Mutation-Crossover Isomorphisms and the Construction of Discriminating Functions

Joseph C. Culberson *†

December 1, 1994

Abstract

We compare the search power of crossover and mutation in Genetic Algorithms. Our discussion is framed within a model of computation using search space structures induced by these operators. Isomorphisms between the search spaces generated by these operators on small populations are identified and explored. These are closely related to the binary reflected Gray code. Using these we generate discriminating functions which are hard for one operator but easy for the other and show how to transform from one case to the other.

We use these functions to provide theoretical evidence that traditional GAs use mutation more effectively than crossover, but dispute claims that mutation is a better search mechanism than crossover. To the contrary we show that methods that exploit crossover more effectively can be designed and give evidence that these are powerful search mechanisms. Experimental results using GIGA, the Gene Invariant Genetic Algorithm, and the well known GENESIS program support these theoretical claims.

Finally, this paper provides the initial approach to a different method of analysis of GAs that does not depend on schema analysis or the notions of increased allocations of trials to hyperplanes of above average fitness. Instead it focuses on the search space structure induced by the operators and the effect of a population search using them.

*Supported by Natural Sciences and Engineering Research Council Grant No. OGP8053. Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2H1. email:joe@cs.ualberta.ca

†This article to appear in Evolutionary Computation Vol. 2 No. 3

1 Introduction

Despite much effort, the analysis of genetic algorithms remains somewhat problematic and the subject of much debate. In particular, the precise roles of crossover and mutation in the search process is yet unclear, and too poorly understood to yield detailed predictions about GA behavior on previously untested functions.

Traditional schema analysis (Goldberg, 1989; Holland, 1975) attempts to capture the effects of large populations undergoing dynamic changes under the influence of selection. It assumes that the population covers the search space so that there are many examples of various hyperplanes, and under these assumptions shows an exponential increase in the proportion of the above average schemata over time. Refinements of this analysis tend to focus on the disruption of schemata by the various operators used (Spears & De Jong 1991). Extensions to the theory include the building block hypothesis which suggests that schemata of short defining length and above average fitness are favored in traditional genetic algorithms (TGAs). Mühlenbein (1991) and Radcliffe (1992) discuss some of the limitations and concerns with this approach to GA analysis.

It has always been intuitively recognized that a major contribution of crossover was its capability to recombine schemata into new ones that might reflect the combination of features leading to improved strings. Mutation on the other hand has traditionally been seen as a means of introducing diversity into a population, since proportional selection causes the population to converge towards a set of highly similar individuals, effectively thwarting further exploration (Goldberg, 1989). Its disruption rate has tended to be ignored, because it is usually run at a very low rate (Spears & De Jong, 1991). Spears (1992) presents a theoretical discussion of the relative capabilities of crossover and mutation to construct new schemata. (The reader is referred to this article for a summary of the crossover–mutation debate).

Schaffer & Eshelman (1991), Schaffer, Caruana, Eshelman & Das (1989) and others have presented disconcerting evidence that traditional GAs often do as well using mutation alone as they do when using crossover. In one sense this is not surprising since almost any operator will be superior to another for some subset of all possible functions. However, their study indicates that we have little concrete theory to decide which functions will be more amenable to each of the operators. Despite the claims that crossover is the magic behind GAs, there is almost no analysis which unequivocally supports this claim, or that can be shown theoretically or experimentally

to differentiate between these operators. Fogel & Atmar (1990) make the claim that "...crossover and inversion ...do not compare favorably with more simple random mutation." It should be pointed out that their notions of mutation and crossover differ from the binary form presented here. Still, debates about this sentiment echo throughout the GA community.

In contrast to this claim, it will be shown in section 6 that there is an isomorphism induced by the mutation operator on single strings and crossover on pairs of complementary strings. This isomorphism is with respect to the *Search Space Structures* induced by these operators, which are discussed in section 2. These search space structures give us a new means of describing easy and hard functions with respect to these operators, as presented in section 3. In particular, they give us a precise way of describing the peaks and valleys of a function with respect to very simple mutation or crossover based search. For example, we show that for one-point crossover, the familiar One-Max function has exponentially many false peaks with respect to simple crossover search. This simple crossover search takes place on populations of two complementary strings. However, in section 4 it will be argued that these false peaks also negatively influence search in larger populations, and experimental evidence in section 10 will confirm this claim.

Using the isomorphism, it is shown in section 7 that any function that is hard for crossover search on a pair of complementary strings can be converted to an *identically* hard function for mutation based search on a single string, and vice versa. Further transformations and discussion are provided in section 8. In section 9 a series of conversions of the One-Max function are are generated using these transformations. Given that One-Max is easy for mutation and hard for crossover, these functions can be used to *discriminate* between various algorithms with respect to their effective use of these two operators. Thus is developed a powerful research tool which can be used to assist our understanding and design of genetic algorithms in general.

Some of the functions presented in this paper do not exhibit easily identifiable building blocks, or short schemata, that can be evaluated independently of other bits. Although the building block hypothesis cannot hold for these functions, they are easily solved to optimality by a genetic algorithm that is designed especially to take advantage of the communication powers of crossover. This algorithm, which uses no mutation, is called the Gene Invariant Genetic Algorithm (GIGA) and will be discussed in section 5. Notions of information communication within a population by crossover are presented in section 4.

In particular, we claim that the traditional GA does not effectively use

this inherent power of crossover. Both the theory developed herein, and the experiments reported in section 10 support this thesis. This confirms the observations cited above, and to some extent explains the apparent contradiction between the intuition that crossover is the driving force of GAs and the empirical observations that crossover is of minor effect in many optimization trials.

Finally, it is noted that this work is preliminary, and in the conclusion several suggestions are made for future directions in this research. Genetic Algorithms are very complex, and it is unlikely that any single analysis will both capture all the behavioral characteristics of GAs over all functions and at the same time provide useful detailed guidelines for their use in specific applications.

2 Search Spaces for Mutation and Crossover

In this section we develop the intuitions and lay the foundations we require to present the analysis.

2.1 Computational Models and Notation

GAs are often touted as "no prior knowledge" algorithms. This means that we expect GAs to perform without special information from the environment.

Our model is the following. An algorithm A generates a binary string which is then evaluated by the environment $\mathcal{E}$. This is the only means the algorithm has of communicating with the environment. The environment acts as a black box, and so we refer to this as the *black box* model.

We consider the class of functions $\mathcal{F}_l = \{f : \mathrm{S}^l \rightarrow \mathcal{R}\}$, where $\mathcal{R}$ is some fixed range and $\mathrm{S} = \{0, 1\}$. For this paper we will assume that $\mathcal{R}$ has a total ordering, and that the objective of the search is to find the optimum value of f, or some approximation to the optimum. This *function optimization* view of GAs is familiar, although GAs may be used in more general settings (De Jong, 1992).

In this paper we wish to discuss the tradeoffs between the operations crossover and mutation. *Crossover* of two strings x and y produces a new pair of strings u and v such that for each $1 \leq i \leq l$, either $u_i = x_i$ and $v_i = y_i$, or $u_i = y_i$ and $v_i = x_i$. The *Hamming Closure* ($HC(x, y)$) of two strings x, y is the set of all strings u, v satisfying these properties. A *crossover point* is a pair $i, i+1$ such that for i the first condition holds and for

$i+1$ the second condition holds, or vice versa. *One-point crossover* means that there exists exactly one crossover point. It is usually implemented in algorithms by picking the crossover point at random and then exchanging the substrings on one side of the point. *k-point crossover* in this paper means *up to k* crossover points. The reason is that in many implementations the crossover points are chosen at random and so repetitions may occur. If a point is selected twice the net effect is no crossover at that point. In *uniform crossover* for each $i, 1 \leq i \leq l$ the assignment to u_i and v_i is made independently with some fixed probability p. If this probability is $1/2$ then it is *unbiased uniform crossover*, otherwise it is biased.

Mutation of a string x produces a new string u with from 0 to l bits replaced by their complements. To be consistent with the definition of k-point crossover, we define *k-bit mutation* to mean that k elements are complemented with possible repetitions, implying that up to k elements are changed. In *Uniform mutation* each bit is changed with some fixed probability.

Although crossover and mutation appear to be quite different operators, we will demonstrate an isomorphism of the searches generated by these operations under restricted circumstances. We may then, for example, describe precisely how to convert hard problems for one operator into hard ones for the other.

2.2 Search Space Structure

In this paper, when we refer to a *problem* we mean a function optimization problem as defined in the preceding section. A *search space* for a problem is the set of all n-member populations of l-bit binary strings. Each population is called a *point* in the search space. Strings are evaluated by function f, and the *value of a point* v in the space, $\mathrm{val}_f(v)$, is the optimum of the values of the strings in the population. In its most general form, the *generalized search space structure* (GSSS) induced by an algorithm A on a search space under a function f is the digraph $G = (V, E)$ where the vertices correspond to the points, and two vertices u and v are joined by an edge if there is an operation in the algorithm that produces the population corresponding to vertex v from that corresponding to vertex u.

The definition of the GSSS is too general to be of much use in analyzing any but quite simple algorithms operating on simple problems. However, it brings to our attention the fact that an algorithm may be working on a quite different structuring of the search space than the one we may be

most familiar with. For example, the *Hamming cube of dimension* l is the graph obtained when vertices are binary strings of length l, and edges adjoin vertices when they differ in exactly one position. It is very easy to fall into the trap of thinking of the space of binary strings as being *inherently* structured by the Hamming cube, and thus intuitively believing that an algorithm is conducting its search according to this graph. Alternatively, we may intuitively believe that the search space is closely related to the interpretation of strings as binary numbers, in either the usual binary coding or in the Gray coded form. This corresponds to a graph consisting of a simple path from lowest to highest binary value. When an algorithm is generating new strings during a search it may be generating the strings in a way that has little to do with either of these graphs. For example, Jones (1994) points out that the crossover operator search space, which he terms a *landscape*, may not even be connected!

In this paper a simplified notion of a *search space structure* (SSS) will be used. The idea here is to define a graph that is induced by all possible applications of some simple operation. In this way it may be possible to abstract certain properties of particular operations that apply to many algorithms which include the operation as part of its search mechanism.

We consider only those operators which are *blind*; that is, those operators which generate new strings or sets of strings by manipulating other strings without regard to the evaluation of the strings. In particular we will define the SSSes generated by generic mutation and crossover operations on very restricted populations. We will then demonstrate an isomorphism between these spaces.

2.3 The SSSes for Basic Mutation and Crossover

The 1-bit *mutation SSS* is defined on a population of size one. Two strings are adjacent in the search space structure iff they differ in exactly one bit. This SSS is the Hamming cube, and any (possibly probablistic) local improvement algorithm restricted to moves on this graph is a Hamming Hill Climber (HHC).

Note that this structure underlies many algorithms. These include both deterministic and randomized algorithms, such as: repeatedly flip the leftmost bit that improves the value until no such improvement is possible; randomly select a bit and flip it if the value would improve; and many others.

Also note that the graph is bi-directed, since the operation is reversible,

and so we represent the SSS as an undirected graph. We will refer to the SSS for mutation on l-bit strings by the notation $\mathcal{H}_M(l)$.

The *1-point crossover SSS* is defined on populations of two complementary strings. Two pairs of strings are adjacent in the search space structure iff one can be derived from the other by a single 1-point crossover operation. This SSS is also a hypercube, as will be shown in the section 6, and any algorithm restricted to moves on this graph is a Crossover Hypercube Hill Climber (CHHC). We use the notation $\mathcal{H}_X(l)$ to refer to the graph of the crossover SSS for $l+1$-bit strings, since it is a hypercube of dimension l.

3 Easy and Hard Functions

The definitions of hardness and easiness of functions may be made with respect to specific algorithms, or with respect to certain operators. Furthermore, many gradations of hard and easy can be defined, which adds further confusion. Intuitive notions do not always bear relation to the reality of the search space. There has been much research and controversy into the notions of deception, for example see Goldberg (1989); Goldberg, Korb & Deb (1989); Deb & Goldberg (1992); Grefenstette (1992).

We will base our definitions on the SSS induced by a blind operator. The assumption is that an algorithm will use the operator to generate neighboring points, and then based on the evaluation of those points and the current population, generate a new population. An algorithm using the operator may exclude certain transitions, or under the influence of a particular function reduce the likelihood of certain transitions. If the algorithm uses other operators, then it may choose to ignore the contribution from this operator. Or it may choose to apply the operators in combination so that the SSS is part of a combined operator SSS. However, we will assume that the properties of an SSS limits its applicability even when used as only part of a more general algorithm. Although this is not necessarily true, it gives us a starting point and for the SSSes we consider, the experimental evidence tends to support this view.

A v, w-*path* in an SSS is any sequence of points $v = v_0, v_1, v_2, \ldots, v_k = w$ such that v_i, v_{i+1} for $0 \leq i < k$ is an edge. The *length* of a path is the number of edges(k). The *distance* between two vertices is the length of the shortest path joining them. The *diameter* d of an SSS is the greatest distance between two vertices. The *degree* of an SSS is the largest degree of any vertex.

The diameter and degree are important measures of an SSS. If the diameter is too great then no algorithm restricted to the SSS can be efficient since in the worst case we may have to traverse the entire longest path. Similarly, if the degree is too large, then either we must randomly sample the neighbors of a point, or we must generate them all. The graph gives too little information, so too much work must be done to find a good neighbor. If the graph is a complete graph then the best search is a random search.

The diameter and degree of an l-dimensional hypercube are both l. We henceforth assume the SSS is a hypercube as in mutation and crossover, although the definitions and observations have wider application.

A function is *non-misleading* for an SSS if for every point in the SSS there is a monotonic (with respect to the values of the points) path to a global optimum. That is, f is non-misleading on an SSS G if for each point $v \in G$ there is a v, w-path $v = v_0, v_1, \ldots, v_k = w$ such that $\mathrm{val}_f(v_i) \leq \mathrm{val}_f(v_{i+1}), 0 \leq i < k$ and $\mathrm{val}_f(w) = \max_{u \in G}\{\mathrm{val}_f(u)\}$. The function is *directive* on G if for every point in G there is a strictly monotonic path to a global optimum; i.e. $\mathrm{val}_f(v_i) < \mathrm{val}_f(v_{i+1})$. The distinction is important. The function which is everywhere zero, except for one-point of value one will be non-misleading on any SSS, but nothing better than a random search can be applied to such a function. However, a directive function for an SSS means that a deterministic algorithm can always make an improvement to a non-optimal point by evaluating each of the neighbors.

It might seem that *directive* captures the distinction between those functions for which an SSS represents an efficient search mechanism and those for which it does not. However, this is not the case. Horn, Goldberg & Deb (1992) have shown that there exist exponentially long induced paths in the hypercube. These are called 'snake-in-the-box' (Harary, Hayes & Wu, 1988) or *distance preserving codes* (Reingold, Nievergelt & Deo, 1977). These can be used to create directive functions that require on average an exponential amount of work for a hypercube hill-climber.

We say a function is *polynomially efficient* or *efficient* for an SSS G if it is directive and the expected length of a strictly monotonic path from a randomly chosen point to a global optimum is polynomial in the diameter d. It is *linear* if the expected path length is $O(d)$. It is *strictly linear* if it is directive and every steepest path is a shortest path. A path is a *steepest path* if for each i v_{i+1} has the greatest value of any neighbor of v_i. We note that random search induces an SSS on which every function f is strictly linear since the graph is a complete graph (on populations of size one). We repeat the observation that an SSS is most useful if it has small degree and

diameter.

In the sense of non-misleading given above, almost all functions are misleading. Any hill-climber on an SSS will be misled to some extent by the presence of points in the SSS which are not on monotonic paths to an optimum. The difficulty will increase as the number of such points increases, and as the number of strictly monotonic paths leading to such points increases. The degree to which a function misleads an algorithm can be measured in several ways, each of which implies different function characteristics for maximal effect. For example, we could consider the distance from the optimal point in the SSS, the difference in value from the optimal point, the difference in rank, the likelihood of being misled or the number of ways of being misled. We could also consider the distance an algorithm is misled in some other structuring of the search space, for example the difference $|bin(x) - bin(y)|$ where x is an optimal string under f, y is a suboptimal string produced by the algorithm and bin is the value of the string interpreted as a binary number. All of these are affected in different ways if the function is allowed to have more than one global optimum.

For this paper, we will assume that the presence of many (e.g. exponential in l) false peaks in the SSS is sufficient evidence that the operator is not likely to be efficient on the function.

3.1 An Example: The One-Max Function

As an example we will look briefly at the One-Max function, also called the ones counting function. This function is defined as

$$f_u(x) \quad = \quad \sum_{i=1}^{l} x_i$$

With respect to the one-bit mutation operator, this function is strictly linear. An optimal deterministic HHC needs to make at most $\binom{l+1}{2} + 1$ string evaluations to reach the maximum from any point in the SSS.

Under one-point crossover the function is not so well behaved. Figure 1 and figure 2 illustrate that the crossover SSS has false peaks for $l = 5$ and 6, that is suboptimal points for which every neighbor has lower value. Each node of the graph represents a complementary pair of strings (only one of which is shown in figure 2). The value of a node is the number of ones in the string with the maximum number of ones. The string pair $(0^l, 1^l)$ is the

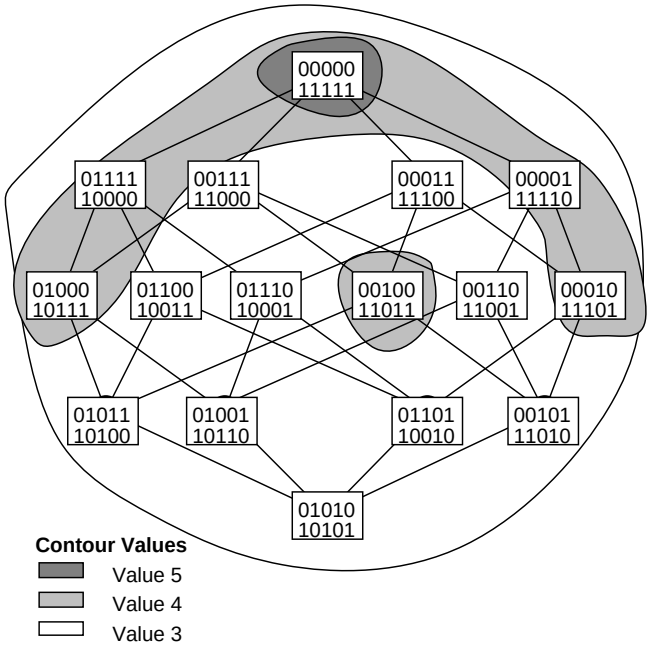

Figure 1: Crossover SSS with Values from One-Max for $l = 5$

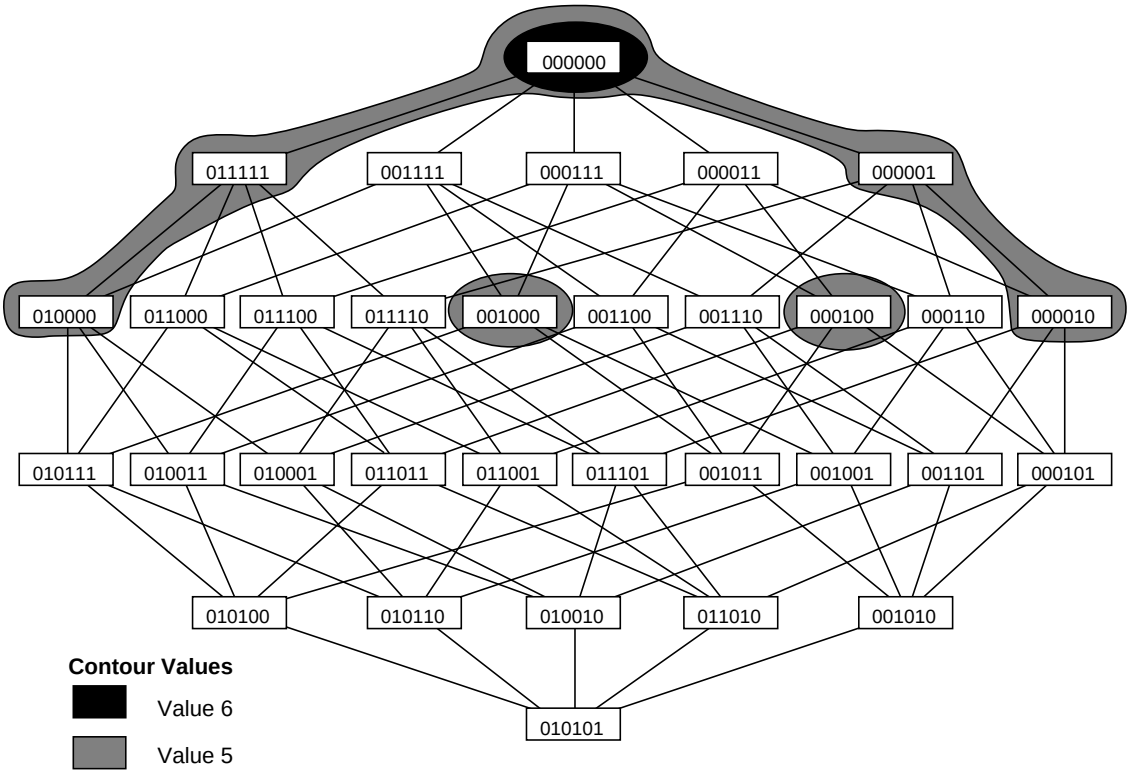

Figure 2: Crossover SSS with Values from One-Max for $l = 6$

l	Peaks	Fraction Increase	l	Peaks	Fraction Increase
5	2	2.000	18	3769	1.870
6	3	1.500	19	7176	1.904
7	5	1.667	20	13532	1.886
8	9	1.800	21	25842	1.910
9	15	1.667	22	49113	1.901
10	28	1.867	23	93995	1.914
11	49	1.750	24	179775	1.913
12	91	1.857	25	344796	1.918
13	166	1.824	26	662676	1.922
14	307	1.849	27	1273880	1.922
15	574	1.870	28	2457275	1.929
16	1065	1.855	29	4735080	1.927
17	2016	1.893	30	9158972	1.934

Table 1: The Number of Peaks in the Crossover SSS for One-Max

maximum value, all other peaks are false peaks. (The case is similar if we are minimizing.)

We would expect that crossover would have more difficulty with this function than would mutation based search. In particular, if the search is elitist, then it is possible that it could terminate on a false peak.

The problem becomes much more acute as l increases. We do not have an exact formula for the number of peaks, but it increases exponentially with l as proved in the following theorem. In fact, the number of peaks appears to grow at a rate greater than $O(1.93^l)$ as evidenced by the fraction increase in table 1. We suspect that the fraction increase in the number of peaks has a limit of two. Note that for $l \leq 4$ the number of peaks is one.

We let $\phi = (\sqrt{5}+1)/2 = 1.618\cdots$ (the golden ratio).

THEOREM **3.1** *The number of peaks in the crossover SSS for f_u is greater than $C\phi^l$ for a constant C.*

Proof: The proof hinges on the observation that for $l > 4$, $(x, \bar{x})$ is a peak iff for each j, crossing at j will produce a pair of strings each of which has fewer 1-bits than $\bar{x}$, assuming that $\bar{x}$ has the maximum of the pair. This holds when the number of zeroes exceeds the number of ones in both $x_1 \ldots x_j$ and $x_{j+1} \ldots x_l$ for each j. (Or equivalently, the number of ones exceeds the

number of zeroes in the complement string $\bar{x}$.) Note in particular that the first and last two bits must be zeroes whenever x is a peak.

For $l = 5$ there are two peaks, 00000 and 00100. Given a peak x for $l = k$ we can create two peaks for $l = k + 2$ by inserting 00 and 10 after x_2. Applying this recursively shows that the number of peaks is at least $2^{(l-4)/2}$.

To improve this lower bound, we observe that if $x_3 = 0$ then we can create an additional peak of length $k + 2$ by inserting 01. Let a_i equal the number of peaks with bit 3 equal 1, and b_i the number of peaks with bit 3 equal 0, after $i-1$ insertions. We use $a_1 = b_1 = 1$ for the base case $l = 5$. We may create type a strings by inserting 10 in either type a or type b strings. Similarly, we may create type b strings by inserting 00 into either type a or b strings, or by inserting 01 into type b strings. Thus,

$$\begin{aligned} a_i &= a_{i-1} + b_{i-1} \\ b_i &= a_{i-1} + 2b_{i-1} \end{aligned}$$

From these we determine that $a_i = 3a_{i-1} - a_{i-2}$, and observing that $a_1 = 1$ and $a_2 = 2$, we obtain

$$a_i = \frac{1}{\sqrt{5}} \left(\phi^{2i-1} + \phi^{1-2i} \right)$$

The total number of peaks created is $a_i + b_i = a_{i+1}$. Dropping the second term in the expansion of a_{i+1}, and noting that i corresponds to a string of length $l = 2i + 3$, we find the theorem is satisfied for odd l by

$$C = \frac{1}{\sqrt{5}\phi^2} = 0.1708\cdots$$

A similar result holds for even l using $l = 6$ as the base. ■

Although the number of false peaks grows exponentially, this still does not prove that a random CHHC will be more likely to reach a false peak than the optimal one. Nevertheless we suspect that this is the case, and conjecture that One-Max is a difficult function for crossover using a population of size two.

Thus, One-Max is a function which apparently differentiates between crossover and mutation on trivial populations. In the next section we consider how this relates to larger populations.

4 The Effects of Larger Populations

In this section we will take a slightly different view of the search process, in order to briefly consider the interactions of larger populations with the mutation and crossover operators. This section is useful in understanding the design of GIGA presented in section 5 and the experimental results reported in section 10.

We identify three influences of a larger population. First we note a limited effect on mutation only search. The other two effects are concerned with crossover. The first is the power of focusing that occurs when two similar strings are mated. The second crossover effect requires a certain diversity of population that is usually lacking in TGAs. It is a migration effect dependent on the presence of one or more sequences of pair-wise similar strings over which crossover can migrate substrings or patterns found anywhere in the sequence. The efficacy of this migration is greatly influenced by the selection mechanism. We now describe each of these influences in more detail.

In figure 3 we illustrate the notion of a search using a larger population where mutation is the only operator. We assume selection is done proportionately and that a new population is generated by selecting parents and performing one-bit mutations.

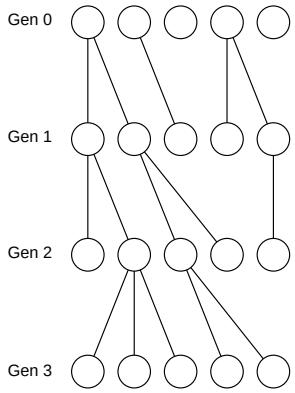

Figure 3: The Interaction of Mutation with a Population

Under these conditions, the effect of a population is that several mutation searches are carried out in parallel. Communication between members of the

population is restricted to the selection mechanism; that is, certain searches are truncated because other searches are doing better. In this particular example by the third generation every string is a descendent of one member of the initial population. This means that the Hamming distance between any two strings in the third generation is at most 6. This convergence occurs rapidly in TGAs, even in the absence of selection pressure (Louis & Rawlins, 1991).

The edges in this search structure are just the edges of the Hamming Cube associated with the mutation SSS. Thus, this population is searching points 3 steps from a particular vertex of the Hamming Cube.

With such a search mechanism, if the mutation SSS has many false peaks, then it is highly probable that the search will be unsuccessful. This is because each of the searches will be fooled in a similar manner, and the restricted communication between members will not let this fact be known.

If we use crossover in the search, then other effects of a larger population may become apparent. The first of these is the focusing power that occurs when two non-complementary strings are mated. Basically, this focusing power is the restriction of a search to the Hamming closure of two or more strings.

For example, we have shown that when two complementary strings are involved in a one-point crossover search of One-Max, the number of peaks is exponential in the length l. However, when the population is larger than two, crossover will seldom involve complementary strings. In the case of One-Max, we need only consider those positions in which the two strings differ. If the Hamming Distance between two strings x and y is $H(x, y)$, then the crossover SSS on those two strings is isomorphic to one on a pair of complementary strings of length $H(x, y)$. Thus, if the population is large, we would hope to reduce the peaks significantly between adjacent strings. Suppose for example that we have two strings each with value $l - 2$. The Hamming Distance cannot exceed 4 in this case, and so there can be no false peaks in the Hamming Closure. Thus, repeatedly mating these two strings using one-point crossover will eventually produce optimal strings, provided the Hamming Distance is exactly 4.

Even when the Hamming distance is greater than 4, the distance between the strings will be far less than l in most cases. But this implies that the number of false peaks is reduced *exponentially* from the case of a complementary pair of strings in terms of the Hamming distance between them. This illustrates the focusing power of populations using crossover. It has some similarity to search by population based mutation, but the search

is focused on the region where the two strings differ.

This example also shows that in some cases it is worthwhile mating strings which are "near-by". In this example, "near-by" is the same whether the distance is measured by value or by Hamming distance. Of course, mating strings with $H(x, y) < 2$ cannot produce new strings. If the locality exhibited by One-Max does not hold, as for example in a randomly chosen function, then mating "near-by" strings may not be helpful.

The tendency of TGA populations to converge may severely limit the exploratory capabilities of the crossover operator. The crossover operator can only search within the Hamming Closure of the members of the current population. Once the population is converged to a set of highly similar individuals, the search becomes too narrowly focused.

If the population is kept diverse, then crossover may be used to encourage 'migration'. Consider once again the One-Max function. If two strings which are nearly equal in value are mated, then there is a high probability their offspring will have a greater difference in value. The ones are unlikely to be equally distributed over the two offspring, unless the parents are both all ones or both all zeroes. Now if the mating process is *always* restricted to pair strings close in value, the larger valued offspring will eventually by mated with another one of nearly equal value, and again there will be a separation of ones and zeroes. Also, the lower valued offspring will eventually mate with another of nearly the same value, and the same process of separation will be repeated. Thus, on the One-Max function, restricting mating to pairs that are close in value will encourage a separation of ones from zeroes, with the ones migrating to form strings of higher values and zeroes migrating to form strings of lower value.

For more complex functions, it may be that certain substrings or patterns may arise more readily in strings of lower value. For example, in deceptive functions (Deb and Goldberg, 1992) substrings of zeroes are assigned a maximum value, while strings containing ones have value which increases with the number of ones. With the separation process described above, eventually strings containing substrings of all zeroes will be produced. Since these will all tend to be of high value, they will tend to mate with one another. Eventually optimal substrings will combine to form a string of optimal value.

TGAs do not maintain enough diversity or enforce the mating of pairs which are close in fitness value, to achieve the full benefit of this migration process. However, it is possible to greatly enhance these effects. The Gene Invariant Genetic Algorithm (GIGA) described in section 5 ensures that the population never converges, and that pairs close in value are most likely to

be mated. Functions of the deceptive type are studied in Culberson (1992a) using GIGA. Most deceptive functions found in the literature were solved easily.

Nevertheless, functions such as One-Max which have an exponential number of false peaks for crossover may still prove difficult. Culberson (1992a), using GIGA, found that One-Max remained difficult for a crossover only search. With small populations and one-point crossover, it seldom succeeds. With populations of size 10 to 20 success was usually but not always achieved, but required several thousand strings to be produced. More detailed experiments in section 10 in this paper show similar results for a modified One-Max.

Why do false peaks in the 2 string case imply difficulty even for larger populations? A complete formal analysis of the effects of the population has not yet been obtained. Considering again One-Max, suppose the population is of size 4, with $l = 100$. Then if the strings are in some sense "equally distributed" we might still expect the top two to have Hamming distance on the order of 20 or more. But using the data from table 1, this would imply possibly thousands of false peaks. A moment's thought would indicate a similar number of false peaks between each pair of strings. Thus, progress would be difficult due to insufficient focusing.

Of course this argument is quite informal, but it gives the impression that false peaks in the crossover SSS imply difficulties for crossover even for larger populations in many cases. Experimental evidence presented in section 10 supports this intuition.

In the next section we show how these insights are used in the design of a different genetic algorithm, the Gene Invariant Genetic Algorithm.

5 Brief overview of GIGA

Since most readers are likely unfamiliar with the Gene Invariant Genetic Algorithm (GIGA) we will briefly describe the program and the various parameters it uses. More detailed descriptions and various experiments can be found in Culberson (1992a & 1992b).

GIGA is designed to illustrate the power of crossover, and uses no mutation. Its design reflects the considerations of the preceding section. For each iteration of the population updating cycle, GIGA selects a pair of parents, usually a pair that is close in fitness value, and produces a family of pairs of offspring through crossover. The best of these pairs replaces the parents

in the population. In this way, the multiset of binary values in any column of the population matrix remains unchanged over time. This property is referred to as *genetic invariance*.

In all cases in this paper we wish to minimize the function, and so we select as the *best* that pair which has the string of least fitness value in the family. Note that we select a *pair*, one of which is judged to be best in the family of pairs. The GIGA program allows other definitions of "best", including for example the pair with maximum difference in fitness value (Culberson, 1992a & 1992b).

The chief difference between GIGA and a TGA then is in the replacement policy. But this is crucial, because now the population *cannot converge* in the usual sense. Perhaps it is worth mentioning that originally GIGA was designed to be a GA which explicitly violated the conditions of the schema theorem. In fact, in the version studied by Lewchuk (1992) an offspring pair always replaced its parents independent of fitness value, and a pair was selected for mating if it was closest in fitness value over all pairs in the population. Thus, *never* was there direct selection pressure by above average fitness value, as required by the schema theorem. Nevertheless, this restricted version of GIGA was able to remain competitive with TGAs over a variety of functions.

GIGA does not use mutation, and therefore it is important that the initial population have each character of the alphabet (in our cases $\{0, 1\}$) available for each position. Thus, *Random Rotation* generates strings in randomly selected complementary pairs, and a single unpaired string if the population is required to be odd.

GIGA is designed to exploit the migration of information through a population, and so it is generally (but not always) advantageous to mate strings that are close in fitness value as explained in the previous section. To this end the population may be kept in sorted order.

GIGA achieves global effects as a side effect of local changes in the population. An adjacent pair is selected from the population and produces a family of pairs by applying crossover. The best *pair* of this family is then selected to replace the pair of parents, thus maintaining the invariance property. As a reminder to the reader, genetic invariance requires that the number of 1-bits in any column of the population matrix remains unchanged throughout the evolutionary process.

Note that this is a non-generational process, reminiscent of Whitley's GENITOR program (Whitley, 1989). As soon as a replacement pair is selected they are free to participate in the very next mating. In fact, if

sorting were not enforced selecting an adjacent pair would guarantee that one of the offspring of any mating would be a parent in the next mating, assuming that we mate adjacent pairs sequentially. Alternatively, a family may be seen as a small subpopulation whose only offspring are then merged into the population at large. This is also somewhat like an extreme version of Mühlenbein's parallel genetic algorithm approach (Mühlenbein, 1991 & 1992) although GIGA is not used in a parallel mode in these experiments.

Recently, in the context of genetic programming, Altenberg (1994) and Tackett (1994) have suggested a similar approach using the notion of *soft brood selection* from population biology. In each case only the fittest of the brood are selected, and thus selection is "soft" because the number of offspring produced does not influence the number contributed to the next generation. Altenberg also suggests the notion of *"Upward-mobility" selection* which is similar to the notion of local elitism described below. Tackett suggests that the brood selection function can be cheaper, in terms of CPU time to evaluate, than the parent selection function. In some environments, both memory and CPU usage may be reduced over the alternative of using larger populations. Neither of these techniques ensure genetic invariance. In neither case are the parents necessarily replaced by the offspring, and although Tackett assumes two offspring are selected, the selection is not necessarily a *pair* in the sense defined above.

Because GIGA is non-generational, the effects of varying population size are different than the effects on a TGA, and thus comparisons between these algorithms based on population size are largely meaningless. For this reason, all experimental comparisons are based on the number of evaluations required to obtain an optimal solution, or the best values after some number of evaluations have been performed.

If elitism is in effect, as it is in the experiments in section 10, then the parents are also included in the offspring selection process. This *local elitism* ensures global elitism as a side effect since the population will contain the best string ever seen after each mating. The previous best will only be replaced if one of the parents is that best string, and one of the offspring is better than this parent.

The order in which parent pairs are chosen can be varied, but in these experiments we use *Alternating Adjacent Pairs*. We start at the low end of the population and work our way to the other, and then back down again, repeating the cycle when we reach the bottom. After each replacement of parents by offspring, the order of the population is adjusted to keep it sorted by fitness value.

Selection pressure comes in two forms in GIGA. The first is the general pressure towards better strings induced by selection of the pair with the better fitness value from each mating. However, because we select a pair formed by some crossover operation, the other member of the pair is not necessarily an improvement. In fact if there is the usual expected correlation between schema and fitness value, the other member is likely to be worse. Thus, in these cases the effect is to create a more diverse population with respect to the fitness values of strings.

Secondly, there is selection pressure induced by the sorting of the population and the selection of adjacent members of the population. Even if explicit sorting is turned off this pressure is still manifest, although considerably slower in action. In fact, even if the family size is restricted to one pair, and this pair always replaces the parents regardless of fitness value, some evolutionary pressure is maintained in the population, provided that the better child always replaces the parent of higher index. The population gradually becomes more diverse, with the higher indices holding strings of higher fitness value as time passes. Note that it is not necessary to mate strictly adjacent members of the population for these effects to be maintained. One might liken this to natural evolution where the migration of creatures into different ecological niches implies they are more likely to mate similar individuals.

These selection pressures stand in sharp contrast to those in most GA's, where selection pressure is towards a convergence of the population to a set of highly similar individuals.

The GIGA program is fairly simple minded. It only stops after the specified number of matings have been made. A mating is the production of a family. Thus, the total number of evaluations of strings will be the product of the family size and maximum iterations plus the initial population size. Each time there is a change in the overall maximum or minimum value found, this is reported, and so we can determine the number of evaluations required to find the minimum.

GIGA has the capability of using a mixture of crossover operators. For each pair produced, one of the operators is chosen randomly from the allowed set, with probability proportional to the rate out of the total rate. When uniform crossover is used, the *swap bias* indicates the probability of a parent bit going to a specific child. The selections are made independently on a bit by bit basis. We use 0.5 throughout the experiments in this paper, which means that each child is equally likely to receive each bit from either parent when uniform crossover is used.

6 The Crossover-Mutation Isomorphism

We now return to the SSS model of computation for search on populations of size one or two. The analysis in this section shows that the crossover SSS $\mathcal{H}_X$ is a hypercube by demonstrating an isomorphism to the mutation SSS.

Between two hypercubes of dimension l there are $l!2^l$ isomorphisms. We may map the string 0^l onto any other and then choose any permutation of its neighbors. The set of isomorphisms from $\mathcal{H}_M(l)$ to $\mathcal{H}_X(l)$ we refer to as the MX transformations. We will use $\mathcal{I}$ as the canonical isomorphism, which maps 0^l, onto $(0^{l+1}, 1^{l+1})$, and the ith bit onto the ith potential crossover point. It should be noted however, that any of our proofs could use any of these isomorphisms.

A crossover-mutation isomorphism may be obtained operationally by mapping the ith bit of an l-bit string under mutation to the ith crossover point of an $l+1$-bit complementary pair of strings under crossover, assuming one-point crossover and one-bit mutation.

More formally, we define a mapping

$$\begin{aligned} \mathcal{I}(x) &= (a, \bar{a}) \\ a_i &= \begin{cases} 0 & \text{if } i = 1 \\ x_{i-1} \oplus a_{i-1} & 1 < i \leq l+1 \end{cases} \end{aligned}$$

where $\oplus$ is the *"exclusive or"* or *"sum mod 2"* function and x_i is an element of $\{0, 1\}$. Noting that $x_1 \oplus 0 = x_1$, an equivalent expression is $a_i = x_1 \oplus x_2 \oplus \ldots \oplus x_{i-1}, i > 1$. As an example, $x = 10111$ maps to $(a, a') = (011010, 100101)$ under $\mathcal{I}$.

We now show that this mapping is an isomorphism from $\mathcal{H}_M(l)$ to $\mathcal{H}_X(l)$.

Lemma 6.1 *(x, y) is an edge in $\mathcal{H}_M(l)$ iff $(\mathcal{I}(x), \mathcal{I}(y))$ is an edge in $\mathcal{H}_X(l)$.*

Proof: We have that $x =< x_1, x_2, \ldots, x_l >$ and $y =< y_1, y_2, \ldots y_l >$, where each $x_i, y_i \in \{0, 1\}$. x, y is an edge in $\mathcal{H}_M(l)$ iff there is exactly one $1 \leq i \leq l$ such that $x_i \neq y_i$. Assume w.l.o.g. that $x_i = 1$, $y_i = 0$.

Let $\mathcal{I}(x) = (u, \bar{u})$ and $\mathcal{I}(y) = (v, \bar{v})$. Note $u_1 = v_1 = 0$. Then $((u, \bar{u}), (v, \bar{v}))$ is an edge in $\mathcal{H}_X(l)$ iff there is exactly one $1 \leq k \leq l$ such that $k, k+1$ is a crossover point as defined in section 2.1; that is, if for $j \leq k$ $u_j = v_j$ and for $j > k$ $u_j = \bar{v}_j$. Simple induction shows that this is true for $k = i$ exactly when (x, y) is an edge in $\mathcal{H}_M$. ■

The inverse mapping is

$$\begin{aligned} \mathcal{I}^{-1}((a,\bar{a})) &= x \\ x_i &= (a_i \oplus a_{i+1}),\ 1 \le i \le l \end{aligned}$$

This last is very close to the mapping for transforming the standard binary number encoding to the binary reflected Gray code (γ) (Reingold, Nievergelt & Deo, 1977; Bäck, 1993).

$$\begin{aligned} \gamma(x) &= y \\ y_i &= \begin{cases} x_i & \text{if } i = 1 \\ x_{i-1} \oplus x_i & 1 < i \le l \end{cases} \end{aligned}$$

The inverse Gray code is the same as $\mathcal{I}$ except that the leading 0 is truncated. $\gamma^{-1}(x) = y$ produces a string y in which $y_i = \text{odd parity}(x_1, \ldots, x_i)$.

7 Function Transformations

Using $\mathcal{I}$ and $\mathcal{I}^{-1}$ we can easily convert hard functions for mutation into hard functions for crossover and vice versa, and similarly for easy functions. This is the basis of the statement in the introduction that mutation and crossover on trivial populations are equally powerful operators.

To convert a function $f^{(0)}$ with certain characteristics under mutation to a function $f^{(1)}$ with the *identical* characteristics under crossover (on a population of size two) we apply $\mathcal{I}$ to each string to obtain the new pair of strings with the same value. Thus,

$$\begin{aligned} f^{(1)} &= \mathcal{I}(f^{(0)}) \\ f^{(1)}(x) &= f^{(1)}(\bar{x}) = f^{(0)}(\mathcal{I}^{-1}(x,\bar{x})) \end{aligned}$$

We emphasize that in this case x and $\bar{x}$ will receive the same value. This is consistent with our earlier definition in which the value of the population is the value of the optimal element of the population. A much larger class of functions can be created which induce the same search characteristics as this function by assigning to exactly one of each complementary pair any less optimal value. This enlargement is possible because in crossover space the number of points is one half the number of strings.

Similarly, to convert a problem from crossover to mutation we use

$$\begin{aligned} f^{(-1)} &= \mathcal{I}^{-1}(f^{(0)}) \\ f^{(-1)}(x) &= \mathrm{Opt}\{f^{(0)}(\mathcal{I}(x)_1), f^{(0)}(\mathcal{I}(x)_2)\} \end{aligned}$$

Again, the notation reflects the fact that the *apparent* value of a string a in crossover space is the optimum of a and $\bar{a}$. Notice how this means the conversion is dependent on whether the optimization is a minimization or a maximization.

To avoid such dependencies we restrict our discussion to complementary equivalent functions. A function $f : \mathrm{S}^l \rightarrow \mathcal{R}$ is *complementary equivalent* if $f(x) = f(\bar{x})$ for all x. For any function f, we can create a complementary equivalent function $f' : \mathrm{S}^{l+1} \rightarrow \mathcal{R}$ by padding with one bit so that $x = x'_2, \ldots, x'_{l+1}$. Then

$$f'(x') = \begin{cases} f(x) & x'_1 = 0 \\ f(\bar{x}) & x'_1 = 1 \end{cases}$$

In general, f' will not have the same characteristics under either operation as f. For example, the number of maximum points will double. However, certain properties will be preserved.

Lemma 7.1 *If f is non-misleading, directive or there exist strictly monotonic paths of bounded length from each vertex to the optimum under mutation then the same is true for f'. The converse is not true, and in fact f' can be strictly linear even though f is not non-misleading.*

Proof: The proof of the first statement is trivial since each condition will be true for each of the two hypercubes wherein the added bit is fixed.

As a counter-example to the converse, consider an f with a unique optimum and which is strictly linear for mutation with the single exception that the complement of the optimal point has the second largest value. Thus, f is not non-misleading for mutation. The mutation SSS for the strings of length $l + 1$ will have an edge from each second optimum under f' to the opposite optimum, and so will be strictly linear. ■

Thus, generally speaking, the complementary equivalence transformation may make functions that are easier for an HHC, but not harder ones. For example, the complementary equivalent form of Goldberg's k-bit deceptive functions (Goldberg, 1989) are non-misleading for mutation, although the

iterated forms are not. Also, note that we could insert the complementary control bit in any location without changing the results, so this is just one of a class of equivalent transformations.

Now let us transform the function f' using $\mathcal{I}^{-1}$. Since f' is complementary equivalent, we only need consider the $l+1$-bit strings that start with 0, and these have the value of f without the leading zero. Thus, we can use γ^{-1} on the truncated l-bit strings and

$$\begin{aligned} f'^{(-1)}(x) &= \mathcal{I}^{-1}(f')(x) \\ &= f(\gamma^{-1}(x)) \end{aligned}$$

Since mutation and crossover SSSes are isomorphic, there is an analogous transformation to complementary equivalence that creates transformed functions which are no harder than the originals for crossover. We leave its definition as an exercise for the reader.

8 Extensions to the Isomorphisms and Other Equivalences

Until now we have concentrated on one-point crossover and one bit mutation. The isomorphisms also apply to multi-point crossover and multi-bit mutation. In particular, k-point crossover is isomorphic to k-bit mutation and uniform crossover is isomorphic to uniform mutation. Here k-bit mutation is assumed to mean that up to k bits are mutated.

Two point crossover includes one bit mutation in the following sense. If we restrict two point crossover to two adjacent crossover points, allowing the positions before x_1 and after x_l as legitimate crossover points, then the resulting pair of offspring will each have changed in exactly one bit. Since the strings are complementary, this is effectively a mutation. Thus, two point crossover includes one bit mutation as a special case. Similarly, $2k$-point crossover includes k-bit mutation. For this reason, One-Max is non-deceptive under k-point crossover for $k > 1$, assuming as we do throughout that we allow any number up to k crossover points to be selected. The results are different if exactly k points are selected. In fact for $k = 2$ the exact k-bit mutation SSS (and thus by isomorphism the crossover SSS also) is not even connected, since only strings of the same parity can be generated.

Although One-Max has no false peaks, for any fixed k it is easy to create functions for which the SSS does have false peaks. We can then use the

isomorphisms to convert between crossover and mutation functions as we do for $k = 1$.

The SSS induced by 2-bit mutation (and by 2-point crossover) is a chorded hypercube, in which every path of length 2 has an edge added between the first and last vertex of the path. Equivalently, each 2-dimensional facet has two diagonals added to it. Since there are $\binom{l}{2}$ 2–D facets to each vertex, this means the degree of the graph increases quadratically to $\binom{l+1}{2}$. The diameter is halved.

In search terms, the trade-off is between a larger branching factor in the search space versus longer paths from an arbitrary point to the optimum point. With longer paths and smaller degree, there is more chance of creating false peaks far (in terms of distance in the SSS) from the optimum. On the other hand with larger branching factors, we must spend more time exploring to determine the best direction in our neighborhood.

With $k = l$ the SSS is a complete graph, which reduces the search (under our restricted population criterion) to a random search. Note that this is the case when each bit is mutated with probability 1/2 or when we have uniform crossover with $p = 1/2$. If $p \neq 1/2$ then the SSS is still the complete graph, although different probabilities will apply to different edges depending on the number of bits that are mutated. In our restricted populations uniform crossover and uniform mutation are *equivalent* search spaces. There are no false peaks in the uniform SSS.

8.1 Iterating The Gray Code

Recall that the inverse Gray code $\gamma^{-1}(x)$ produces a string y in which $y_i =$ odd parity$(x_1, \ldots, x_i)$. Equivalently, the mapping γ^{-1} may be defined by

$$\begin{aligned} \gamma^{-1}(x) \quad &= \quad y \\ &\text{where} \quad y_i = \left(\sum_{j=1}^{i} x_j \right) \bmod 2 \end{aligned}$$

Since γ^{-1} is an isomorphism of $\mathrm{S}^l \rightarrow \mathrm{S}^l$ it induces a permutation. Let $\gamma^{-(i+1)}(x) = \gamma^{-1}(\gamma^{-i}(x))$, $\gamma^0(x) = x$. The following theorem describes the cycles induced by this sequence of mappings. We use the notation $x^i = \gamma^{-i}(x)$.

THEOREM **8.1** *If $m = 2^k$ where k is such that $2^{k-1} < l \leq 2^k$ then $x^m = x$ and if $x_1 = 1$ then for $0 < m' < m$, $x^{m'} \neq x$.*

This theorem is proved in Appendix I.

In figure 4 we show the orbits of each vertex of the Hamming cube for $l = 5$ under γ^{-1}.

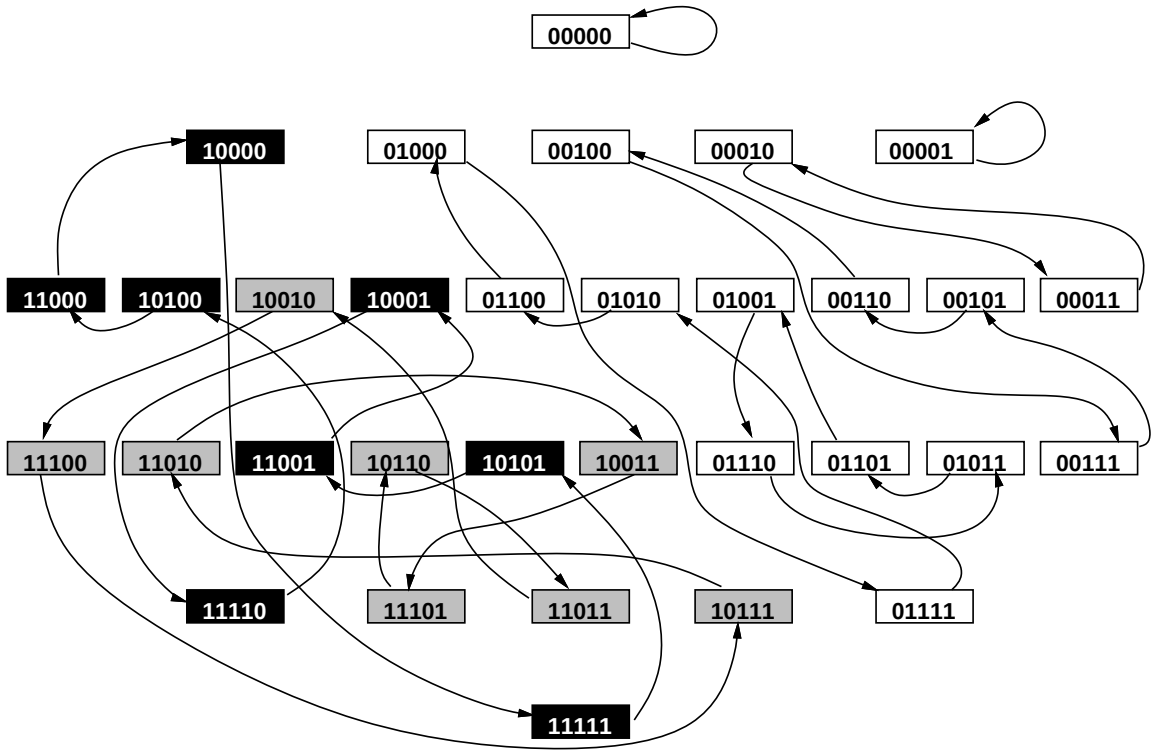

Figure 4: Orbits under γ^{-1}

These illustrate the cycles discussed in the proof of theorem 8.1. Applying γ^{-1} iteratively yields $2^{\lceil \log_2 l \rceil}$ different search spaces for mutation or crossover. Each of these is a hypercube of dimension l. We could do several searches under different numbers of iterations of γ^{-1} and with luck one of them would be a search space that is not too deceptive. We remind the reader, however, that applying γ (or any other transformation) to *all* functions will yield no change in the *average* search cost (Battle & Vose, 1991).

It was observed by Culberson (1992a) that the function $bin(x)$, the binary number function, is easy for both crossover and mutation. In fact, it is

strictly linear for both. $bin(x)$ is easily solved by either GENESIS or GIGA under many parameter settings.

If we look at how binary numbers are mapped onto the cycles in figure 4, we notice that (from a minimization viewpoint) the vertices on the shorter cycles receive values of greater fitness than the longer ones. Also, in the underlying Hamming cube, for every vertex in a cycle, there is always an edge to some vertex in a cycle closer to the optimum.

Generalizing, every element of a cycle generated by γ^{-1} has the leading one in the same position. Mutating this bit generates a member of a different cycle, and this scheme forms a natural hierarchy on the cycles. Noting that the cycles reflect the mapping of the strings onto the crossover hypercube (with a prepended 0), it is easy to see that

Lemma 8.1 *If values are mapped onto strings such that for any x in a cycle generated by γ^{-1}, the value is strictly less optimal than every y in any cycle above the cycle containing x in the hierarchy, then the resulting function is strictly linear for both mutation and crossover.*

In fact, we can transform the function using any number of iterations of γ^{-1} and the function will still be strictly linear for both mutation and crossover. We call such functions *always linear*.

It is tempting to assume that we can construct always-hard functions by carefully violating monotonicity on the orbits induced by γ^{-1}. Although this may be true for mutation and crossover search spaces on trivial populations, it may not produce genuinely hard functions for larger populations because the approach ignores the power of population. It also ignores multi-point operators. This area is left for future research.

We noted earlier that there are $l!2^l$ mutation-crossover isomorphisms. The canonical one led us to the Gray code. In general, we can create equivalent transformations by first choosing a string $b \in \{0,1\}^l$ and a permutation π of its neighbors. Then define

$$\begin{aligned} \gamma^{-1}_{b,\pi}(x) &= y \\ &\text{where} \\ y_{\pi(i)} &= \text{odd parity}(x_{\pi(1)} \oplus b_{\pi(1)}, \ldots, x_{\pi(i)} \oplus b_{\pi(i)}) \end{aligned}$$

For each pair (b, π), for any f, we can create a (generalized) complementary equivalent function f', and then apply $\gamma^{-1}_{b,\pi}$ to obtain a function equally hard

for mutation as f' was for crossover. Each of these mappings has properties similar to γ^{-1}. Each would induce an isomorphic but different cycle pattern on the hypercube.

9 Discriminating Functions

Putting the transformations to work, we consider f'_u, the complementary equivalent transformation of the one max function, which we call OMC (One-Max Complementary). This function (for $l = 5$) is shown under the crossover operation in figure 5. For each complementary pair, only the string with lead bit 0 is shown. The contours represent regions from which all exits via crossover lead to regions of lower value. Optimization is maximization in this example, although there is an isomorphic contour system for minimization.

Comparing OMC to the 6-bit version of One-Max under crossover in figure 2 we see that OMC has many more false peaks and ridges. Assuming this is indicative for larger l, OMC should prove more difficult for crossover on average.

On the other hand, it is easily seen that f'_u is strictly linear for mutation. The maximum distance from any string to one of the two optima following steepest paths is $\lceil (l+1)/2 \rceil$. Thus, this function is in one sense *easier* than one max for mutation in the best case.

Let us transform this function using $\mathcal{I}^{-1}$ to obtain a function which has many false peaks on the mutation SSS. This transformed one max function (TOM) is TOM $= f'^{(-1)}_u = \mathcal{I}^{-1}(f'_u)$. Using the construction of the previous section,

$$\begin{aligned} \mathrm{TOM}(x) &= f_u(\gamma^{-1}(x)) \\ \mathrm{TOM}(x) &= \sum_{i=1}^{l} \left(\left(\sum_{j=1}^{i} x_j \right) \bmod 2 \right) \end{aligned}$$

TOM is as hard for mutation as OMC is for crossover, and replacing the crossover hypercube with the Hamming cube would result in a diagram isomorphic to that in figure 5. Figure 6 and figure 7 show TOM contours under crossover for 5 and 6 bit versions. There are no false peaks on 5 and one with an extended ridge for 6 bits. It would appear from this limited evidence that TOM would be easier for crossover than for mutation.

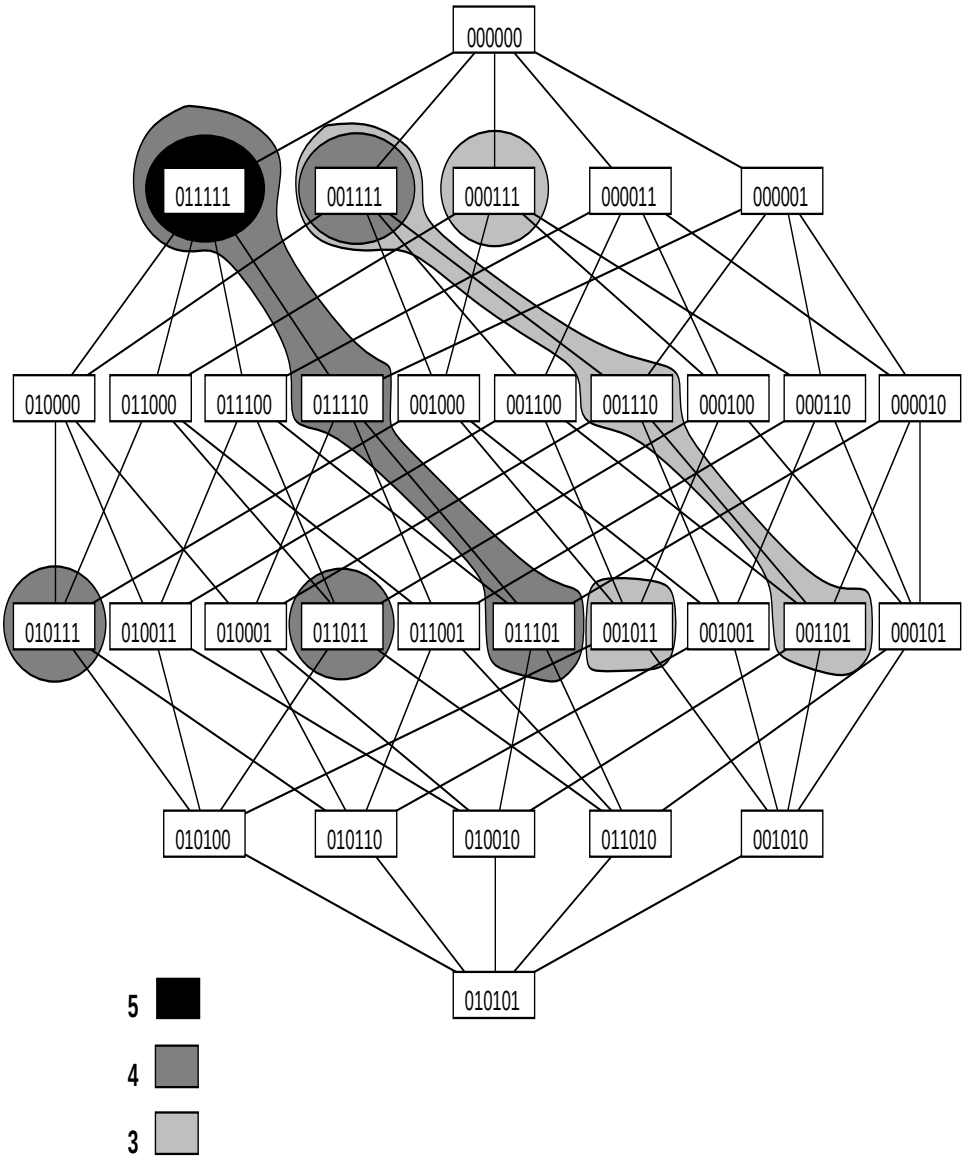

Figure 5: OMC with Contours under Crossover

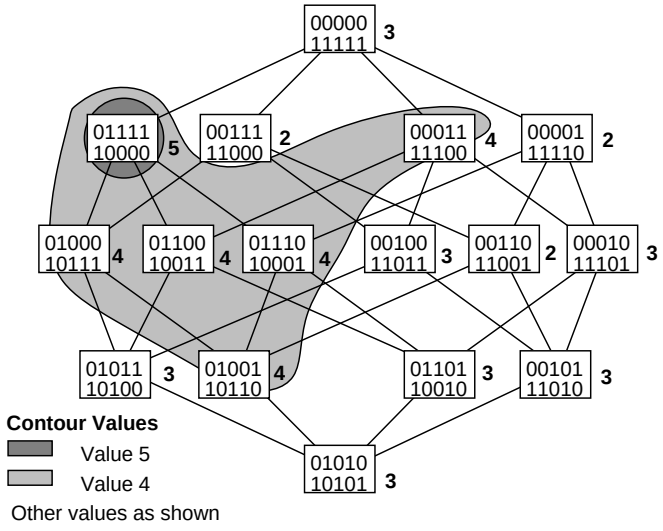

Figure 6: SSS for Crossover with Values from TOM for $l = 5$

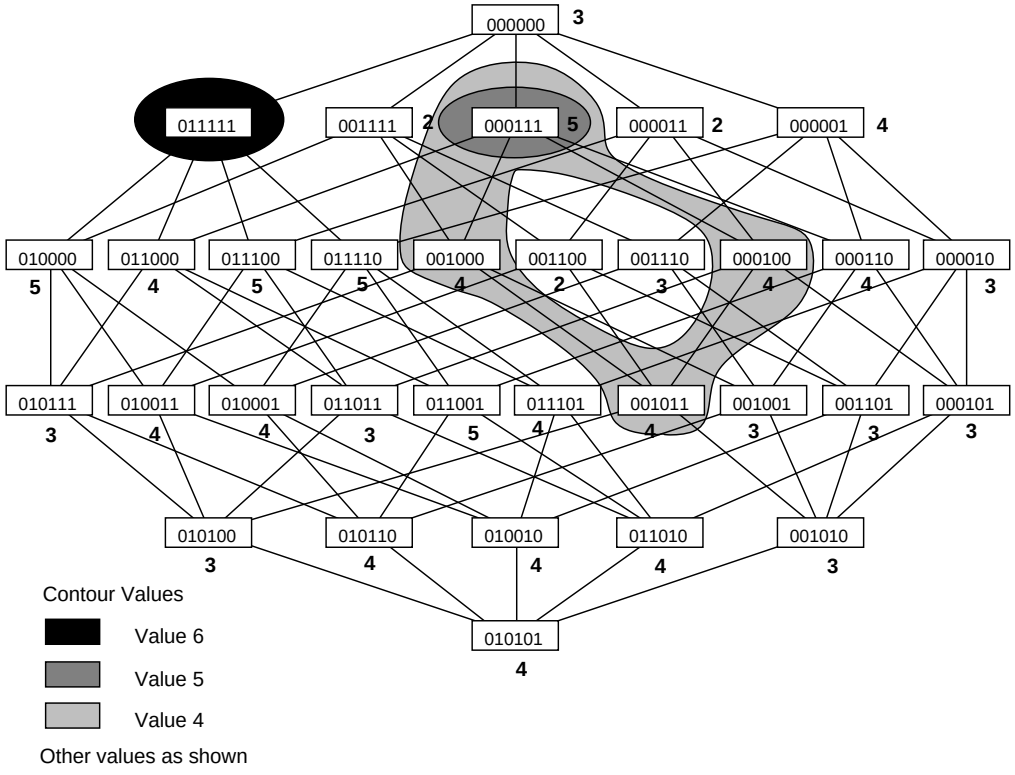

Figure 7: SSS for Crossover with Values from TOM for $l = 6$

Going in the other direction, we can create functions from One-Max and OMC that are strictly linear for crossover. The *transition function* (T) is

$$\begin{aligned} T &= \mathcal{I}(f_u) \\ T(x) &= \sum_{i=2}^{l} x_i \oplus x_{i-1} \end{aligned}$$

T is called the transition function because it counts the number of positions in which adjacent bits differ.

Lemma 9.1 *T is non-misleading for mutation under minimization or maximization, but it is not directive.*

Proof: To see that it is non-misleading under minimization consider any binary string x. Consider any adjacent pair such that $x_i \neq x_{i-1}$. We can flip either of these bits without increasing the number of transitions. Flipping enough bits eventually leads to a string of all ones or all zeroes which has minimal cost.

To show that T is not directive under minimization for mutation, consider a string with some k such that for $i < k$ the bits are 0 with the remaining bits 1. Flipping the $k-1$st or kth bits will not change the value. Any other mutation will increase the value.

Similar arguments hold for maximization. ■

We argue that an HHC is likely to have difficulty with T. In order to make progress towards a string in which all bits are equal, from one like that described in the preceding paragraph, the HHC must repeatedly mutate the next bit (in one direction or the other). This is a random walk and would require $O(l^2)$ steps on average to move the necessary $O(l)$ flips to achieve equality (see e.g. Feller, 1968) But the probability that one of the two boundary bits are flipped is $2/l$. All other mutations produce strings of worse fitness. Thus, on average $O(l^3)$ evaluations would be required if one bit-mutation is used. If more than one mutation may occur, then the probability of success is lower, and the search time longer.

The *transition complementary function*(TC) is

$$\begin{aligned} TC &= \mathcal{I}(OMC) \\ TC(x) &= \sum_{i=3}^{l} x_i \oplus x_{i-1} \oplus x_2 \oplus x_1 \end{aligned}$$

It counts the number of transitions not equal to the first. It is to crossover as OMC is to mutation, and thus is strictly linear under crossover. Using analysis similar to lemma 9.1 we find

Lemma 9.2 *TC is non-deceptive and not directive for mutation.*

The importance of these functions is as discriminators between crossover and mutation. Our claim is that a TGA will find the functions One-Max and OMC easier to solve because as discussed they are strictly linear in mutation space. TOM, T and TC should prove difficult for a TGA because they are difficult for an HHC and due to convergence of the population, the TGA will not be able to take advantage of the crossover search. A crossover based search on the other hand should have the opposite behavior.

These conclusions are supported by the experimental results in section 10.

10 Experimental Results

In this section we present experimental results for the functions described in this paper. As our canonical TGA we use the GENESIS program (Grefenstette 1991) We use GIGA (Culberson, 1992c) as the example which exploits crossover without mutation.

Our purpose is to provide evidence supporting the analysis given in the preceding sections, not to demonstrate superiority of one program over the other. For each function, we select experiments that highlight the differences between crossover and mutation search and further illuminate the mechanisms of crossover. We are not concerned with proving that a program can be made to solve a particular problem. For this reason, we vary only basic parameters, and avoid modifying most of the parameters available in GENESIS and to a lesser extent in GIGA.

For GENESIS we vary the mutation rate, crossover rate, population size and the total trials. The other parameters are the GENESIS defaults, and are listed in figure 8. Also listed is the default total trials which we used in any experiment where this value is not specified in the tables.

Mutation in GENESIS is not precisely one-point mutation, and so the results may not be as tightly tied to our analysis as one would hope. Crossover also cannot be selected to give the pure forms that our analysis assumes. However, the experiments perform very much as expected. In a sense this supports our larger contention that the TGA's performance is in a sense

bounded above by what it can do on the mutation SSS since it does not use the crossover SSS as effectively as it could.

For GIGA we vary population size, crossover types and rates, and the maximum number of matings. The default settings, and the maximum matings used when not specified in the tables, are shown in figure 9. These were described in section 5.

Total Trials = 100000
Structure Length = 100
Generation Gap = 1.000000
Scaling Window = -1
Report Interval = 1000
Structures Saved = 6
Max Gens w/o Eval = 2
Dump Interval = 6
Dumps Saved = 1
Options = cel
Random Seed = 73862564
Maximum Bias = 0.990000
Max Convergence = 100
Conv Threshold = 0.800000
DPE Time Constant = 0
Sigma Scaling = 2.000000

Figure 8: Defaults for the GENESIS Experiments

In all of our experiments, we report a "Success Ratio". The bottom number is the number of experiments of a particular type; the top is the number of experiments in which the optimal value was found. In those instances when GENESIS fails frequently, we include the average and variance for the best value seen over the experimental set, as reported by the program. Note that the optimal value for all of these functions is zero. When the programs succeed in all instances of a particular test, we include the average number of evaluations to first find the optimal value, and the standard deviation.

Table 2 shows how a TGA (GENESIS) and GIGA did on TOM using strings of 100 bits. As expected, assuming that a mutation SSS with many false peaks thwarts TGAs, GENESIS found TOM fairly difficult. The performance of the algorithm seems to be more strongly correlated with

Random Rotation
Sorted Population
Elitism
Family Size 5
Minimum Pair
Alternating Adjacent Pairs
Maximum matings = 5000
Swap Bias = 0.5

Figure 9: Defaults for GIGA

mutation rates than with crossover rates. Using high rates of both, and allowing significantly more evaluations, GENESIS was able to obtain 100% success.

The basic difficulty seems to be that with lower crossover rates, the convergence in similarity of the population makes it difficult for crossover to escape the false peaks. A higher mutation rate on the other hand aids crossover but then makes it more difficult to achieve the optimum because of the high rate of disruption. This function has false peaks in both mutation and crossover space, although the peaks are more numerous in mutation space. The last GENESIS experiment illustrates that further increases in the mutation rate may reduce efficiency. In that experiment the successes required from 407 thousand to over 933 thousand evaluations.

TOM has false peaks in the crossover SSS, and GIGA on a population of size two fails as expected. The average value obtained in this case was 34.15 with a variance of 9.71. Larger populations allow GIGA to succeed without too much difficulty. The focusing power evidently eliminates false peaks as GIGA succeeds on larger populations even using only one-point crossover. However, success comes more easily if mixed crossover types are allowed.

GENESIS was quite successful in solving OMC, as shown in table 3. With a small population and using only mutation, the optimal string was found in all cases. This is to be expected since OMC is strictly linear under mutation. For these experiments the average number of evaluations required for the first success is reported along with the standard deviation. The rapidity of success appears to be more highly correlated with mutation rate than with crossover rate. Varying the crossover rate from 0.0 to 1.0 has little effect when the mutation rate is held at 0.001. If the mutation rate is

GENESIS TOM					
Population Size	Crossover Rate	Mutation Rate	Success Ratio	Best Value Seen	
				Average	Variance
20	0.05	0.010	0/10	3.9	6.77
20	0.60	0.001	0/10	28.4	6.49
20	0.60	0.020	0/10	5.1	4.77
30	0.03	0.010	0/20	5.3	2.85
30	0.04	0.030	0/10	9.4	3.82
30	0.60	0.001	0/10	24.7	7.12
30	0.60	0.010	0/10	3.7	1.57
50	0.02	0.050	0/10	16.7	9.12
100	0.03	0.010	0/10	10.0	4.89
Allowing More Total Trials			Evaluations to Succeed		
				Average	Std
20	0.60	0.010	10/10	189818	51354
50	0.80	0.020	8/10	1000000 Total Trials	

GIGA TOM						
Population Size	Crossover Type	Rate	Swap Bias	Success Ratio	Evaluations to Succeed Average	Std
2	1-point			0/20		
30	1-point			20/20	23916	2898.2
50	1-point			20/20	30252	4216.9
25	1-point	50				
	2-point	50				
	Uniform	10	0.5	20/20	17879	2439.7
10	1-point	50				
	2-point	50				
	Uniform	10	0.5	20/20	15340	2477.2
5	1-point	50				
	2-point	50				
	Uniform	10	0.5	20/20	28610	9802.5

Table 2: Sample Runs on TOM

set too high, then too many errors are introduced when the result is nearly obtained. In this case a higher crossover rate seems to alleviate this problem somewhat. If mutation is eliminated, then GENESIS fails.

On the other hand, GIGA found this function more difficult. Using one-point crossover alone, it failed completely on a population of size 30 with a maximum of 5000 matings. (5000 matings imply approximately 50000 evaluations when family size is 5). Increasing the number of matings to 10000, and with a larger population of 100, it succeeded in only two instances out of twenty. Thus, even the power of a larger population is of limited effect. Comparing these results to the TOM results for GIGA we find support for our observations that OMC appears to have more false peaks than TOM.

GIGA has much better success when different crossovers are used. As noted previously multipoint crossover includes mutation as a special case. It seems clear that the more we mimic mutation, the better the search. This is expected since the mutation SSS is strictly linear.

Next we study the transition function. This function counts the number of positions in which the two adjacent bits differ. This function is strictly linear on the crossover SSS, in the same way that One Max is for mutation. Given the analysis in section 7 we do not expect GENESIS to do very well, even though T is non-deceptive for mutation. This is confirmed in table 4. On the other hand, GIGA does very well with a population of size two, as one would expect on a strictly linear function.

The surprise comes in the odd behavior of GIGA as the population is increased. As a general rule difficulty increases with increasing population. Even sized populations work quite well, *but* for small odd n, for example $n = 3, 5, 7$, GIGA is unable to solve the problem using one-point crossover. This phenomenon is only partially understood. For even populations, the solutions are *perfect* in the sense that in the final population every string has zero transitions. This is only possible because during initialization, every string is generated with its complement. For odd populations a perfect solution could only occur if we were lucky enough that the left over string was either all 0's or all 1's.

However, this does not explain why odd sized populations find it more difficult to get *any* optimal solutions. Even on $n = 11$ we find that in the final population 7 out of the 11 strings are often non-optimal. Perhaps for small odd populations the effects of doing local family searches are less effective because the parents of the family are less likely to be complementary. This would explain why a population of size three is so difficult, since if the first mating is not between complementary pairs, then it is unlikely that the

GENESIS OMC						
Total Trials	Population Size	Crossover Rate	Mutation Rate	Success Ratio	Evaluations to Succeed Average	Std
10000	20	0.05	0.010	9/10		
10000	10	0.00	0.010	10/10	3309.5	1050.9
10000	10	0.60	0.010	10/10	1788.0	342.0
10000	10	0.00	0.001	10/10	599.1	101.3
10000	10	0.60	0.001	10/10	535.4	152.5
10000	10	1.00	0.001	10/10	617.7	96.6

GIGA OMC						
Maximum Matings	Population Size	Crossover Type	Rate	Success Ratio	Evaluations to Succeed Average	Std
5000	30	1-point		0/20		
10000	100	1-point		2/20		
5000	25	1-point	45			
		2-point	45			
		Uniform	10	20/20	27102	1694
5000	50	1-point	50			
		2-point	50			
		Uniform	10	20/20	33138	6305
5000	5	1-point	50			
		2-point	50			
		Uniform	10	9/20		
5000	30	1-point	10			
		2-point	50			
		Uniform	30	20/20	16748	3951

Table 3: Sample Runs on OMC

next will be. On the other hand, for $n = 4$ the second mating will likely be between strings that are not complementary, and one would think the same effects should be evident.

Perhaps the effects are the result of interactions involving sorting and the order of selection or other interactions with the way the population is organized. For example, sorting is by value, and perhaps if strings of the same value were allowed to switch place (i.e. a non-stable sort) then the effects would be different. This unexplained result emphasizes the fact that there is still much to learn about large population search.

GENESIS T						
Total Trials	Population Size	Crossover Rate	Mutation Rate	Success Ratio	Best Value Seen	
					Average	Variance
50000	25	0.05	0.010	0/10	5.0	5.78
50000	25	0.60	0.001	0/10	7.0	9.11
50000	25	1.00	0.001	0/10	8.1	6.54
100000	50	0.03	0.030	0/10	5.3	2.01
1000000	30	0.80	0.010	0/10	2.7	1.12

GIGA T						
Maximum Matings	Population Size	Crossover Type	Crossover Rate	Success Ratio	Evaluations to Succeed Average	Evaluations to Succeed Std
5000	2	1-point		20/20	1103.0	203.2
5000	3	1-point		2/20		
10000	4	1-point		20/20	5405.5	1306.9
10000	5	1-point		1/20		
10000	6	1-point		20/20	11675.5	2856.8
10000	7	1-point		13/20		
5000	10	1-point		20/20	13765.0	2901.5
10000	11	1-point		20/20	14262.0	2793.0

Table 4: Sample Runs on Transition

Finally we look at the Transition Complementary (TC) function. This is the inverse transformation of OMC. As expected, GENESIS is unable to make progress. Results are displayed in table 5.

GIGA solves the problem easily with a population of size two using one-point crossover, which is to be expected since the function is strictly linear

in crossover space. Also as expected, about one half of the solutions are (complementary pairs of) strings in which all bits are equal, and the other solutions are strings of alternating bit sequences.

However, on larger populations GIGA has increased difficulty. As the population and search time are increased significantly, complete success is finally achieved again with a population of size 50. Although even larger populations might succeed given sufficient time, the experiment on population of size 100 indicates that increasing the population does not reduce the cost. The last two experiments show that it seems to make little difference whether one point or a mixture of crossovers is used. There is less indication of the odd-even behavior observed for T.

An interesting triviality is that for the first three experiments on TC whenever GIGA succeeds in finding the minimum, it also finds the maximum value within a few hundred more evaluations. The reason? To obtain the maximum it only has to crossover at the lead position of a minimal complementary pair.

Although not a proof, the results of this section strongly support the conclusions and conjectures based on the analysis in previous sections.

11 Conclusions and Future Research

It has been a standard assumption in the GA community that TGAs rely on crossover as the underpinning of their success, and that mutation is a secondary consideration necessary only to maintain diversity. The theoretical and experimental evidence in this paper indicates that this assumption is false. To the contrary it seems, as recent papers suggest, that TGAs are much better suited to exploring search spaces compatible with mutation, and are less able to exploit the full powers of crossover in either the large or the small.

There is a view held by some members of the GA community that crossover is somehow less powerful than mutation. Fogel and Atmar (1990) make the statement in the context of real parameter encodings, but the sentiment is often carried to binary encodings (Spears, 1992). The mutation referred to may be biased according to population statistics concerning the frequency of alleles in specific locations. This mutation fits the notions of crossover more than natural mutation (Spears, 1992). The previous sections of this paper clearly demonstrate that within the context of GAs using binary crossover, this view is incorrect. In the small, crossover searches a

GENESIS TC						
Total Trials	Population Size	Crossover Rate	Mutation Rate	Success Ratio	Best Value Seen	
					Average	Variance
50000	30	0.03	0.010	0/10	4.7	2.01
100000	100	0.03	0.020	0/10	6.6	1.16
100000	30	0.60	0.001	0/10	5.9	2.32
100000	30	1.00	0.001	0/10	5.7	4.01
1000000	30	1.00	0.004	0/10	3.4	1.82
1000000	30	0.80	0.010	0/10	3.1	1.43
1500000	50	0.60	0.005	0/10	2.3	0.23

GIGA TC						
Maximum Matings	Population Size	Crossover Type	Rate	Success Ratio	Evaluations to Succeed Average	Std
1000	2	1-point		20/20	1175	404.8
2500	3	1-point		7/20		
10000	4	1-point		11/20		
20000	10	1-point		0/20		
10000	25	1-point		3/20		
20000	50	1-point		20/20	109196.5	33981.1
10000	100	1-point		0/20		
10000	100	1-point	20			
		2-point	50			
		Uniform	30	1/20		

Table 5: Sample Runs on Transition Complementary (TC)

distinct but isomorphic search space structure from that searched by mutation. In the large, crossover has the capability, if properly used, to propagate information between members of the population in a way entirely foreign to unbiased mutation.

Battle and Vose (1991) show that Gray coding is a special case of the transformations of problem representations induced by matrices. These transformations can be viewed as transforming the GA operators instead of the representation. Battle and Vose generalize Holland schemata and show that these perform the same role under the GA transformations as Holland schemata do in the GA. This paper shows that Gray code (very nearly) transforms crossover to mutation and vice versa. In this sense then crossover is exploring transformed (inverse Gray coded) schemata of a closely related function. This is only approximate, because our isomorphism also requires the complementary equivalence transformation, and it only applies between SSSes induced by trivial populations. Our understanding of the role of schemata in GA optimization remains incomplete.

The isomorphisms studied in this paper were used to produce a series of functions which can be used to test algorithms to give some idea of how effectively they use the operators crossover and mutation. We have presented a Genetic Algorithm that uses crossover to separate and migrate substrings and patterns of superior fitness value, then recombine them to produce superior individuals. As theory predicts, this algorithm is more effective on crossover friendly functions than TGAs. A typical TGA on the other hand is shown to use crossover less effectively, but obtains better results from mutation friendly functions.

This paper surely raises more questions than it answers. It merely scratches the surface in the theoretical development of discriminating functions based on the MX isomorphisms. Only One-Max has been proven to have an exponential number of false peaks under crossover, while being strictly linear for mutation. OMC and TOM have not been theoretically analyzed for crossover, although examples seem to indicate they are much more difficult. Despite being strictly linear for crossover, the TC and to a lesser extent the T functions seem to thwart large population crossover search by GIGA. This bizarre behavior warrants further research to determine exactly what properties large population crossover search exploits.

Using GIGA with a population of size two indicates how a simple crossover hill-climbing algorithm works on a mutation–friendly function. This also means, because of the isomorphism used, that a one string mutation system would perform similarly on a crossover–friendly function. In fact, for any

program of one type we can create a program which behaves identically on the transformed function. Still, it may be useful to test various algorithms of either type against functions friendly to the opposite SSS because often there are nuances in implementations of which we are unaware. Consider for example the unexplained effects of odd sized populations using GIGA on T.

GIGA is an infant program of mostly theoretical interest designed to exploit the propagation and focusing power of crossover in a population to the exclusion of all else. This design is based on a largely intuitive analysis of this power. Elements close in value are mated to use the available focusing power, while genetic invariance ensures survival of material to support propagation. The usual analytic methods based on increased allocation to hyperplanes of above average fitness simply do not apply. Despite its infancy, a number of experiments in Culberson (1992a) and this paper suggest that this approach has promise. Culberson (1992a,b) also lists some possible future modifications that would be interesting, such as relaxing the constraint that only adjacent pairs be allowed to mate.

Goldberg (1989) presents an introduction to the idea of optimizing along a Pareto front for a multiple dimensional range. Given the selection pressures operating in GIGA as described in section 5, it should be interesting to use a GIGA-like approach in an environment where there are multiple evaluations of the string. Closeness of strings could be based on one or more of the evaluations and thus the diversification of the population could be along several dimensions at once.

Relaxing the invariance constraint, by allowing the population size to vary and adding small amounts of mutation for example, it should be possible to introduce some selection criteria that would lead towards convergence. In this way, the selection pressures could be used to trade-off one against another, instead of the TGA approach in which convergence pressures from selection are pitted against the randomization of mutation. Such systems might be implemented using sharing, crowding and geometric distribution of the population (see Goldberg, 1989; Mühlenbein, 1990). Looked at in this way GIGA might be seen as an extreme combination of these techniques.

Classifier systems (Goldberg, 1989; Holland, 1975) use GAs as the driving engine for creating and maintaining rule sets. One of the inherent problems with TGAs is that the natural tendency to converge to a set of highly similar strings runs counter to the need for a diverse rule set. Perhaps using multiple dimensional evaluation and taking advantage of the natural tendency of the GIGA approach to produce a diverse population, some of these

problems might be alleviated.

Acknowledgements

The author would like to thank the three anonymous referees for extensive comments and suggestions. In addition, thanks to Terry Jones for reading and editing an early draft of the paper, and for many discussions and ideas.

References

Altenberg, L. (1994). The evolution of evolvability in genetic programming. In K. E. Kinnear Jr., (Ed.), *Advances in Genetic Programming*, Chapter 3. Cambridge, MA. MIT Press.

Bäck, T. (1993). Optimal mutation rates in genetic search. In S. Forrest, (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, (pp. 2–8). Morgan Kaufmann.

Battle, D. L. & Vose, M. D. (1991). Isomorphisms of genetic algorithms. In G. J. E. Rawlins, (Ed.), *Foundations of Genetic Algorithms*, (pp. 242–251). Morgan Kaufmann.

Culberson, J. (1992a). Genetic invariance: A new paradigm for genetic algorithm design. Technical Report TR 92-02, University of Alberta Department of Computing Science. ftp ftp.cs.ualberta.ca pub/TechReports.

Culberson, J. (1992b) GIGA program and experiments. ftp ftp.cs.ualberta.ca in pub/TechReports/TR92-02/GIGA.

Culberson J. (1992c) GIGA program description and operation. Technical Report TR 92-06, University of Alberta Department of Computing Science. ftp ftp.cs.ualberta.ca pub/TechReports.

Deb, K. & Goldberg, D. E. (1992). Analyzing deception in trap functions. In L. D. Whitley, (Ed.), *Foundations of Genetic Algorithms 2*, (pp. 93–108). Morgan Kaufmann.

DeJong, K. A. (1992). Genetic algorithms are NOT function optimizers. In L. D. Whitley, (Ed.), *Foundations of Genetic Algorithms 2*, (pp. 5–17). Morgan Kaufmann.

Feller, W. (1968). *An Introduction to Probability Theory and Its Applications*, volume I. John Wiley & Sons, Inc., New York, New York.

Fogel, D. B. & Atmar, J. W. (1990). Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics*, 63, 111–114.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc.

Goldberg, D. E., Korb, B. & Deb K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3, 493–530.

Grefenstette, J. J. (1991). GENESIS 1.2ucsd. Enhanced version by N. N. Schraudolph, 1991 Version. ftp cs.indiana.edu in pub/alife/software/unix/GAucsd.

Grefenstette, J. J. (1992). Deception considered harmful. In L. D. Whitley, (Ed.), *Foundations of Genetic Algorithms 2.* (pp. 75–92). Morgan Kaufmann.

Harary, F., Hayes, J. P. & Wu, H. (1988). A survey of the theory of hypercube graphs. *Computers and Mathematics with Applications*, 15, 4, 277–289.

Holland, J. H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: The University of Michigan Press.

Horn, J., Goldberg, D. E. & Deb, K. (1994). Long path problems. (To appear in) *Parallel Problem Solving from Nature 3*, Lecture Notes in Computer Science. Springer-Verlag.

Jones, T. (1994). *Evolutionary Algorithms, Fitness Landscapes and Search* Ph.D. Thesis Proposal (unpublished).

Lewchuk, M. (1992). Genetic invariance: A new approach to genetic algorithms. Master's thesis, University of Alberta, Edmonton Alberta. Technical Report TR 92-05 "Genetic Invariance: A New Type of Genetic Algorithm" ftp ftp.cs.ualberta.ca pub/TechReports.

Louis, S. J. & Rawlins, G. J. E. (1991). Designer genetic algorithms: Genetic algorithms in structure design. In R. Belew and L. B. Booker, (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, (pp. 53–60). Morgan Kaufmann.

Mühlenbein, H. (1991). Evolution in time and space — the parallel genetic algorithm. In G. J. E. Rawlins, (Ed.), *Foundations of Genetic Algorithms*, (pp. 316–337), Morgan Kaufmann.

Mühlenbein, H. (1992). How genetic algorithms really work I. mutation and hillclimbing. In R. Männer and B. Manderick, (Eds.), *Parallel Problem Solving from Nature, 2*, (pp. 15–25), Elsevier Science Publishers.

Radcliffe, N. J. (1992). Non-linear genetic representations. In R. Männer and B. Manderick, (Eds.), *Parallel Problem Solving from Nature, 2*, (pp. 259–268). Elsevier Science Publishers.

Reingold, E. M., Nievergelt, J. & Deo, N. (1977). *Combinatorial Algorithms: Theory and Practice*. Prentice-Hall, Inc.

Schaffer, J. D., Caruana, R. A., Eshelman, L. J. & Das,R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. Schaffer, (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, (pp. 51–60). Morgan Kaufmann.

Schaffer, J. D. & Eshelman, L. J. (1991). On crossover as an evolutionarily viable strategy. In R. K. Belew and L. B. Booker, (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, (pp. 61–68). Morgan Kaufmann.

Spears, W. M. (1992). Crossover or mutation? In L. D. Whitley, (Ed.), *Foundations of Genetic Algorithms 2*, (pp. 221–237). Morgan Kaufmann.

Spears, W. M. & De Jong, K. A. (1991). An analysis of multi-point crossover. In G. J. E. Rawlins, (Ed.), *Foundations of Genetic Algorithms*, (pp. 301–315). Morgan Kaufmann.

Tackett, W. & Carmi, A. (1994). The unique implications of brood selection for genetic programming. *Proceedings of the First IEEE Conference on Evolutionary Computation*, Orlando, Florida, (pp. 160–165). IEEE Press.

Vose, M. D. (1992). Modeling simple genetic algorithms. In L. D. Whitley, (Ed.), *Foundations of Genetic Algorithms 2*, (pp. 63–73). Morgan Kaufmann.

Whitley, L. D. (1989). The GENITOR algorithm and selection pressure: Why rank–based allocation of reproductive trials is best. In J. D. Schaffer, (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, (pp. 116–121), Morgan Kaufmann.

12 Appendix I: Proof of Cycle Theorem

We here restate and prove Theorem 8.1. We use the notation $x^i = \gamma^{-i}(x)$.

THEOREM **12.1** *If $m = 2^k$ where k is such that $2^{k-1} < l \leq 2^k$ then $x^m = x$ and if $x_1 = 1$ then for $0 < m' < m$, $x^{m'} \neq x$.*

To prove Theorem 8.1 we first prove a number of lemmas. We begin with a restatement of the definition of γ^{-1}.

Lemma 12.1

$$x_j^i = x_{j-1}^i \oplus x_j^{i-1},\ i > 0, 1 < j \leq l$$
$$x_1^i = x_1,\ i \geq 0$$

Proof: From the definition of γ^{-1}. ■

Lemma 12.2

$$x_j^i = x_{j-2^k}^i \oplus x_j^{i-2^k}, j > 2^k, i \geq 2^k$$

Proof: By applying lemma 12.1 recursively, we find

$$\begin{aligned} x_j^i &= x_{j-1}^i \oplus x_j^{i-1} \\ &= x_{j-2}^i \oplus x_{j-1}^{i-1} \oplus x_{j-1}^{i-1} \oplus x_j^{i-2} \\ &= x_{j-2}^i \oplus x_j^{i-2} \\ &\vdots \\ &= x_{j-2^k}^i \oplus x_j^{i-2^k} \end{aligned}$$

■

Lemma 12.3 *For $1 \leq j \leq m = 2^k$*

$$x_j^{mi} = x_j^{m(i-1)}$$

Proof: The proof is by induction.
Basis: $x_1^i = x_1$ by lemma 12.1. Also, by two applications of lemma 12.1

$$\begin{aligned} x_2^{2i} &= x_1^{2i} \oplus x_1^{2i-1} \oplus x_2^{2i-2} \\ &= x_1 \oplus x_1 \oplus x_2^{2(i-1)} \\ &= x_2^{2(i-1)} \end{aligned}$$

Induction: Assuming the statement true for $h = k - 1$ (and $j \leq m/2$) we wish to prove it true for k. By applying lemma 12.2 twice we have

$$\begin{aligned} x_j^{mi} &= x_{j-\frac{m}{2}}^{mi} \oplus x_j^{m(i-\frac{1}{2})} \\ &= x_{j-\frac{m}{2}}^{mi} \oplus x_{j-\frac{m}{2}}^{m(i-\frac{1}{2})} \oplus x_j^{m(i-1)} \end{aligned}$$

Since $j - m/2 \leq m/2$ we can apply the inductive hypothesis to obtain

$$\begin{aligned} x_j^{mi} &= x_{j-\frac{m}{2}} \oplus x_{j-\frac{m}{2}} \oplus x_j^{m(i-1)} \\ &= x_j^{m(i-1)} \end{aligned}$$

■

Lemma 12.4 *If $m = 2^k$, $l \geq m$ then*

$$x_{m+1}^{mi} = x_1 \oplus x_{m+1}^{m(i-1)}$$

Proof: From lemma 12.2 we have

$$\begin{aligned} x_{m+1}^{mi} &= x_1^{mi} \oplus x_{m+1}^{m(i-1)} \\ &= x_1 \oplus x_{m+1}^{m(i-1)} \end{aligned}$$

■

We are now ready to prove theorem 8.1.
Proof: The equality condition follows directly by applying lemma 12.3 with $i = 1$.

The non-equality can be proved by contradiction by assuming that there is an m' such that equality is achieved. Then consider the largest i such that $m' \equiv 0 \pmod{2^i}$. Thus, $h = \frac{m'}{2^i}$ is odd. Letting $z = 2^i$, we apply lemma 12.4 iteratively to obtain

$$\begin{aligned} x_{z+1}^{m'} &= x_{z+1}^{zh} \\ &= (h \otimes x_1) \oplus x_{z+1} \\ &= x_1 \oplus x_{z+1} \\ &\neq x_{z+1} \end{aligned}$$

■