genetic Algorithm, and Stochastic Hillclimbing

Uday K. Chakraborty
Dept. of Math & Computer Science
University of Missouri
St. Louis, MO 63121, USA
uday@cs.umsl.edu

Cezary Z. Janikow
Dept. of Math & Computer Science
University of Missouri
St. Louis, MO 63121

Abstract

This paper employs a Markov model to study the relative performance of binary and Gray coding in the univariate marginal distribution algorithm, genetic algorithm, and stochastic hillclimbing. The results indicate that while there is little difference between the two for all possible functions, Gray coding does not necessarily improve performance for functions which have fewer local optima in the Gray representation than in binary.

1 INTRODUCTION

Use of Gray coding has been shown to produce improved genetic algorithm performance in some cases [9, 2, 16]. This has led some researchers (e.g., [5]) to abandon binary coding in favor of Gray. Some others (e.g., [8]), however, did not find Gray helpful. Most of the previous research into the binary-versus-Gray issue in genetic algorithms has been based on (non-exhaustive) empirical studies. In this paper, we undertake a Markov chain theory-based exhaustive numerical approach to investigate the relative performance of the two representations in three cases: (1) univariate marginal distribution algorithm (UMDA) [12], (2) genetic algorithm (GA), and (3) stochastic hillclimbing (SH). As expected, due to the no-free-lunch theorem [20], our model indicates that for all possible functions there is little difference between the two. In [14, 19], it was argued that Gray encoding would outperform binary encoding on the special class of functions for which the number of local minima in the binary Hamming space is greater than the corresponding number in the Gray Hamming space. Our results show that even though it is often the case, it is not universally true.

2 LOCAL OPTIMA

In any binary representation, the neighbors of a given string are those with Hamming distance one. In the integer representation, the neighbors are those integers immediately greater and smaller. Thus, an L-bit string has exactly L neighbors in any binary representation and two such neighbors in the integer representation. A *local optimum* in a discrete search space is a point whose fitness is better than those of all of its neighbors.

It is possible for a function to have different numbers of local optima under different neighborhoods (i.e., different representations). The number of local optima of a fitness landscape is also referred to as its *modality*.

As an example, let us consider a discrete function of a single variable, $F(x)$, where the independent variable x can have a total of eight possible values. The eight x values can, without loss of generality, be mapped to the eight integers 0, 1, ..., 7. In the integer neighborhood, integer j has exactly two neighbors: $j-1$ and $j+1$. We consider a wrapping neighborhood, that is, $x = 7$ has neighbors $x = 6$ and $x = 0$, and $x = 0$ has neighbors $x = 1$ and $x = 7$. Function F is said to have a local optimum at $x = j$ if $F(j)$ is better than both $F(j-1)$ and $F(j+1)$. It is easy to see that in the integer neighborhood, the number of local optima of $F(x)$ in the above example can be 1, 2, 3 or 4.

In the Gray representation, the eight strings are 000, 001, 011, 010, 110, 111, 101, 100. Unlike the integer coding, each point in the Gray coding has exactly three neighbors. For example, the string 110 has 010, 111 and 100 as its neighbors. The binary encoding also induces three neighbors, but in general both the number and the relative locations of the local optima of a given function are different for integer, Gray and binary neighborhoods. For example, the function in Table 1 has 2, 1, and 3 local minima in the integer, Gray, and binary representations, respectively ($F(x)$

Table 1: A function may have different numbers of local minima in integer, Gray and binary neighborhoods.

x			$F(x)$
Integer	Gray	Binary	
0	000	000	20
1	001	001	0
2	011	010	10
3	010	011	40
4	110	100	60
5	111	101	50
6	101	110	70
7	100	111	30

Local minima		
Integer	Gray	Binary
0 at $x = 1$ 50 at $x = 5$	0 at $x = 001$	0 at $x = 001$ 10 at $x = 010$ 30 at $x = 111$

is an arbitrary function).

3 MARKOV MODEL OF GENETIC ALGORITHM

Markov chains have a long history of being used in the analysis of evolutionary algorithms (e.g., [7, 18, 13, 4, 17, 3, 15]). In the Markov model used here each population configuration represents a state. Let N and L represent, respectively, the population size and the string length. The number of occurrences of each of the 2^L strings in a given state is given by $state(i)$ for $i \in S$ where $S = \{0, 1, \ldots, 2^L - 1\}$. Let s represent the state space of the genetic algorithm. Then the size of the state space is given by ([13])

$$|s| = \left(\begin{array}{c} N + 2^L - 1 \\ N \end{array} \right).$$

Given a particular state (*state*), fitness-proportionate selection [6] selects a particular string (*str*) with probability

$$P_{sel}(str|state) = \frac{F(str) \cdot state(str)}{\sum_{j \in S} F(j) \cdot state(j)},$$

where F represents the fitness function. In the present model, two parent strings are selected (using proportional selection with replacement), crossed with probability of crossover p_c, and two children are produced by a head-tail swap (one-point crossover) of the parents. Finally, one of the two children is randomly chosen to go to the next generation. Thus the probability of creating a particular string, *str*, from a particular state, *state*, by the application of selection and crossover is given by $P_{sel-cross}(str|state) = p_c \times \sum_{str1, str2 \in S str1 \leq str2} \{P_{sel}(str1|state) \times P_{sel}(str2|state)\} \times \frac{1}{L-1} \sum_{cutpoint=1}^{L-1} Generate(str1, str2, str, cutpoint) + (1 - p_c) P_{sel}(str|state)$ where the function $Generate(str1, str2, str, cutpoint)$ returns 1 or 0 depending on whether or not the string *str* can be generated by crossing strings *str*1 and *str*2 at the cross-site denoted by *cutpoint*. Bit-wise mutation (with mutation probability p_m) changes a string, *str1* to another, *str2* with probability

$$P_{mut}(str2|str1) = p_m^{H(str1,str2)} \times (1 - p_m)^{L - H(str1,str2)}$$

where $H(i, j)$ is the Hamming distance between strings i and j. Therefore the probability that a particular string, *str*, is obtained from a particular state, *state*, by the application of selection, crossover and mutation is given by $P_{sel-cross-mut}(str|state) = \sum_{j \in S} P_{mut}(str|j) \cdot P_{sel-cross}(j|state)$. The transition from one state, *state1*, to another, *state2*, in a generational, non-elitist, simple genetic algorithm is governed by a multinomial distribution, and the transition probability is given by

$$P(state2|state1) = \frac{N!}{\prod_{str \in S} state2(str)!} \times \prod_{str \in S} (P_{sel-cross-mut}(str|state1))^{state2(str)}. \quad (1)$$

4 MARKOV MODEL OF UMDA

We consider the following univariate marginal distribution algorithm:

1. Set $t = 0$; create N individuals randomly.
2. Select $N_{select} \leq N$ individuals according to fitness-proportionate selection.
3. Compute the marginal frequencies $p_{select}(X_i, t)$, $i = 0, \ldots, L - 1$, from the selected individuals.
4. Sample N new individuals according to the distribution

   $$p(X_0, X_1, \ldots, X_{L-1}, t + 1) = \prod_{i=0}^{L-1} p_{select}(X_i, t);$$

   set $t = t + 1$.
5. Apply mutation (with a pre-determined probability) to the individuals created in the above step.
6. If the termination criteria are not satisfied, go to step 2.

We will compute two transition probability matrices: one corresponding to selection (step 2) , the other corresponding to sampling followed by mutation (steps 4 and 5). The final transition matrix of UMDA will be given by the product of these two matrices.

In UMDA, a single sampling operation creates a string $str \in \{0, 1, \cdots, 2^n - 1\}$ with probability

$$P_{sample}(str|state) = \prod_{i=0}^{L-1} prob(bit_i = \alpha_i),$$

where bit_i, the i-th bit (gene) in the string str, has the value (allele) $\alpha_i \in \{0, 1\}$. Note that $\sum_{str \in S} P_{sample}(str|state) = 1$. The probability at each locus can be computed from the configuration of the current population (state). The probability that mutation changes a string *str1* to another, *str2*, is given by

$$P_{mut}(str2|str1) = p_m^{H(str1,str2)} \times (1 - p_m)^{L - H(str1,str2)}$$

where $H(i, j)$ is the Hamming distance between strings i and j, and p_m is the probability of mutation. The state transition is described as sampling from a multinomial distribution:

$$P_{sample-mut}(state2|state1) = \frac{N!}{\prod_{str \in S} state2(str)!} \times \prod_{str \in S} (P_{sample-mut}(str|state1))^{state2(str)}, \quad (2)$$

where

$$P_{sample-mut}(str|state) = \sum_{j \in S} P_{mut}(str|j) \cdot P_{sample}(j|state).$$

Given a particular state (*state*), fitness-proportionate selection selects a particular string (*str*) with probability

$$P_{sel}(str|state) = \frac{F(str) \cdot state(str)}{\sum_{j \in S} F(j) \cdot state(j)},$$

where F represents the fitness function. The transition probability matrix corresponding to selection alone is given by

$$P_{sel}(state2|state1) = \frac{N!}{\prod_{str \in S} state2(str)!} \times \prod_{str \in S} (P_{sel}(str|state1))^{state2(str)}. \quad (3)$$

The final transition matrix of UMDA (with proportional selection and mutation) is then given by the product of the two matrices given by equations 2 and 3:

$$P_{UMDA} = P_{sel} \times P_{sample-mut}. \quad (4)$$

5 EXPECTED FIRST PASSAGE TIME TO CONVERGENCE

For a nonzero p_m, the Markov chain (in both GA and UMDA) is irreducible, that is, every state can be reached from every other state (all the entries in the transition probability matrix are strictly positive). In addition, the chain is regular (that is, ergodic with no cycles). By standard Markov chain theory it can be shown that the asymptotic transition probability distribution possesses a limit — the *stationary distribution* — and is independent of the starting state. Thus for the three-operator genetic algorithm, $\lim t \to \infty \mathcal{P}^{(t)}$ has all rows identical and no element in a row is zero.

We fill the $\begin{pmatrix} N + 2^L - 1 \\ N \end{pmatrix} \times \begin{pmatrix} N + 2^L - 1 \\ N \end{pmatrix}$ transition probability matrix with probabilities obtained by using equation 1 (for GA) or 4 (for UMDA). We compare the performances of binary and Gray encodings using the following metric: the expected first passage time to a state that contains at least one copy (instance) of the global optimum. Clearly, the lower this value, the better.

We denote by $p_{ij}^{(t)}$ the probability of transition from state i to state j in t steps. Let $f_{ij}^{(t)}$ stand for the probability that starting from state i the *first* entry to state j occurs at the t-th step:

$$P(T_{ij} = t) = f_{ij}^{(t)},$$

where T is a random variable representing the first passage time. We put $f_{ij}^{(0)} = 0$ for $i \neq j$, and $f_{jj}^{(0)} = 1$. Then $f_{ij}^{(1)} = p_{ij}^{(1)} = p_{ij}$ and

$$p_{ij}^{(t)} = \sum_{m=1}^{t} f_{ij}^{(m)} p_{jj}^{(t-m)}$$

where $p_{jj}^{(0)} = 1$, and $p_{ij}^{(0)} = 0$ for $i \neq j$. We can now get the f's recursively:

$$f_{ij}^{(t)} = p_{ij}^{(t)} - \sum_{m=1}^{t-1} f_{ij}^{(m)} p_{jj}^{(t-m)}$$

For a GA (or UMDA) with nonzero p_m, the $\{f_{ij}^{(t)}\}$ for any given pair of states (i, j) is a true probability distribution, that is, $\sum_{t=1}^{\infty} f_{ij}^{(t)} = 1$. The mean (expected) first passage time to state j, starting from state i, is then given by

$$E(T_{ij}) = \sum_{t=1}^{\infty} t \cdot f_{ij}^{(t)}.$$

The mean first passage time can be calculated by using the iterates of the transition probability matrix $\mathcal{P}$. However, the mean and the variance of the first passage time can also be obtained algebraically [11].

Let s_g represent the set of states containing at least one copy of the global optimum. Let h and k be two states such that $k \in s_g$, and $h \in s \backslash s_g$. To study what happens when, given an initial state h, the algorithm hits the state k for the first time, we can "stop" the process as soon as it reaches state k. We can accomplish this "stopping" by making k an absorbing state. In fact, we can go further and make each global-optimum-containing state an absorbing state. Finally, since we are interested in finding at least one copy of the global optimum, the absorbing states thus created can all be lumped into a single absorbing state, making our task easier. The modified transition probability matrix, $\mathcal{P}\prime$, then has exactly one absorbing state and the other states are transient. Let $\mathcal{Q}$ be the matrix obtained by truncating $\mathcal{P}\prime$ to include only the non-absorbing states. (As an example, for $L = 3$ and $N = 2$, $\mathcal{P}$ is a 36×36 matrix, and the dimensions of $\mathcal{Q}$ are 28×28.) Then $\mathcal{I} - \mathcal{Q}$ gives the "fundamental matrix" [11], and the mean time to absorption, starting from a given transient state, is given by the row-sum of the corresponding row of the matrix $(\mathcal{I} - \mathcal{Q})^{-1}$ (the number of rows in $(\mathcal{I} - \mathcal{Q})^{-1}$ is equal to the number of non-absorbing states in $\mathcal{P}\prime$).

Assuming a uniform random (0, 1) distribution for generating the bits in the initial generation ($t = 0$) of the GA (or the UMDA), each of the $|s|$ states is equally likely to represent the initial population, and this probability is $\frac{1}{|s|}$. The expected value of the expected first passage time to the global optimum is then given by

$$\mathcal{E} = \frac{1}{|s|} \sum_{i=1}^{|s|} E(T_i) \tag{5}$$

where E denotes expectation, and T_i is a random variable for the first passage time, given the start state i. For an absorbing state i, $P(T_i = 0)$ is unity.

The expected value $\mathcal{E}$ is computed for both binary and Gray encoding and is used as the basis of comparison in the remainder of this paper.

6 STOCHASTIC HILLCLIMBING

The following version of stochastic hillclimbing [1] is used in this paper (the problem considered is one of minimization):

1. Select a point — the *current* point, x_c — at random and evaluate it. Let the fitness be f_c.
2. Select an *adjacent* point, x_a, at random and evaluate it. Let f_a be its fitness.
3. Accept the adjacent point as the current point (that is, $x_c \leftarrow x_a$ with probability $\frac{1}{1+e^{\frac{f_a - f_c}{T}}}$ where T is a parameter (the "temperature") of the algorithm.
4. If a predetermined termination condition is not satisfied, go to step 2.

In stochastic hillclimbing the search begins with a single point and proceeds from one point (state) to another. For an L-bit problem the search space consists of 2^L points (states). At any single step, the process can move from a given point to itself or to any one of the L adjacent points (an adjacent point is a unit-Hamming-distance neighbor). A move from a current state f_i to a next (adjacent) state f_j takes place with probability

$$\frac{1}{L} \cdot \frac{1}{1 + e^{(f_j - f_i)/T}}.$$

The process stays in the same state f_i with probability

$$1 - \frac{1}{L} \sum_{k \in A_i} \frac{1}{1 + e^{(f_k - f_i)/T}}$$

where A_i is the set of states that are adjacent to f_i, $|A_i| = L$.

Therefore the entries of the $2^L \times 2^L$ transition probability matrix of the Markov chain for stochastic hillclimbing are given by

$$p_{ij} = \begin{cases} \frac{1}{L} \cdot \frac{1}{1+e^{(f_j - f_i)/T}} \text{ for } j \in A_i \\ 1 - \frac{1}{L} \sum_{k \in A_i} \frac{1}{1+e^{(f_k - f_i)/T}} \text{ for } i = j \\ 0 \text{ otherwise} \end{cases} \tag{6}$$

In this case, there is exactly one optimal state (and that state corresponds to the globally best string). When we make that state into an absorbing state, the truncated matrix of size $(2^L - 1) \times (2^L - 1)$ represents the $\mathcal{Q}$ matrix referred to earlier. We can now obtain the mean first passage times to optimality from row-sums of the matrix $(\mathcal{I} - \mathcal{Q})^{-1}$.

7 RESULTS

There are infinitely many functions defined over L bits, differing by function evaluations and their permutations. To have a finite case, we restrict function evaluations to the range 1 to 2^L and we permute these

2^L distinct values. Thus, for $L = 3$, we have a total of $(2^3)! = 40{,}320$ different functions, corresponding to as many permutations. For example, $L = 3$ gives $2^3 = 8$ function evaluations: 1,2, ..., 8, and for these 8 evaluations, one possible permutation is $\{F(0) = 1, F(1) = 2, \ldots, F(7) = 8\}$.

Without loss of generality, we consider a minimization problem. For each of these 40,320 functions, we count the number of optima in each of the three representations. In Table 2 we show these counts in four categories, corresponding to 1, 2, 3 and 4 local minima in the integer representation. For instance, out of a total of 40,320 functions, 2176 have four minima each in the integer representation. Among these 2176 functions, 32 have two minima, 704 have three, and 1440 have four each in the Gray representation. Again, the same 2176 functions can be grouped into two classes: 1408 functions having one minimum each and 768 with two minima each in the binary representation. Therefore, as expected, a given function can have different numbers of local minima under different representations. However, the total number of functions with a given number of minima is the same for Gray and binary representations (see Table 3). By covering all $(2^L)!$ functions, we have included all possible situations. For example, over 3 bits, there will always be exactly 1232 functions with 2 local minima in the integer neighborhood, 2 in the Gray neighborhood and 3 in binary, regardless of the particular fitness values assigned to the individual strings.

Table 2: The number of local minima in all possible functions defined over three bits. The functions are divided into 4 categories corresponding to 1,2,3 or 4 local minima in the integer neighborhood. See also Table 3.

Integer		Gray		Binary	
#min	#fun	#min	#fun	#min	#fun
1	512	1	512	1	64
				2	384
				3	64
2	14592	1	6144	1	3056
		2	8448	2	10032
				3	1360
				4	144
3	23040	1	1984	1	4112
		2	16000	2	13296
		3	5056	3	4336
				4	1296
4	2176	2	32	1	1408
		3	704	2	768
		4	1440		
Total	40320		40320		40320

Performance comparisons for the GA ($L = 3$, $N = 2$, $p_c = 0.8$, $p_m = 0.05$), the UMDA ($L = 3$, $N = 2$, $p_m = 0.05$) and SH ($L = 3$) are shown in Tables 4 - 7 where the expected first passage times (equation 5) have been used as the basis of comparison. An encoding is better if it has a smaller expected first passage time to find the global optimum. Note that no GA / UMDA / SH runs (experiments) were performed; we obtained the first passage times theoretically, via the Markov chain calculations of Sections 3 - 6. For presentation, the functions are divided into 26 equivalence groups based on the number of local minima in the three neighborhoods.

Table 3: The total number of functions with 1,2,3 or 4 local minima under the three neighborhoods. $L = 3$.

#minima	#functions in different neighborhoods		
	Integer	Gray	Binary
1	512	8640	8640
2	14592	24480	24480
3	23040	5760	5760
4	2176	1440	1440
Total	40320	40320	40320

Table 4: Performance comparison of Binary and Gray coding in the GA ($L = 3$, N=2, $p_c = 0.8$, $p_m = 0.05$). Gray wins a total of 19296 times, binary wins 21024 times. I = Integer, G = Gray, B = Binary

#Func	#Minima			No. of Times Coding Better	
	I	G	B	Gray	Binary
64	1	1	1	40	24
384	1	1	2	380	4
64	1	1	3	64	0
768	2	1	1	324	444
5248	2	1	2	5032	216
128	2	1	3	128	0
2288	2	2	1	52	2236
4784	2	2	2	1764	3020
1232	2	2	3	532	700
144	2	2	4	84	60
224	3	1	1	116	108
1568	3	1	2	1500	68
192	3	1	3	192	0
2016	3	2	1	276	1740
9024	3	2	2	5132	3892
3664	3	2	3	2332	1332
1296	3	2	4	924	372
1872	3	3	1	0	1872
2704	3	3	2	332	2372
480	3	3	3	24	456
16	4	2	1	0	16
16	4	2	2	8	8
432	4	3	1	0	432
272	4	3	2	60	212
960	4	4	1	0	960
480	4	4	2	0	480

As we observe from the tables, both representations produce approximately the same number of winners. This reiterates the known fact [20] that no represen-

tation should be superior for all classes of problems. The small discrepancy may probably be attributed to the choice of parameters. To evaluate the effect of operator probabilities, we re-calculated the first passage times for different probabilities. Table 5 shows some representative cases. We can see that while the relative performance is affected by the parameter values, the differences are not very significant even for the extreme crossover/mutation rates. The results are dependent on the (raw) fitnesses because of our use of fitness-proportionate selection. Use of a rank-based selection would eliminate that dependence.

Table 5: Performance comparison of Binary and Gray coding in the GA (different crossover and mutation probabilities have been used). $L = 3$, $N = 2$.

#Func	#Min			Parameters		Times Winner	
	I	G	B	p_c	p_m	Gray	Binary
64	1	1	3	0.8	0.05	64	0
				0.0	0.1	60	4
1232	2	2	3	0.8	0.05	532	700
				0.0	0.2	500	732
				1.0	0.001	640	592
144	2	2	4	0.8	0.05	84	60
				0.0	0.2	68	76
				1.0	0.001	84	60
272	4	3	2	0.8	0.05	60	212
				0.0	0.1	56	216
				1.0	0.0001	64	208

Overall, the results show that contrary to popular belief, it is not necessarily true that fewer local optima make the task easier for the genetic algorithm. (The 9th row in Table 4, showing 1232 functions with 2, 2 and 3 local minima in integer, Gray and binary representations, respectively, is particularly interesting: binary is the winner in more that half of the 1232 functions.) This corroborates Horn and Goldberg [10], who have shown that some maximally multimodal functions can be easier than unimodal functions for the genetic algorithm. In [14, 19] it was argued that Gray would be better than binary for functions with fewer local optima in the Gray Hamming space than in the binary Hamming space. From the above results we see that this is not always true.

8 CONCLUSIONS

This paper has shed some light on the Gray-versus-binary debate in evolutionary computation. Finite-population models of univariate marginal distribution algorithm and genetic algorithm, and a model of stochastic hillclimbing have been developed using well-known techniques from Markov chain theory, and the relative performance of Gray and binary encoding studied using the expected first passage time to optimality as the figure of merit. Over all possible functions there is not much difference between the two representations, but fewer local optima do not necessarily make the task easier for Gray coding. The present model is complete, that is, all the three operators – selection, crossover and mutation – have been taken into account, and it is exact, that is, it does not need any approximation or assumption. The results were validated for different probabilities of mutation and crossover.

Table 6: Performance comparison of Binary and Gray coding in the UMDA ($L = 3$, N=2, $p_m = 0.05$).

#Func	#Minima			No. of Times Coding Better	
	I	G	B	Gray	Binary
64	1	1	1	40	24
384	1	1	2	380	4
64	1	1	3	64	0
768	2	1	1	324	444
5248	2	1	2	5032	216
128	2	1	3	128	0
2288	2	2	1	52	2236
4784	2	2	2	1764	3020
1232	2	2	3	532	700
144	2	2	4	84	60
224	3	1	1	116	108
1568	3	1	2	1500	68
192	3	1	3	192	0
2016	3	2	1	276	1740
9024	3	2	2	5132	3892
3664	3	2	3	2332	1332
1296	3	2	4	924	372
1872	3	3	1	0	1872
2704	3	3	2	332	2372
480	3	3	3	24	456
16	4	2	1	0	16
16	4	2	2	8	8
432	4	3	1	0	432
272	4	3	2	60	212
960	4	4	1	0	960
480	4	4	2	0	480

A limitation of the present approach is that it allows us to study all possible functions defined on up to 3 bits. An exhaustive enumeration of all possible functions on a large number of bits and calculating the first passage times is computationally prohibitive (for 4 bits, there are $16! \approx 2 \times 10^{13}$ possible functions).

Acknowledgments

We are grateful to two anonymous referees for their comments.

Table 7: Performance comparison of Binary and Gray coding in stochastic hillclimbing. $L = 3$.

#Func	#Minima			T	Times Coding Better	
	I	G	B		Gray	Binary
384	1	1	2	10	356	28
				5	372	12
5248	2	1	2	10	4560	688
				20	4488	760
2288	2	2	1	10	308	1980
				15	336	1952
				12	328	1960
1232	2	2	3	0.9	640	592
				0.1	696	536
				0.5	656	576
				0.8	656	576
				1.0	652	580
				5.0	668	564
1568	3	1	2	10	1308	260
				5	1368	200
				15	1284	284
				20	1268	300
2016	3	2	1	10	548	1468
				12	552	1464
				15	560	1456
3664	3	2	3	10	2392	1272
				15	2368	1296
2704	3	3	2	0.2	720	1984
				0.5	680	2024
272	4	3	2	0.5	112	160
				1.0	76	196

References

[1] Ackley, D. H. (1997): *A Connectionist Machine for Genetic Hillclimbing*, Boston, MA: Kluwer Academic Publishers.

[2] Caruana, R.A. and Schaffer, J.D. (1988): Representation and hidden bias: Gray vs. binary coding for genetic algorithms, Proc. 5th Internat. Conf. on Machine Learning, pp. 153-161, Los Altos: CA, Morgan Kaufmann.

[3] Chakraborty, U.K., Deb, K. and Chakraborty, M. (1996): Analysis of selection algorithms: A Markov chain approach. *Evolutionary Computation* 4(2), 133-167.

[4] Davis, T.E. and Principe, J.C. (1993). A Markov chain framework for the simple genetic algorithm. *Evolutionary Computation*, **1**, 269-288.

[5] Eshelman, L.J. (1991): The CHC adaptive search algorithm, Foundations of Genetic Algorithms - I, Morgan Kaufmann.

[6] Goldberg, D.E. (1989): *Genetic Algorithms in Search, Optimization, and Machine Learning*, Boston: Addison-Wesley.

[7] Goldberg, D.E. and Segrest, P. (1987). Finite Markov chain analysis of genetic algorithms. In J.J. Grefenstette (Ed.), *Proc. 2nd International Conf. on Genetic Algorithms* (pp. 1-8). Hillsdale, NJ: Lawrence Erlbaum.

[8] Haupt, R.L. and Haupt, S.E. (1998) *Practical Genetic Algorithms*, New York: Wiley.

[9] Hollstein, R.B. (1971): *Artificial genetic adaptation in computer control systems*, PhD Thesis, Univ. of Michigan.

[10] Horn, J., Goldberg,D.E., Genetic algorithm difficulty and the modality of fitness landscapes, FOGA-3, 1994, 243-269.

[11] Kemeny, J.G. and Snell, J.L. (1960) *Finite Markov Chains*, Van Nostrand, Princeton.

[12] Muehlenbein, H. & Paass, G. (1996) From recombination of genes to estimation of distributions, Proc. PPSN-IV, Springer.

[13] Nix. A.E. and Vose, M.D. (1992). Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence* **5**, 79-88.

[14] Rana, S.B., Whitley, L.D. (1997): Bit representations with a twist. Proc. 7th ICGA, pp. 188-195.

[15] Rudolph, G., Finite Markov Chain Results in Evolutionary Computation: A Tour d'Horizon, Fundamenta Informaticae 35(1-4):67-89, 1998.

[16] Schaffer, J.D. et al. *A study of control parameters affecting online performance of genetic algorithms for function optimization*, Proc. 3rd ICGA, 1989 (Morgan Kaufmann).

[17] Suzuki, J. (1993). A Markov chain analysis on a genetic algorithm. In S. Forrest (Ed.), *Proc. Fifth Int'l Conf. on Genetic Algorithms* (pp. 146-153). San Mateo, CA: Morgan Kaufmann.

[18] Vose, M.D. (1993). Modeling simple genetic algorithms. In L.D. Whitley (Ed.), *Foundations of Genetic Algorithms - 2* (pp. 63-74). San Mateo, CA: Morgan Kaufmann.

[19] Whitley, D (1999): A free lunch proof for Gray versus binary encodings, Proc. Genetic and Evolutionary Computation (GECCO-1999), pp. 726-733.

[20] Wolpert, D.H. and MacReady, W.G. (1997): No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1(1), pp. 67-82.