An Analysis of Gray versus Binary Encoding in Genetic Search

Uday K. Chakraborty[1] **and Cezary Z. Janikow**
Dept. of Mathematics and Computer Science
University of Missouri
St. Louis MO 63121 USA

Abstract

This paper employs a Markov model to study the relative performance of binary and Gray coding in genetic algorithms. The results indicate that while there is little di erence 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

Any parameter optimization technique, including genetic algorithms (GAs) [9], requires some method of representing the parameters. Without loss of generality, we discuss here integer parameters. One approach used in genetic algorithms is to code an integer parameter directly in its base-2 representation, using explicit bits (the genotype level), and then apply a standard binary-to-integer mapping to decode the parameter value (the phenotype level). Alternatively, one may represent the parameter using the readily available integer data type (thus merging the genotype and phenotype levels). Among possible bit-string representations, the Gray code is known to alleviate the "Hamming cli " problem. An example of a Hamming cli is the transition from 7 to 8 in binary coding, where all the bits (in four-bit coding) change (from 0111 to 1000) corresponding to a change of one in the phenotype. The distance between two chromosomes, at the genotype level, is measured by Hamming distance, which is simply the number of bits that di er. In the Gray code, the Hamming distance is always one for any two strings (chromosomes) that are adjacent (di ering by one) at the phenotype level. That is not the case in the standard binary code where a single bit flip at the most significant position dramatically changes the value (phenotype). There are many Gray codes [8]; in this paper we use the *binary reflected Gray code* and refer to it as simply the Gray code. The algorithms for binary-to-Gray and Gray-to-binary conversions are given below (a binary string $b_1, \ldots, b_L$ and a Gray string $g_1, \ldots, g_L$ are considered):

procedure Binary-to-Gray
begin
  $g_1 = b_1$;
  for $i = 2$ to n do
    $g_i = b_{i\ \ 1}$ XOR b_k;
end

procedure Gray-to-Binary
begin
  b_1 = bitvalue = g_1;
  for $i = 2$ to n do
  begin
    if $g_i = 1$ then bitvalue = COMPLEMENT(bitvalue);
    b_i = bitvalue;
  end
end

Corresponding author Email: uday@cs umsl edu

Use of Gray coding has been shown to produce improved genetic algorithm performance in some cases [13, 2, 19, 4]. This has led some researchers (e.g., [7]) to abandon binary coding in favor of Gray. Some others (e.g., [12]), however, did not find Gray helpful. Most of the previous research into the binary-versus-Gray issue in genetic algorithms have 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. As expected, due to the no-free-lunch theorem [23], our model indicates that for all possible functions there is little di erence between the two. In [17, 22], 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.

We also analyze the comparative performance of Gray and binary encoding for a simpler search algorithm of the same genre, namely stochastic hillclimbing. Our Markov model for stochastic hillclimbing shows essentially the same Gray-versus-binary behavior.

The remainder of this paper is organized as follows. In Section 2 we explain integer, binary and Gray neighborhoods and illustrate with a specific example how a function may possess di erent local optima in di erent neighborhoods. Section 3 describes the Markov model, and Section 4 explains the metric used for comparison — the expected first passage time to optimality. Results of genetic algorithm performance with the two encodings are presented in Section 5. Section 6 argues that alongside the choice of representations, the right choice of genetic operators is also important for any practical application. In Section 7 a Markov model is developed for stochastic hillclimbing and relative performance results are presented for Gray and binary. Section 8 provides a summary and some concluding remarks.

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 di erent numbers of local optima under di erent neighborhoods (i.e., di erent representations). The number of local optima of a fitness landscape is also refereed 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: $j \quad 1$ 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 $F(j \quad 1)$ 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 di erent 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 ($F(x)$ is an arbitrary function).

3 The Markov model

Markov chains have a long history of being used in the analysis of evolutionary algorithms (e.g., [6, 10, 11, 21, 16, 5, 20, 3, 18]). 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 $S = \{0, 1, \ldots, 2^L \quad 1\}$. Let s

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 $x = 111$

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

represent the state space of the genetic algorithm. Then the size of the state space is given by ([16])

$$|s| = \begin{matrix} N + 2^L & 1 \\ N & \end{matrix} \Bigg).$$

Given a particular state (*state*), fitness-proportionate selection [9] 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 of the parents. Finally, one of the two children is randomly chosen to go to the next geneartion. Thus the probability of creating a particular string, *str*, from a particular state, *state*, by the application of selection and crossover is given by

$$\begin{aligned} P_{sel \;\; cross}(str|state) \;\; = \;\; & p_c \times \sum_{\substack{str \;\; str2 \in S \\ str \;\;\;\; str2}} P_{sel}(str1|state) P_{sel}(str2|state) \times \\ & \frac{1}{L \;\; 1} \sum_{cutpoint=1}^{L \;\; 1} Generate(str1, str2, str, cutpoint) + \\ & (1 \;\;\; p_c) P_{sel}(str|state) \end{aligned}$$

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

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

For a nonzero p_m, the Markov chain 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.

4 Expected rst passage time to convergence

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. 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 in a genetic algorithm starting from state i the *first* entry to state j occurs at the t-th step:

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

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:

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

For a genetic algorithm 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, in this paper we use a di erent approach [15].

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 genetic 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 $\mathcal{I} \quad \mathcal{Q}$ gives the "fundamental matrix" (see Appendix), 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 $(\mathcal{I} \quad \mathcal{Q})^{-1}$ (the number of rows in $(\mathcal{I} \quad \mathcal{Q})^{-1}$ 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 genetic algorithm, 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

$$\mathcal{E} = \frac{1}{|s|} \sum_{i=1}^{|s|} E(T_i) \tag{2}$$

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.

5 Results

There are infinitely many functions defined over L bits, di ering 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 $(2^3)! = 40{,}320$ di erent 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 di erent numbers of local minima under di erent 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.

Performance comparisons for $L = 3$, $N = 2$, $p_c = 0.8$, $p_m = 0.05$ are shown in Table 4 where the expected first passage times (equation 2) 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 runs (experiments) were performed; we obtained the first passage times theoretically, via the Markov chain calculations of Sections 3 and 4. For presentation, the functions are divided into 26 equivalence groups based on the number of local minima in the three neighborhoods.

As we observe from Table 4, both representations produce approximately the same number of winners: 19296 for Gray and 21024 for binary. This reiterates the known fact [23] that no representation should be superior for all classes of problems. The small discrepancy may be attributed to the choice of parameters. The p_c, p_m values used in Table 4 are "standard". To evaluate the e ect of operator probabilities, we re-calculated the first passage times for vastly di erent probabilities. Table 5 shows some representative cases. We can see that while the relative performance is a ected by the parameter values, the di erences are not very significant even for the extreme crossover/mutation rates. The results are quite stable for more reasonable variations in the standard mutation/crossover rates (as we observed separately). The results in Table 4 are dependent on the (raw) fitnesses because of our use of fitness-proportionate selction in Section 3. Use of a rank-based selection would eliminate that dependence.

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:

Integer		Gray		Binary	
#minima	#functions	#minima	#functions	#minima	#functions
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

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.

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

No. of Functions	No. of Minima in Di erent Neighborhoods			No. of Times Coding Better	
	Integer	Gray	Binary	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

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

No. of Functions	No. of Minima			Parameters		No. of Times Winner	
	Integer	Gray	Binary	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

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

binary is the winner in more that half of the 1232 functions.) This collaborates Horn and Goldberg [14], who have shown that some maximally multimodal functions can be easier than unimodal functions for the genetic algorithm. In [17, 22] 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. In Section 7, similar results are obtained for Gray and binary encoding in the case of stochastic hillclimbing.

6 Another look at the choice of representation

We considered two di erent bit-string representations, while assuming exactly the same mutation and crossover operators. However, in practice one may apply di erent operators, possibly representation- or domain-specific. Thus, in general, the usefulness of a given representation cannot be assessed without taking the operators into account. To prove this point, we show that equivalent operators can always be constructed such that the Gray-coded GA runs *exactly* the same as the binary-coded GA. (Note that this will not be the case for two arbitrary representations if their representative powers are di erent, for example, if one representation cannot map some solution points).

Consider GA_b (based on the standard binary encoding) along with two known operators: mutation mut_b and crossover $cross_b$. Given some operator probabilities, one may now execute GA_b, initialized with some population. Now consider GA_G (Gray-based), with two unknown operators mut_G and $cross_G$, but the same probabilities of application as those of mut_b and $cross_b$, respectively. The algorithms do not di er except for the representation and the operators. For the sake of presentation, we assume the generational model and an integer parameter optimization problem.

Suppose that the initialization routine generates random integer values in the domain, and then encodes them in a given representation. Because of this, GA_b will be initialized with exactly the same population as GA_G, at the phenotype level, and thus $phenotype(POP_b^0) = phenotype(POP_G^0)$. However, the genotypes of POP_b^0 and POP_G^0 are not the same. Now, any rank-based or fitness-based selection will select exactly the same phenotypes in the two algorithms.

Define $chrom : m_b^i$ and $chrom : m_G^i$ to be the chromosomes (genotypes) corresponding to a particular fitness (phenotype) m in GA_b and GA_G, respectively, at generation i. (Note that $phenotype(chrom : m_G^i) = phenotype(chrom : m_b^i) = m$.) Define, for either representation, $mut(chrom : m)$ to be the mutation o - spring of the chromosome $chrom : m$, and $cross(chrom : m, chrom : n)$ to be the pair of crossover o spring. If we could define mut_G and $cross_G$ so that

$$\begin{aligned} phenotype(mut_b(chrom : m_b)) &= phenotype(mut_G(chrom : m_G)) \text{ and} \\ phenotype(cross_b(chrom : m_b, chrom : n_b)) &= phenotype(cross_G(chrom : m_G, chrom : n_G)), \end{aligned}$$

then obviously all subsequent populations would also be phenotypically the same, yielding exactly the same runs. Therefore, if such mutation and crossover can be defined, the representation will become transparent to the application.

Can we define such operators in GA_G? Yes indeed – we illustrate that with actual construction. Take the chromosome $chrom : m_G^0$ selected from the initial population for mutation in GA_G. Remember that according to the initialization routine, $phenotype(chrom : m_G^0) = phenotype(chrom : m_b^0)$. Now define mut_G as follows:

1. Transform $chrom : m_G^0$ to its equivalent standard binary representation $(chrom : m_G^0)_b$, according to the procedure Gray-to-Binary (Section 1). Because the phenotypes were the same, $(chrom : m_G^0)_b = chrom : m_b^0$.

2. Generate mutation o spring of the above chromosome using the known binary mutation mut_b. Of course, $mut_b((chrom : m_G^0)_b)$ must be the same as $mut_b(chrom : m_b^0)$. Let mut_b change the phenotype of the string $chrom : m_b^0$ from m to q and the genotype from $chrom : m_b^0$ to $chrom : q_b^1$. Therefore, $mut_b((chrom : m_G^0)_b)$ must equal $chrom : q_b^1$.

3. Apply the reverse transformation Binary-to-Gray to express $mut_b((chrom : m_G^0)_b) = chrom : q_b^1$ in the Gray encoding as $chrom : q_G^1$.

Clearly, $phenotype(chrom : q_b^1) = phenotype(chrom : q_G^1)$ (and this value is q). In other words, mut_G (as defined above) has produced a string with the *same* phenotype as that of the string produced by mut_b.

The same can be argued for crossover — we can transform both chromosomes from GA_G into the binary representation, apply $cross_b$ to generate two o spring, and then transform these o spring back to the Gray encoding. Again, the resulting o spring would be the same in both algorithms at the phenotype level, resulting in the same evaluations. Thus we have $phenotype(POP_b^1) = phenotype(POP_G^1)$.

Because the selection is either fitness-based or rank-based, any pair of corresponding (i.e., same-phenotype) chromosomes in both representations will either undergo the same operations in the next generation or both die. Therefore, $phenotype(POP_b^2) = phenotype(POP_G^2)$, and finally $phenotype(POP_b^{Final}) = phenotype(POP_G^{Final})$. In other words, both GA_b and GA_G produce the same final populations as far as the fitness values are concerned, thus producing indistinguishable results.

Of course, one may argue that no one would use such carefully crafted operators. This is true for the two simple representations considered here, but is not true for many domain-rich representations in which practitioners often use complex mutation and crossover operators. We only intended to show that for any practical application one must take the operators into account, otherwise representation choices based on presumed superiority are of little importance.

7 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 ane 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

$$\frac{1}{1 + e^{\frac{f_a \quad f_c}{T}}}$$

   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

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

The process stays in the same state f_i with probability

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

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

$$p_{ij} = \begin{cases} \frac{1}{L} \cdot \frac{1}{1+e^{\ f_j \quad f_i) \ T}} \text{ for } j \in A_i \\ 1 \quad \frac{1}{L} \sum_{k \in A_i} \frac{1}{1+e^{\ f_k \quad f_i) \ T}} \text{ for } i = j \\ 0 \text{ otherwise} \end{cases} \tag{3}$$

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 $(2^L \quad 1) \times (2^L \quad 1)$ represents

No. of Functions	No. of Minima in Di erent Neighborhoods			T	No. of Times Coding Better	
	Integer	Gray	Binary		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

Table 6: Performance comparison of Binary and Gray coding in stochastic hillclimbing. $L = 3$.

the $\mathcal{Q}$ matrix referred to earlier. We can now obtain the mean first passage times to optimality from row-sums of the matrix $(\mathcal{I} \quad \mathcal{Q})^{-1}$. The fitnesses used are as in the previous sections. Some representative results for $L = 3$ are shown in Table 6. As in the case of the genetic algorithm, fewer minima do not necessarily make the search easy for hillclimbing. Also, the relative performance is seen to be a ected by the algorithm parameter (T).

8 Conclusions

This paper has shed some light on the Gray-versus-binary debate in genetic algorithms. A finite-population genetic algorithm has been modeled using well-known techniques from Markov chain theory and the relative performance of Gray-coded and binary-coded genetic algorithms studied using the expected first passage time to optimality as the figure of merit. Over all possible functions there is not much di erence 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 di erent probabilities of mutation and crossover.

A Markov model for stocahstic hillclimbing was also developed and performance comparison of Gray and binary coding investigated, using the same criterion of how quickly the global optimum is found out. The results are similar to those obtained for the genetic algorithm.

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).

Appendix: Some de nitions and results from Markov chain theory

We collect here some definitions and results. Proofs can be found in Kemeny and Snell (1960).

DEFINITION A.1: An *ergodic set* of states is a set in which every state can be reached from every other state, and which, once entered, cannot be left.

DEFINITION A.2: An *ergodic state* is an element of an ergodic set.

DEFINITION A.3: A *transient set* of states is a set in which every state can be reached from every other state, and which can be left.

DEFINITION A.4: A *transient state* is an element of a transient set.

DEFINITION A.5: An *absorbing* state is a state which once entered is never left.

DEFINITION A.6: The *period* of any state i is defined as the greatest common divisor of all integers k (≥ 1) for which $p_{ii}^{k)} > 0$. When this greatest common divisor is 1, the state i is *aperiodic*.

DEFINITION A.7: An *ergodic chain* is one whose states form a single ergodic set.

DEFINITION A.8: A *cyclic chain* is an ergodic chain in which each state can only be entered at certain periodic intervals.

DEFINITION A.9: A *regular chain* is an ergodic chain that is not cyclic.

Let the Markov chain have s transient states, and $r \quad s$ ergodic states. Then the canonical form of the transition probability matrix becomes

$$\mathcal{P} = \quad \begin{matrix} \mathcal{S} & \mathcal{O} \\ \mathcal{R} & \mathcal{Q} \end{matrix} \Bigg)$$

The region $\mathcal{O}$ consists of zeros. The $(r \quad s)$x$(r \quad s)$ matrix $\mathcal{S}$ concerns the process after it has reached an ergodic set. The sxs submatrix $\mathcal{Q}$ corresponds to the transition probabilities only among the transient states. $\mathcal{R}$ is an sx$(r \quad s)$ matrix whose elements are the probabilities of transition from the transient states to the ergodic states. It can be shown that as $k \to \infty$, $\mathcal{Q}^k$ tend to $\mathcal{O}$ (zero matrix). For an absorbing chain, $\mathcal{S}$ is an $(r \quad s)$x$(r \quad s)$ identity matrix, $\mathcal{I}_{r \quad s)\mathrm{x}\, r \quad s)}$.

DEFINITION A.10: For an absorbing Markov chain, the *fundamental matrix* is defined to be $\mathcal{M} \equiv (\mathcal{I} \quad \mathcal{Q})^{\ 1}$.

The fundamental matrix plays a useful role in absorbing Markov chain theory. The existence of the inverse of the matrix $(\mathcal{I} \quad \mathcal{Q})$ is established by Theorem A.1 which we state without proof.

THEOREM A.1: For an absorbing Markov chain partitioned as shown above, the inverse $(\mathcal{I} \quad \mathcal{Q})^{\ 1}$ exists, and

$$(\mathcal{I} \quad \mathcal{Q})^{\ 1} = 1 + \mathcal{Q} + \mathcal{Q}^2 + \cdots = \sum_{k=0}^{\infty} \mathcal{Q}^k.$$

THEOREM A.2: The mean times to absorption (corresponding to di erent start states) are given by the entries of $\mathcal{M}$, and the variances by $(2\mathcal{M} \quad I)\mathcal{M} \quad (\mathcal{M}\ )_{sq}$, where $\mathcal{M}$ is the fundamental matrix, is an s-component column vector with all entries 1 and $\mathcal{A}_{sq}$ represents the matrix obtained from $\mathcal{A}$ by squaring each entry.

Acknowledgments

The first author was supported by a UMSL Research Award, 2002.

References

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

[2] Caruana, R.A. and Scha er, 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] Chakraborty, U.K., Dastidar, D.G. (1991): *Chromosomal encoding in genetic adaptive search*, Proc. Int'l Conf. on Signals, Data and Systems, AMSE, Vol. 2, pp. 191-195.

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

[6] De Jong, K.A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Ph.D. Thesis, University of Michigan, Ann Arbor, MI.

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

[8] Gilbert, E. N. "Gray Codes and Paths on the n-Cube" Bell System Tech. J. 37, 815-826, 1958.

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

[10] 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.

[11] Eiben, A. E., Aarts E.H.L. and Van Hee, K.M. (1991). Global convergence of genetic algorithms: a Markov chain analysis, In H.-P. Schwefel and R. Maenner (Eds.), *Parallel Problem Solving from Nature* (pp. 4-12). Berlin: Springer-Verlag.

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

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

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

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

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

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

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

[19] Scha er, J.D. et al. *A study of control parameters a ecting online performance of genetic algorithms for function optimization*, Proc. 3rd ICGA, 1989 (Morgan Kaufmann).

[20] 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.

[21] 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.

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

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