TECHNICAL REPORT NO. 326

Using Genetic Algorithms to Design Structures

by

Sushil J. Louis and Gregory J. E. Rawlins

February 1991

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
Bloomington, Indiana 47405-4101

Using Genetic Algorithms to Design Structures

Sushil J. Louis   Gregory J. E. Rawlins [*]

Abstract

This proposal considers the problem of using genetic algorithms for structure design. We relax some constraining assumptions that classical genetic algorithms make about problem representation and describe a designer genetic algorithm that makes use of differential information about search *direction* to effectively design structures. Analysis of performance and some preliminary results are presented, along with directions for further research.

1 Introduction

The problem of designing structures is pervasive in science and engineering. We can state the problem as:

> Given a function and some materials to work with, design a structure that performs this function subject to certain constraints.

Design is traditionally considered a *creative* process and therefore difficult to automate. We can think of the human design process as a black box (see figure 1). Input to this black box consists of all the knowledge that the design engineer possesses. The output is a structure that performs some useful function. Expert systems which seek to codify knowledge are currently too brittle and not applicable across a broad range of domains. However there is a process that has been spectacularly successful in producing a broad range of robust structures that are efficient at performing a broad range of functions. This is natural selection, the process of evolution. Its success is evident from the abundance and diversity of life on this planet.

Genetic algorithms are based on natural selection. They should therefore enjoy similar success in solving the problem of design. However when naively applied their performance is less than encouraging. This paper considers the problem of using genetic algorithms to design structures.

As an example consider the *combinational circuit design problem*: Given a set of logic gates to work with, design a circuit that performs a desired function. Two instantiations

[*]Department of Computer Science, Indiana University, Bloomington, IN - 47405. Email: louis@iuvax.cs.indiana.edu, rawlins@iuvax.cs.indiana

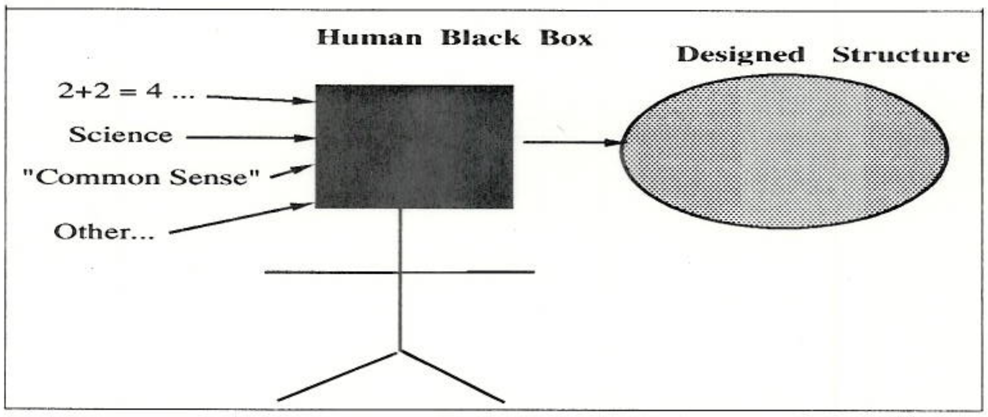


Figure 1: The human design process as a black box.

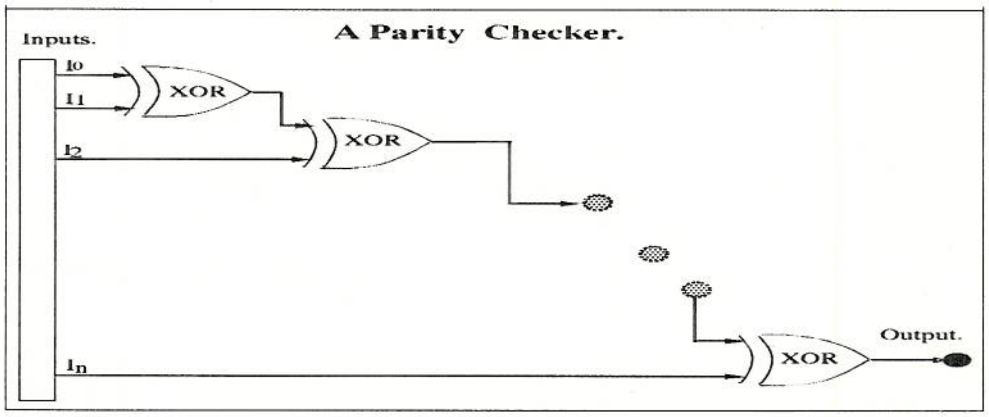


Figure 2: An "n" bit parity checker.

of this problem are the *parity problem* and the *adder problem*. A solution to these two problems is given in most introductory textbooks on digital design (see figures 2 and 3). Both problems are well-defined, unambiguous, easy to evaluate, and can be scaled in difficulty. In addition, we can change the number of solutions (the footprint) in the search space of a particular instantiation by varying the types of gates available. We therefore use them as a testbed and as a basis for performance comparison of various design strategies.

The next section presents an overview of natural selection and motivates the use of genetic algorithms on design problems. Section three defines a classical genetic algorithm and presents problems with using it for design. This leads to what we call a Designer Genetic Algorithm (DGA) described in section four. Preliminary results, presented in the fifth section, indicate the usefulness of DGAs. Finally, we cover application areas, and directions for future research.

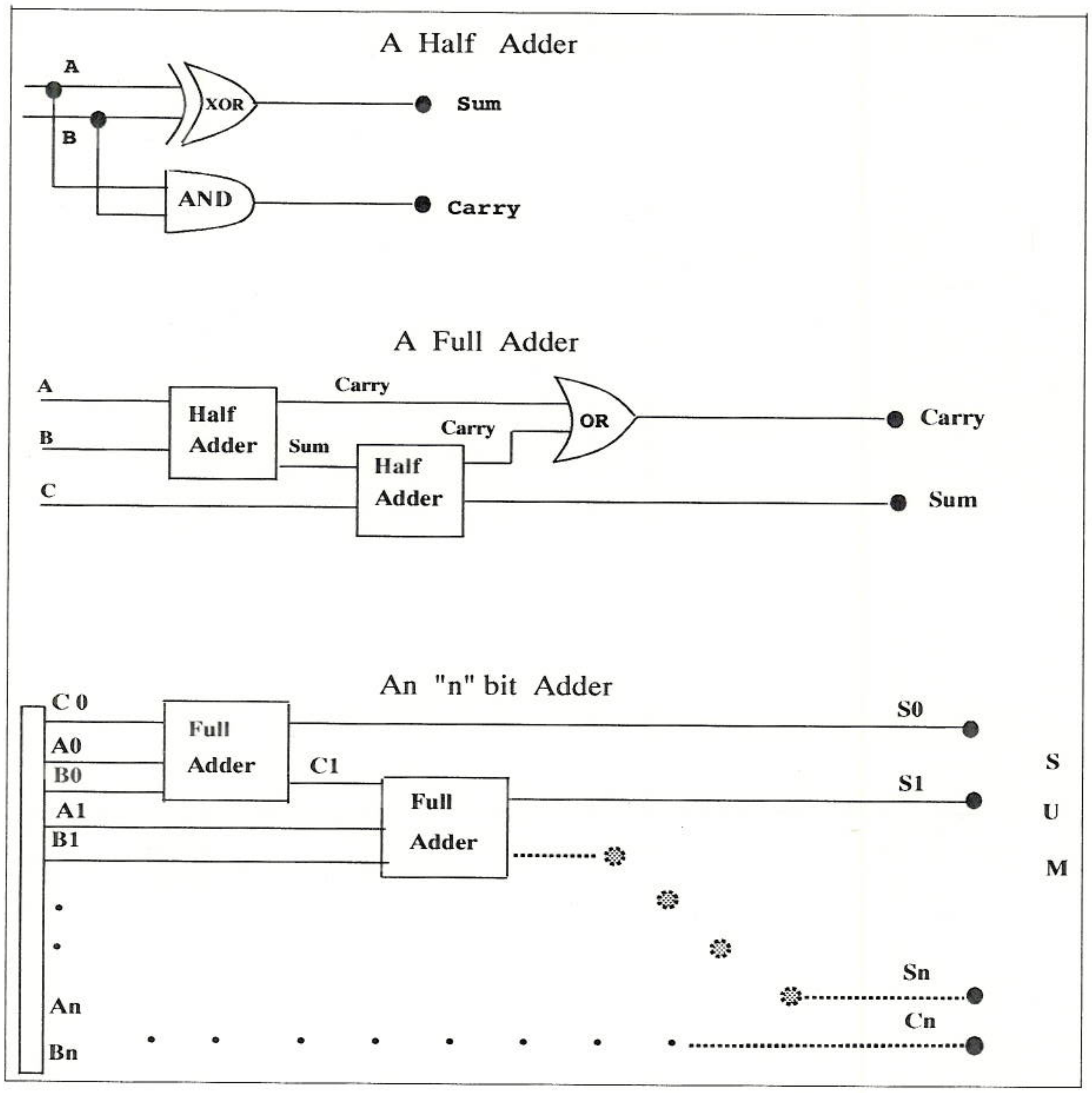


Figure 3: An "n" bit adder.

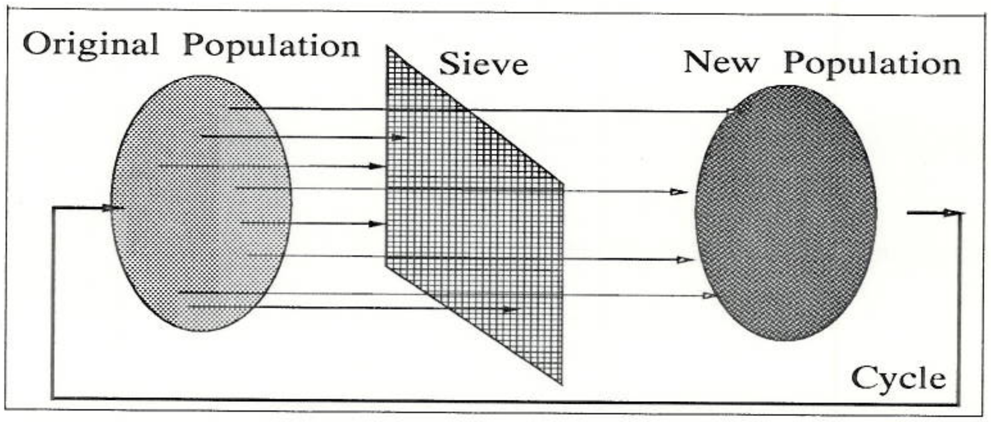


Figure 4: Natural selection as a sieve

2 Natural Selection

Most of the material in this section can be found in Richard Dawkins' book on evolution [5]. In its most general form natural selection means the differential survival of entities. Some entities live and others die. For this to happen there must be a population of entities capable of reproduction. Natural selection prunes this population according to the criterion of survivability (fitness). It acts as a sieve as shown in figure 4. In sexually reproducing species the unit of natural selection is the *gene*. The values of a gene are its *alleles*. However, sieving by itself is not capable of producing designs like those of the life forms on this planet in any reasonable amount of time. Natural selection needs reproducing entities to feed the results of one sieving process on to the next, and so on. It is the continuing *cycle* of reproduction and selection (the sieving process) that is responsible for life on earth. The existence of finite resources leads to competition for these resources and survival of those entities that have a competitive advantage. A life form, or *phenotype*, is a survival machine built by a set of genes (a *genotype*) to create a competitive advantage over other forms. Although it is the phenotypes that are directly competing for survival, it is the genotypes that get selected on the basis of this competition for further consideration. Let us consider the eye as an example of the designing power of natural selection.

An eye has evolved independently at least three different times in the course of earth's history. All three designs, exemplified by the human eye, the octopus eye and the fly's eye, are different. Consider the human eye shown in figure 5.

An eye's resemblance to a camera is striking. The iris diaphragm constantly varies the aperture while the lens focuses light using muscles. This focused image falls on the retina, where it excites photocells.

Also in the figure is an enlarged section of the retina. Light enters from the left and first hits the layer of ganglion cells before going on to the photocells.[1] The ganglion cells gather information from the photocells and do some sophisticated pre-processing before passing it

[1]The octopus' eye has the ganglion cells and photocells reversed and this may be better design.

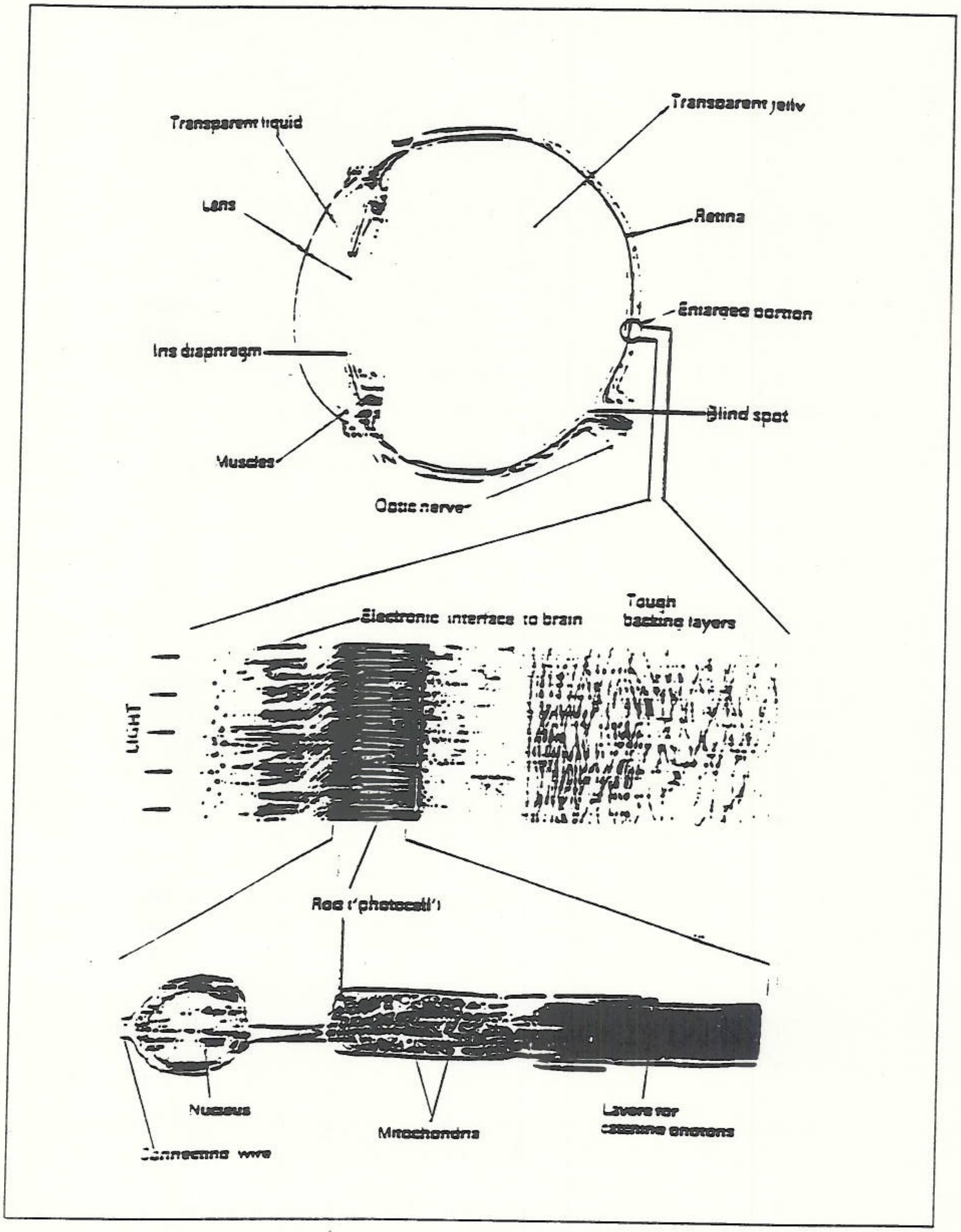


Figure 5: Schematic of a human eye. (from Dawkins' *The Blind Watchmaker* [2].)

on to the brain. Finally, a rod (a type of photocell) is shown at the bottom.

The most notable structural feature of an eye is the high degree of interdependence among the various parts. The lens is the right size to be manipulated by the muscles. Both these parts are the right distance from the retina to focus images properly. The number of photocells and ganglion cells are dependent on each other and influenced by the size of the retina. This interdependence is called *epistasis* and is an important aspect of structures.

We can think of evolution as a search process. It searches a very large space of genotypes producing structures that are efficient at carrying out functions desirable for survival in their environment. For example the search space for the ϕX174 viral genotype is of the order of 4^{5400}, and this is one of the smallest life forms! This motivated John Holland to define a *Genetic Algorithm*, a search process based on natural selection, as a tool for searching the large, poorly-understood spaces that arise in many application areas of science and engineering [19].

3 An Artificial Model of Natural Selection

A Genetic Algorithm (GA) is a randomized parallel search method modeled on evolution. GAs are being applied to a variety of problems and becoming an important tool in machine learning and function optimization. Goldberg's book gives an exhaustive list of application areas [14]. Their beauty lies in their ability to model the robustness, flexibility and graceful degradation of biological systems. However, there has been very little research on their applicability to design problems. Much of the GA literature concerns function optimization. Any references to design invariably have to do with optimization of design parameters. In such problems the structure is given and the object is to optimize some associated cost [2, 21]. For example, in a paper by Goldberg and Samtani [13] a GA minimizes the weight of a 10-member plane truss, subject to maximum and minimum stress constraints on each member. Although such design parameter optimization is important, our problem is to design the structure itself. It is interesting and not a little puzzling to note this lack of literature on the subject. It seems natural to use a genetic algorithm to design structures, since the paradigm on which they are based is so successful at it. The difficulties lie in the enormous size of the problem, the epistasis present in the structure, and the biases inherent in current GAs. Before considering the difficulties in detail, it is helpful to understand the theory behind genetic algorithms.

3.1 Classical Genetic Algorithms

A GA encodes each of a problem's parameters as a binary string. An encoded parameter can be thought of as a gene, the parameter's values, the gene's alleles. The string produced by the concatenation of all the encoded parameters forms a genotype. A randomly generated set of such strings forms the initial population from which the GA starts it search. The three basic genetic operators: selection, crossover, and mutation guide this search. Selection of a string, which represents a point in the search space (think of this as the phenotype), depends

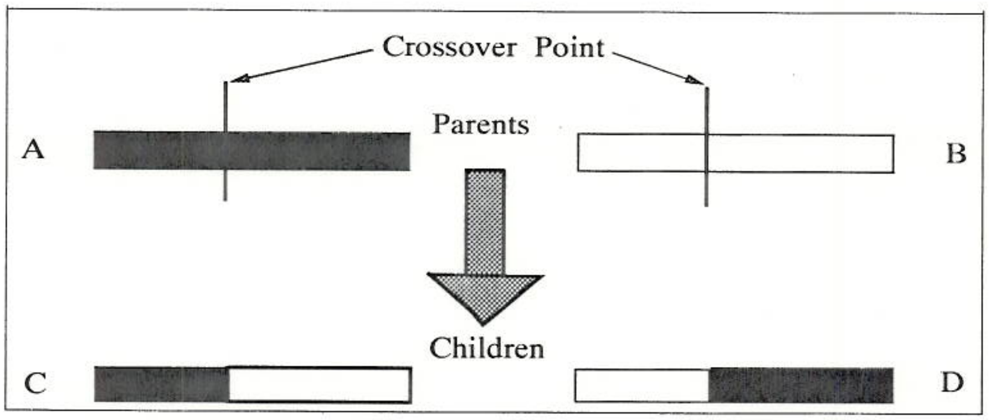


Figure 6: Crossover of the two parents A and B produces the two children C and D. Each child consists of parts from both parents which leads to information exchange.

on the string's fitness relative to that of other strings in the population. The genetic search process is iterative: evaluating, selecting, and recombining strings in the population during each iteration (generation) until reaching some termination condition. The basic algorithm, where $P(t)$ is the population of strings at generation t, is given below.

```
$t = 0$
initialize $P(t)$
evaluate $P(t)$
while (termination condition not satisfied) do
begin
        select $P(t+1)$ from $P(t)$
        recombine $P(t+1)$
        evaluate $P(t+1)$
        $t = t + 1$
end
```

Evaluation of each string is based on a fitness function that is problem dependent. This corresponds to the environmental determination of survivability in natural selection. Selection is done on the basis of relative fitness and it probabilistically culls from the population those points which have relatively low fitness. Recombination, which consists of mutation and crossover, imitates sexual reproduction. Mutation just flips a specific bit which is chosen probabilistically. Crossover is a structured yet randomized operator that allows information exchange between points. It is implemented by choosing a random point in the selected pair of strings and exchanging the substrings defined by that point. Figure 6 shows how crossover mixes information from two parent strings, producing offspring made up of parts from both parents. We note that this operator which does no table lookups or backtracking, is very efficient because of its simplicity. Crossover, where A and B are the two parents, producing two offspring C and D, is shown in figure 6 and defined below:

```
for $i$ from 1 to crossover-point
begin
    copy $i^{th}$ bit from $A$ to $C$
    copy $i^{th}$ bit from $B$ to $D$
end
for $i$ from (crossover-point + 1) to string-length
begin
    copy $i^{th}$ bit from $B$ to $C$
    copy $i^{th}$ bit from $A$ to $D$
end
```

It is easiest to understand GAs from the viewpoint of function optimization. The mapping from genotype (string) to phenotype (point in search space) is almost trivial. As crossover and mutation manipulate the strings in the population thereby exploring the space, selection probabilistically filters out strings with low fitness, exploiting the areas defined by strings with high fitness.

Holland's *schema theorem* is fundamental to the theory of genetic algorithms. A schema is a template that identifies a subset of strings with similarities at certain string positions. For example consider binary strings of length 6. The schema 1**0*1 describes the set of all strings of length 6 with 1*s* at positions 1 and 6 and a 0 at position 4. The "*" denotes a "don't care" symbol which means that positions 2, 3 and 5 can be either a 1 or a 0. Although we only consider a binary alphabet this notation can be easily extended to non-binary alphabets. The *order* of a schema is defined as the number of fixed positions in the template, while the *defining length* is the distance between the first and last specific positions. The order of 1**0*1 is 3 and its defining length is 5. The *fitness* of a schema is the average fitness of all strings matching the schema.

GAs dynamically balance the need for exploration and exploitation through the recombination and selection operators. With the operators as defined above, the schema theorem proves that relatively short, low-order, above average schema get an exponentially increasing number of trials or copies in subsequent generations. This means that schema that are of short defining length and of low order, relative to the length of the string, play a large part in biasing genetic search. Holland proved that the strategy of exponentially increasing allocation of trials is near-optimal. This leads to a statement about the way GAs work, the *building block hypothesis* which states:

> A genetic algorithm seeks near-optimal performance through the juxtaposition of *short*, low-order, high-performance schema (called building blocks).

One corollary to the schema theorem (and the building block hypothesis) is that high performance schema of *long defining length*, but of low order do not play a significant role in biasing genetic search. This constraint has important consequences in applying genetic algorithms to the problem of structure design.

3.2 Applying Genetic Algorithms to Structure Design

Using a genetic algorithm for designing a structure is like playing with a mechano set (a child's construction kit). Given some low level building blocks, the task is to put them together so that they perform a certain function. Using this analogy, a GA used for design can be considered a manipulator. It manipulates low-level "tools" or building blocks, playing with their arrangements, until it finds the required structure. Encoding therefore consists of finding a set of low-level tools for the GA to manipulate. With this as a basis, consider the problems that can arise.

First, a necessary condition for a GA to build a structure is that there should be at least one and preferably many evolutionary paths leading to the desired structure. This means that a GA (or any search method) may perform poorly in designing highly specific structures. In other words, any search method will perform poorly in optimizing a function that is zero at all points but one [4].

Next, because of the encoding, structures may not be exploitably related to those surrounding them in structure space. GAs search for and exploit any similarities in the encoding. An encoding that does not reflect relationships in the problem space can cause the GA to flounder, as it does not follow "the principle of meaningful building blocks," – one of two principles to follow in encoding a problem for a GA[14]. The mapping from genotype to phenotype is now much more complex. We can compare the structure of an eye (a structure phenotype) with a point in the search space (a phenotype in function optimization) to get an idea of this complexity. Epistasis in phenotypic structures therefore plays an important part in determining the suitability of classical genetic algorithms to structure design. Interdependence in phenotypic space may not be reflected in the genotype (unless it is very carefully encoded) and may cause a GA to be mislead.

Finally, since we are working with structures, we often work in more than one dimension. Physical structures exist in three dimensions and may often be made up of many kinds of lower level building blocks. Higher dimensionality and a large alphabet increase the search space tremendously.

Representation or genotypic encoding is a key issue. The biases generated by the encoding play a major role in determining success.

3.3 Bias in Traditional Crossover

The bias toward short schema is both a feature and a bug of the traditional crossover operator. To see why, consider the probability of disruption of a schema. Let H be a schema, $\delta(H)$ its defining length and $O(H)$ its order. Then the probability that the crossover point falls within the schema is $\delta(H)/(l-1)$ where l is the length of the string containing the schema. This is exactly the probability that crossover will disrupt the schema. Since the probability of disruption is proportional to the defining length, schema of long defining length tend to be disrupted more often than their shorter counterparts. However, in epistatic domains, schema of arbitrary defining length need to be preserved. Therefore the bias towards short schema becomes a bug, and not a feature of crossover. This means that if an encoding does not ensure that low order building blocks have short defining length the GA will find it difficult to make progress. On the other hand, in problems with low epistasis, a GA will perform well.

One way out of this is to use inversion. Inversion rearranges the bits in a string allowing linked bits to move close together.[2] To implement inversion, the encoding is changed to carry along a tag which identifies the position of a bit in the string. With the tags specifying position, it is now possible to cut and splice parts of a string allowing bits to migrate and come together. Inversion-like reordering operators have been implemented by Goldberg and others [12, 26] with some good results.

The problem with using inversion and inversion-like operators is the decrease in computational feasibility. If l is the length of a string, inversion increases the search space from 2^l to $2^l!$. Natural selection has geological time scales to work with and therefore inversion is sufficient to generate tight linkage. We do not have this amount of time or the resources available to nature. To combat the problem of disruption in highly epistatic design problems we would like to remove the bias toward short schema and allow low order schema of arbitrary defining length to bias search in useful directions.

Another approach is to use a new crossover operator like punctuated crossover or uniform crossover. Punctuated crossover relies on a binary mask, carried along as part of the genotype, in which a 1 identifies a crossover point. Masks being part of the genotypic string, change through crossover and mutation. Experimental results with punctuated crossover did not conclusively prove the usefulness of this operator or whether these masks adapt to an encoding [22, 23].

Uniform crossover exchanges corresponding bits with a probability of 0.5. The probability of disruption of a schema is now proportional to the order of the schema and independent of defining length. Experimental results with uniform crossover suggest that this property may be useful in some problems [25]. However, in design problems we would like *not* to disrupt highly fit schema whatever the defining length of such a schema.

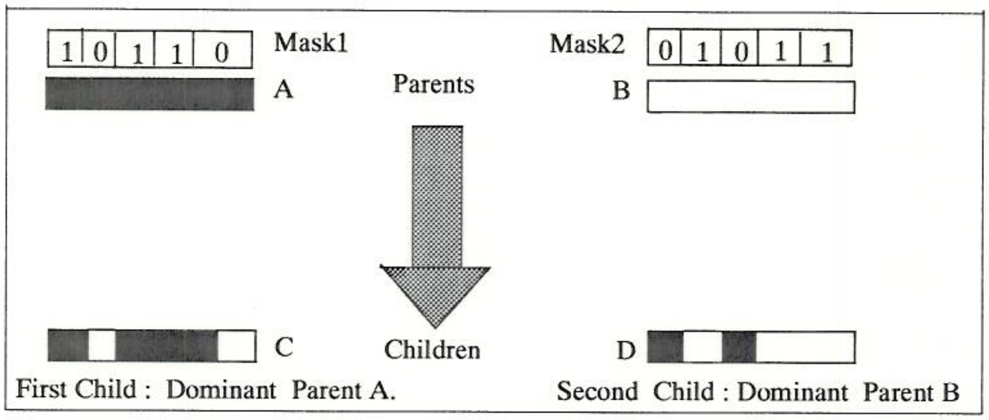
A second point needs to be made. Natural selection works by biasing search in any *direction* that shows the slightest improvement in survivability. This directional information is implicit in the number of competing alleles that exist in a population, and in nature, cannot be stored anywhere. Classical genetic algorithms and their operators, mimicking natural selection are also bound by these constraints.

To solve these problems, we use a masked crossover operator to remove the bias toward short schema and make use of explicit directional information to efficiently bias search.

4 Masked Crossover

We define an operator that directly makes use of the relative fitness of the children, with respect to their parents, to guide crossover. The *relative* fitness of the children indicates the

[2] Inversion occurs in nature and serves a similar function. Genes and their alleles are linked if their expression is dependent on one another. Tight linkage is established when linked alleles are close together. Such alleles are called co-adapted alleles.



1 0 1 1 0
Mask1
Mask2
0 1 0 1 1
A
Parents
B
C
Children
D
First Child : Dominant Parent A.
Second Child : Dominant Parent B

Figure 7: Masked crossover. The bits that are exchanged depend on the masks. This allows preservation of schema of arbitrary defining length

desirability of proceeding in a particular search direction. The use of this information is not limited to our operator, and can be used in classical GAs with very minor modifications. For example: one way to improve traditional crossover operators is to keep a sorted list of previous crossover points that produced highly fit children.

Masked crossover (MX) uses binary masks to direct crossover. Let A and B be the two parent strings, and let C and D be the two children produced. $Mask1$ and $Mask2$ are a binary mask pair, where $Mask1$ is associated with A and $Mask2$ with B. A subscript indicates a bit position in a string. Masked crossover is shown in figure 7 and defined below:

```
copy $A$ to $C$ and $B$ to $D$
for $i$ from 1 to string-length
begin
    if $Mask2_i = 1$ and $Mask1_i = 0$
        copy the $i^{th}$ bit from $B$ to $C$
    if $Mask1_i = 1$ and $Mask2_i = 0$
        copy the $i^{th}$ bit from $A$ to $D$
end
```

Masked crossover tries to preserve schema identified by the masks. Let A be called the dominant parent with respect to C, and B the dominant parent with respect to D. This follows from the definition of masked crossover since during production of C, when corresponding bits of A and B are the same, the bit from A is copied to C. The traditional way of analyzing a crossover operator is in terms of disruption. The probability of disruption P_d, of a schema H due to masked crossover is dependent on the masks. Assuming a random initialization of masks this probability is given by the number of ways that the bit positions in both parent masks corresponding to H can be combined to disrupt H in the following generation. The total number of ways of combining the mask bits corresponding to H is:

$$T_c = 2^{2\times O(H)}$$

The number of ways of disrupting H is T_c minus the number of ways of preserving H, $\mathcal{P}_H$. For each bit position in H, there are three ways of preserving it, therefore:

$$\mathcal{P}_H = 3^{O(H)}$$

So the probability of disruption is:

$$\begin{aligned} P_d &= \frac{T_c - \mathcal{P}_H}{T_c} \\ &= \frac{2^{2\times O(H)} - 3^{O(H)}}{2^{2\times O(H)}} \\ &= 1 - (\frac{3}{4})^{O(H)} \end{aligned}$$

This probability of disruption is only for randomly initialized masks and does *not* depend on $\delta(H)$.

4.1 Masks

Intuitively, 1's in the mask signify bits participating in schema. MX preserves A's schema in C while adding some schema from B at those positions that A has not fixed. A similar process produces D. Search biasing is done by changing masks in succeeding generations. Instead of using genetic operators on masks, we use a set of rules that operate bitwise on parent masks to control future mask settings. Since crossover is controlled by masks, using meta-masks to control mask string crossover then leads to meta-meta masks and so on. To avoid this problem we use rules for mask propagation. Choosing the rule to be used is dependent on the fitness of the child relative to that of its parents.

We define three types of children:

The Good Child: has fitness higher than that of both parents.

The Average Child: has fitness between that of the parents.

The Bad Child: has fitness lower than that of both parents, or equal to one or both parents.

With two children produced by each crossover, and three types of children there are a total of 3^2 or nine possibilities, with associated interpretations and possible actions on the masks. However, since the order of choosing children does not matter, the number of cases falls to six (see figure 8).

Case	Rule
Both good	MF_{gg}
Both bad	MF_{bb}
Both average	MF_{aa}
One good, one bad	MF_{gb}
One good, one average	MF_{ga}
One average, one bad	MF_{ab}

Figure 8: Six ways of pairing children and associated mask rules.

4.2 Rules for Mask Propagation

In this section we specify rules for mask propagation. In each case a child's mask is a copy of the dominant parent's except for the changes the rules allow. The underlying premise guiding the rules is that when a child is less fit than its dominant parent, the recessive parent contributed bits deleterious to its fitness. We encourage search in the area defined by these loci. The idea is to search in areas close to one parent with information from the other parent providing some guidance.[3] A mask mutation operator that flips a mask bit with low probability is assumed to act during mask propagation. We provide three representative mask functions rather than all, to give an intuitive understanding of their form. These are MF_{gg}, used when both children are good, MF_{bb} which is used when both children are bad, and MF_{ab}, used when one child is average and the other is bad.

Let $P1$ and $P2$ be the two parents, $PM1$ and $PM2$ their respective masks. Similarly, $C1$ and $C2$ are the two children with masks $CM1$ and $CM2$. The modifications to masks depend on the relative ordering of $P1$, $P2$, $C1$ and $C2$. In this section's figures, the "#" represents positions decided by tossing a coin.

1. MF_{gg}:

   Case: Both children are good.

   Summary: Very encouraging behavior and as such is reflected in the mask settings below and in figure 9. The parents' masks are OR'd to produce the children's masks, ensuring preservation of the contributions from both parents.

   Action:

   - $CM1$: OR the masks of $PM1$ and $PM2$. If there are any 0's left in $CM1$, toss a coin to decide their value.
   - $CM2$: Same as for $CM1$.
   - $PM1$: No changes except for those produced by mutation.
   - $PM2$: Same as for $PM1$.

2. MF_{bb}:

[3]Note that in MX, this is done without regard to defining length.

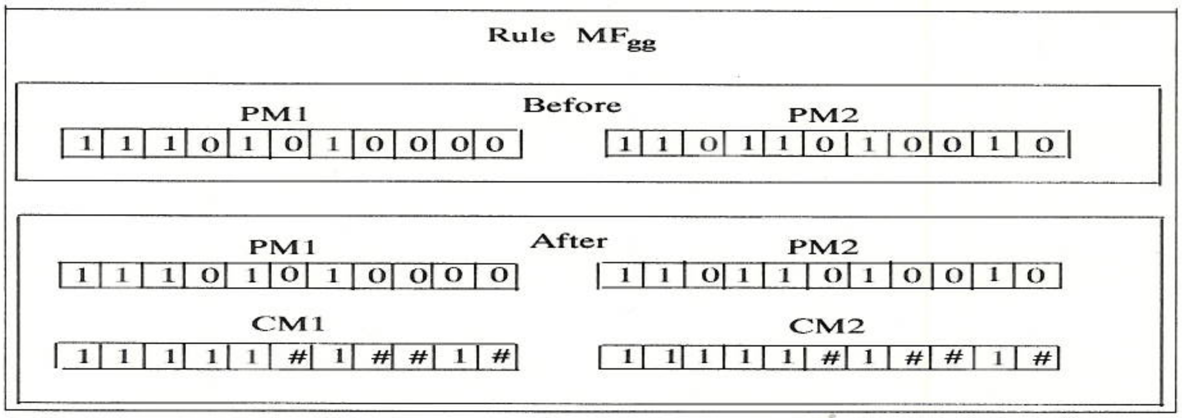


Figure 9: Mask rule MF_{gg}: Example of mask propagation when both $C1$ and $C2$ are good

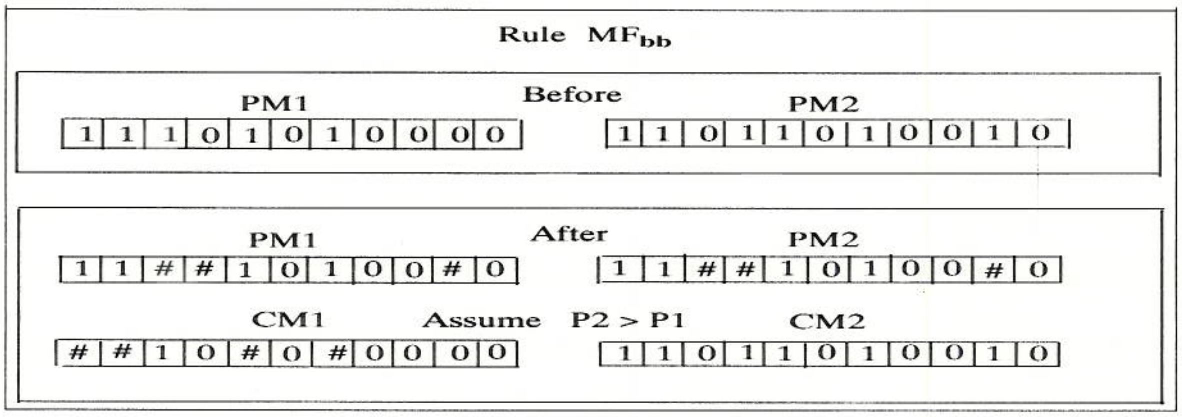


Figure 10: Mask rule MF_{bb}: Example of mask propagation when both $C1$ and $C2$ are bad.

Case: Both children are bad.

Summary: Discouraging behavior and must be guarded against in future. Each parent contributed bits that were detrimental to the children's fitness. MX has not set up the parents masks correctly. Changes are given below and shown in figure 10.

Action:

- $CM1$: This mask should reflect the undesirability of the current search direction. Contributions from $P2$ were detrimental, therefore $CM1$ should search in the area of $P2$'s contribution which is specified by the loci where $PM1_i$ is 0 and $PM2_i$ is 1. Set these loci in $CM1_i$ to 0. If $P2 > P1$ in fitness, toss a coin to set the bits of $CM1$ at those locations where both $PM1_i$ and $PM2_i$ are 1.
- $CM2$: A similar rule applies to $CM2$.
- $PM1$: $P2$'s contribution to $C1$ led to a bad child. The $C1$ positions copied from $P2$ need to be explored in $P1$. These loci are those for which $PM1_i$ was

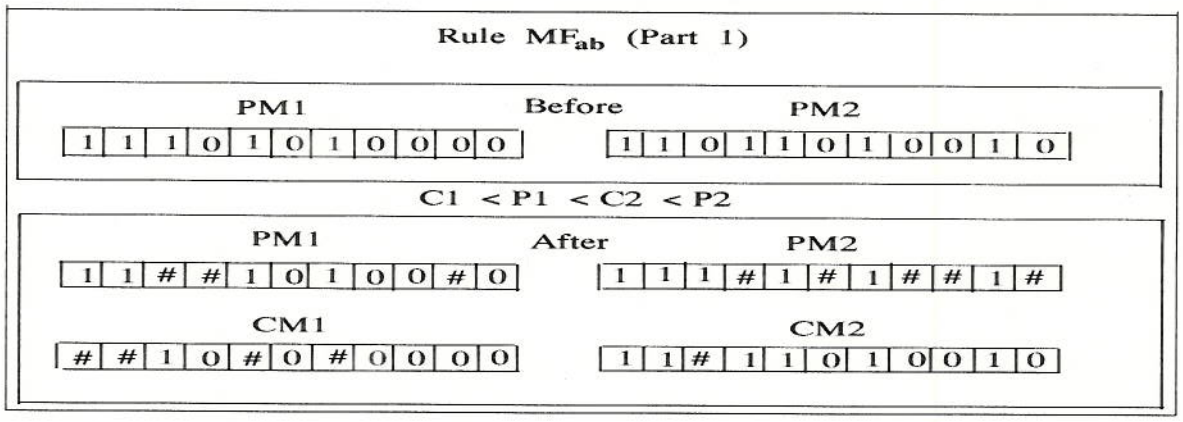


Figure 11: Mask rule MF_{ab}, Part 1: Example of mask propagation when $C1$ is bad and $C2$ is average. ($C1 < P1 < C2 < P2$)

0 and $PM2_i$ was 1. Therefore set these loci in $PM1_i$ by tossing a coin. In addition, $PM1$ specified loci that were detrimental to $C2$. Therefore when $PM1_i$ is 1 and $PM2_i$ is 0, set these locations by tossing a coin.

- $PM2$: A similar rule applies for $PM2$.

3. MF_{ab}:

Case: One child is bad while the other is average. Assume that $C1$ is bad and $C2$ is average.

Summary: There are two sub-cases. If $C2$'s fitness is less than that of $P2$ we can conclude that $P1$'s contribution was deleterious, and that $PM2$ needs to be augmented with 1's at those positions that were copied from $P1$ to $C2$ (see figure 11). If $C2$'s fitness is less than that of $P1$, $P1$'s contribution increased $C2$'s fitness, so preserve $P1$'s contribution in $C2$. The masks are shown in figure 12.

Action:

- When **C1 < P1 < C2 < P2:**
  * $CM1$: Very similar to MF_{bb}'s $CM1$. This mask should reflect the undesirability of the current search direction. Contributions from $P2$ were detrimental, therefore $CM1$ should search in the area of $P2$'s contribution which is specified by the loci where $PM1_i$ is 0 and $PM2_i$ is 1. Set these loci in $CM1_i$ to 0. Since $P2 > P1$ in fitness, toss a coin to set the bits of $CM1$ at those locations where both $PM1_i$ and $PM2_i$ are 1.
  * $CM2$: As $P1$'s contribution decreased $C2$'s fitness, set $CM2_i$ by tossing a coin when $PM1_i$ is 1 and $PM2_i$ is 0, searching around $P1$'s contribution.
  * $PM1$: $P2$'s contribution to $C1$ led to a bad child. The $C1$ positions copied from $P2$ need to be explored in $P1$. These loci are those for which $PM1_i$ is 0 and $PM2_i$ is 1. Therefore set these loci in $PM1_i$ by tossing

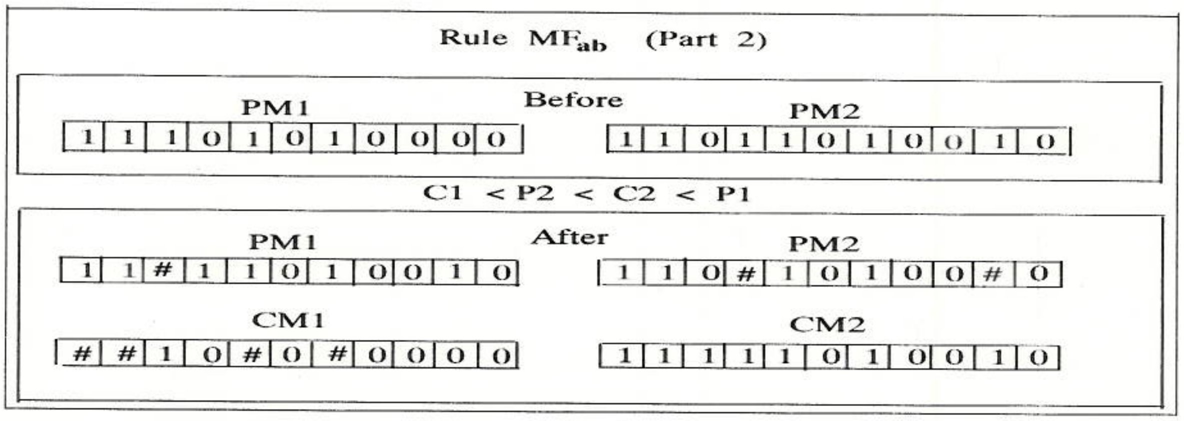


Figure 12: Mask rule MF_{ab}, Part 2: Example of mask propagation when C1 is bad and C2 is average. ($C1 < P2 < C2 < P1$)

a coin. In addition, $PM1$ specified loci that were detrimental to $C2$. Therefore when $PM1_i$ is 1 and $PM2_i$ is 0, again set these locations by tossing a coin.

  * $PM2$: Since $P1$'s contribution led to a decrease in fitness, set $PM2_i$ to 1 for those loci for which $PM1_i$ is 1 and $PM2_i$ is 0. To help fix positions currently 0, toss a coin to fix those loci in $PM2_i$ for which both $PM1_i$ and $PM2_i$ are 0.

- When **C1 < P2 < C2 < P1:**
  * $CM1$: Same as the action for $CM1$ when $C1 < P1 < C2 < P2$. Except that as $P1 > P2$, we do not take any action when both parent masks have a 1 at some position.
  * $CM2$: To preserve $P1$'s contribution, when $PM1_i$ is 1 and $PM2_i$ is 0 set $CM2_i$ to 1.
  * $PM1$: Those bits that contributed to $C1$ but not those that helped $C2$ need to be modified. Therefore for those loci where $PM1_i$ is 1 and $PM2_i$ is 0, toss a coin to decide $PM1_i$. $P2$'s contribution was detrimental so when $PM2_i$ is 1 and $PM1_i$ is 0 set $PM1_i$ to 1.
  * $PM2$: Set $PM2_i$ by tossing a coin for those loci that contributed to $C1$.

These are examples from one of several different sets of rules possible. Many mask propagation rules can be defined. In fact a GA can search the space of mask rules to find a suitable set. This may be overkill, since the number of rules is usually quite small, simpler methods will suffice. Results, outlined in the next section, indicate that a significant performance increase is obtained from even the simple set of rules above.

With mask propagation through mask rules, directional information is explicitly stored in the masks and used by the crossover operator to bias search. The main features of our masked crossover operator are then, storage and use of directional information, and independence from defining length of schema. We think of masked crossover as a golden mean between the disruptiveness of uniform crossover and the bias toward short schema of classical crossover. Compared to the size of the search space when using inversion, $2^l!$, a genetic algorithm using MX searches only 2^{2l}.

Masked crossover presents a problem when using classical selection procedures. The classical strategy of allowing the children produced to replace the original population will not allow a genetic algorithm using masked crossover to converge. Masks will tend to disrupt the best individuals while searching for promising directions to explore because of the nature of the rules guiding mask propagation. Therefore our selection procedure is a modification of the CHC selection strategy. If the population size is N, the children produced double the population to $2N$. From this, the N best individuals are chosen for further consideration [10]. We use this *elitist* selection strategy to guarantee convergence. Another problem which may occur is that although MX preserves schema of arbitrary defining length, the fitness information itself may be misleading. Such problems are called *deceptive*. When fitness information is misleading we expect a GA using masked crossover to perform worse than a GA using crossover operators that do not use such information. This is borne out by results from the adder problem.

A Designer Genetic Algorithm (DGA) therefore differs from a Classical Genetic Algorithm (CGA) in the crossover operator (masked crossover) and in the selection strategy (elitist) used.

Identifying and overcoming deception, is an important area of research, not only for structure design but also for the field of genetic algorithms. Theoretically, deception is identifiable by mathematical analysis. However, from a practical standpoint, this analysis is prohibitively expensive. Messy genetic algorithms (MGAs), developed by Goldberg to handle deception, need to identify deceptive schemas to be applicable [17, 18]. We suggest an approach satisfying both criteria, using designer genetic algorithms.

Deception can be statically identified using the ANODE algorithm suggested by Goldberg [15, 16]. Recent results indicate that the Nonuniform Walsh-Schema Transform (NWST) [3] can dynamically analyze a GA. Using the NWST in concert with the normal operation of a GA, we can collect runtime statistics needed to identify deception. Furthermore, we can improve efficiency by removing some of the determinism in the ANODE algorithm. This will not significantly alter effectiveness as long as the probability of correctly identifying deception is greater than that of incorrectly identifying it. In other words, we propose to let a DGA collect runtime statistics on encoding (through the NWST) and use these statistics to set masks. Whenever the DGA detects deception either through a periodic check of these statistics and/or a decrease in rate of progress, the algorithm identifies *deceptive schema* with the help of the statistics collected and the masks. It then allows an MGA to work on just these schema and solve the deception at this level. The DGA then continues, appropriately seeded with the optimal schema produced by the MGA. Our current research follows this approach and focuses on combining DGAs with MGAs.

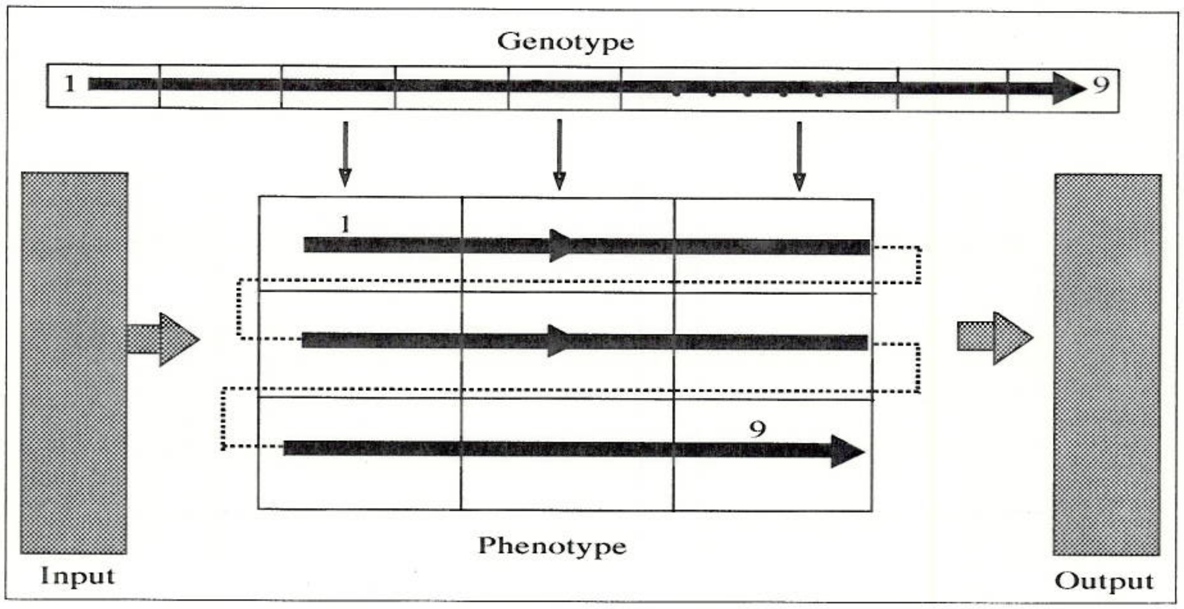


Figure 13: A mapping from a one-dimensional genotype to position in a two-dimensional phenotypic structure.

5 Results

A designer genetic algorithm's performance is compared with that of a classical GA on the adder and parity problems. In all experiments, the population is made up of 30 genotypes. The probability of crossover is 0.7 and the probability of mutation is 0.01. These numbers were found to be optimal through a series of experiments using various population sizes and probabilities. The graphs in this section plot maximum and average fitnesses over ten runs.

Each genotype is a bit string that maps to a two-dimensional structure (phenotype) embodying a circuit as shown in figures 13 and 14. We need 3 bits to represent 8 possible gates. A gate has two inputs and one output. If we consider the phenotype as a two dimensional array of gates S, a gate S_{ij}, gets its first input from $S_{i,j-1}$ and its second from one of $S_{i+1,j-1}$ or $S_{i-1,j-1}$ as shown in figure 14. An additional bit associated with each gate encodes this choice. If the gate is in the first or last rows, the row number for the second input is calculated modulo the number of rows. The gates in the first column, $S_{i,0}$ receive the input to the circuit. Connecting wires are simply gates that transfer their first input to their output. The other gates are AND, OR (inclusive OR), NOT and XOR (exclusive OR).

We determine the fitness of a genotype by evaluating the associated phenotypic structure that specifies a circuit. If the number of bits is n, the circuit is tested on the 2^n possible combinations of n bits. The fitness function returns the sum of the correct responses. This sum is maximized by the algorithm. For the 4-bit parity checker, the binary numbers 0 to 15 are inputs to the decoded circuit. If the circuit is correct, its fitness is 16, which means that the circuit correctly finds the parity of all the 16 possible 4-bit numbers. For the adder problem the input is a set of 2, n-bit numbers. The output is an $n+1$ bit sum. For a 2-bit

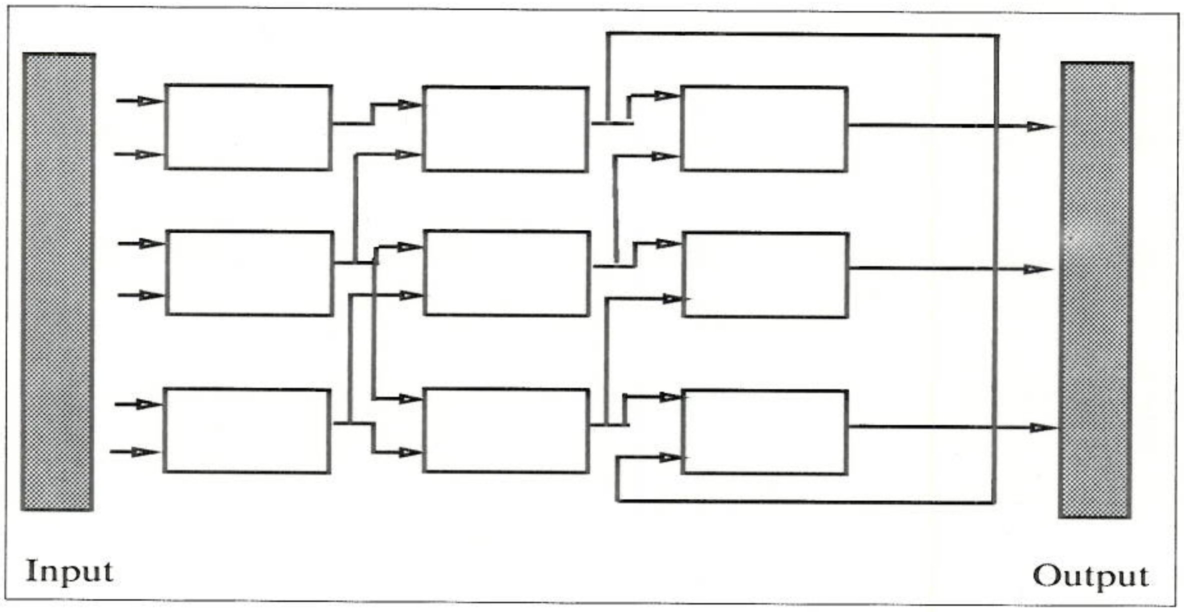


Figure 14: A gate in a two-dimensional template, gets its second input from either one of two gates in the previous column.

adder, the maximum fitness would be $3 * 2^4$ or 48.

We compare the performance of classical GA using elitist selection with a DGA on a 2-bit adder problem, the graphs in figures 15 and 16 show that the classical GA does better, although the difference is not great. This is not very encouraging. However, if we look at the solution space we see that solutions to the adder problem involve deception. A problem is deceptive to a GA, when highly fit low-order schema, lead away from highly fit schema of higher order. As explained earlier, since MX uses fitness information to bias search, it is more easily mislead than traditional crossover. Although a problem is deceptive, it does not mean that no solutions can be found. Figures 17 and 18 show solutions to the 2-bit adder problem found be a designer genetic algorithm and classical genetic algorithm. As wire gates ignore their second input, only one input is shown for such gates. The gate at position S_{33} is shown unconnected because it does not affect the output.

We now consider the parity problem. The encoding described in figure 13 will violate the principle of meaningful building blocks with regard to the solution to the parity problem as shown in figure 2. Since diagonal elements of S (the phenotype) are further apart in the string, any good subsolutions (highly fit, low order schema) found will tend to be disrupted by traditional crossover. MX however, will find and preserve these subsolutions as its performance is independent of defining length. To observe performance under these conditions, we restrict the number of gate types available to the GA to three and do not allow a choice of input (the second input is now always from the next row, modulo the number of rows). Although this reduces the size of the search space, traditional crossover disrupts low-order schema and therefore performs worse than the DGA. Figure 19 and figure 20 show this for a 4-bit parity checker. (In cases where there were no restrictions the performance of both

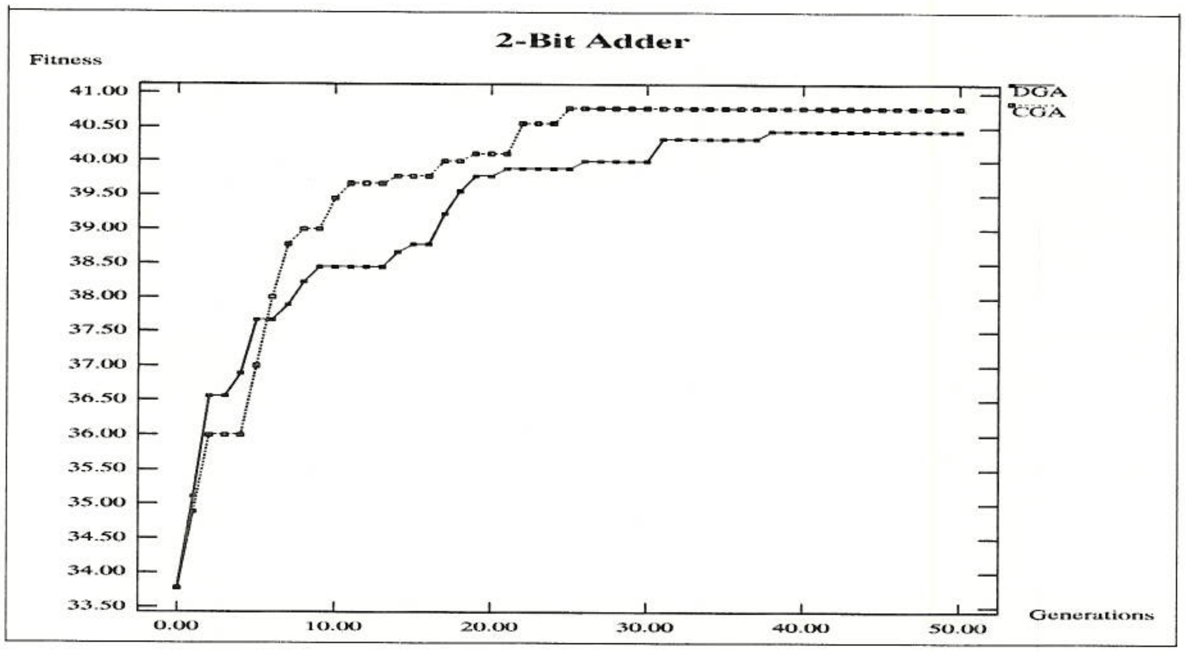


Figure 15: Performance comparison of maximum fitness per generation of a classical GA versus a DGA on a 2-bit adder.

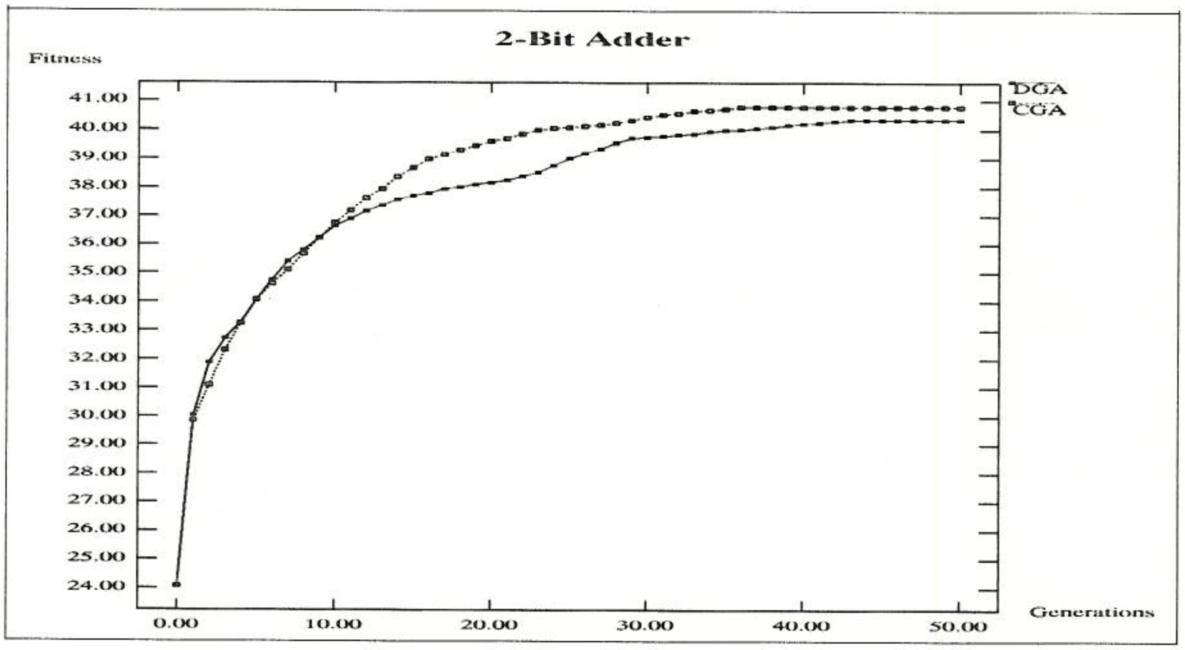


Figure 16: Performance comparison of average fitness per generation of a classical GA versus a DGA on a 2-bit adder.

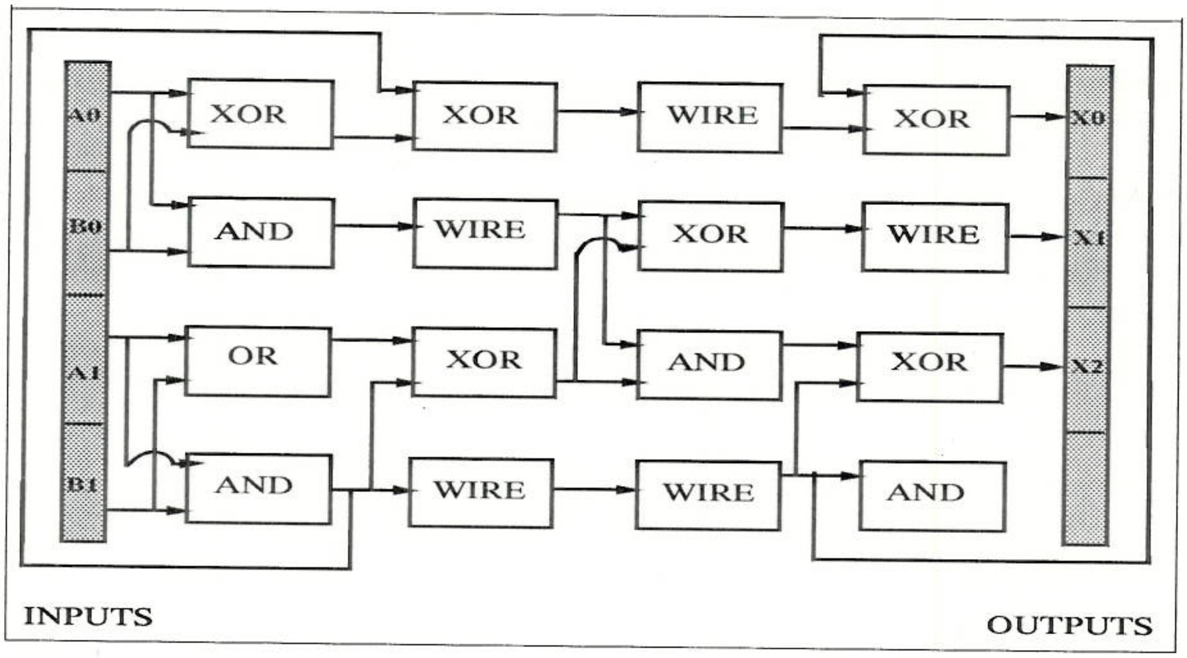


Figure 17: A 2-bit adder designed by a designer genetic algorithm.

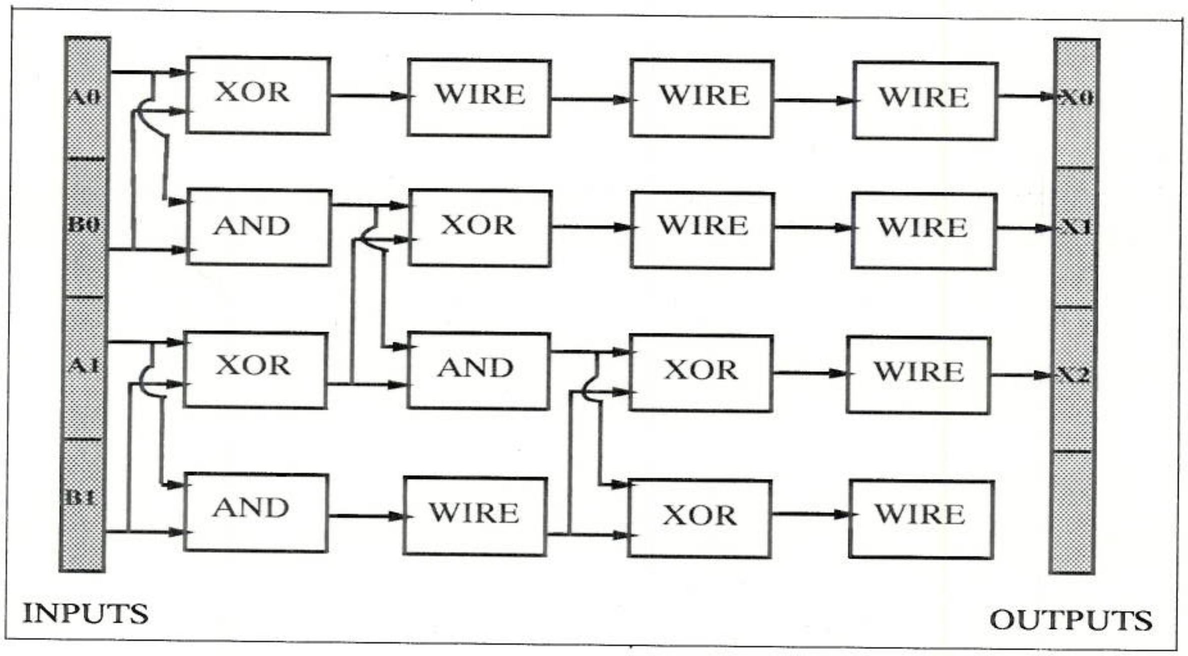


Figure 18: A 2-bit adder designed by a classical genetic algorithm.

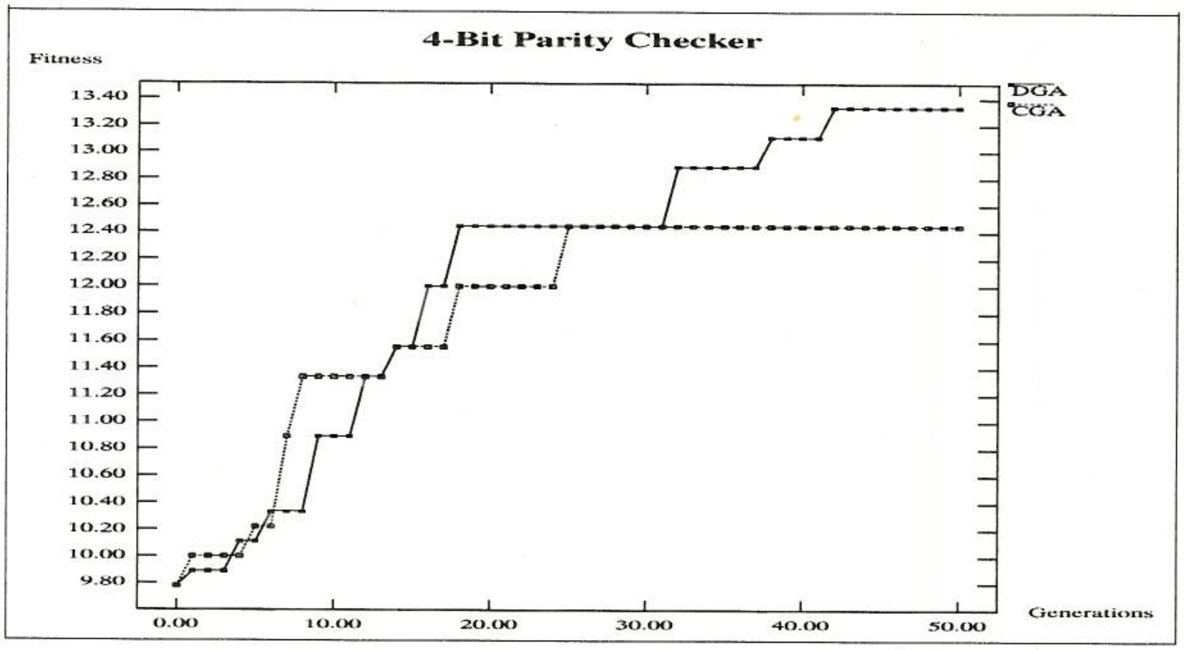


Figure 19: Performance comparison of maximum fitness per generation of a classical GA versus a DGA on a 4-bit parity checker.

GAs were comparable.) The difference in performance gets larger as the problem is scaled in size. Figure 21 and figure 22 compare the maximum and average fitness performance on a 5-bit problem. In the 5-bit experiments the choice of gates was still restricted to the same three as in the previous example. However, input choice was allowed, increasing the number of solutions in the search space. When allowed all possible gates, the performance difference is less, and is due to the large increase in the number of possible solutions and therefore a lesser degree of violation of the meaningful building block principle (see figures 23 and 24). However, as the problem becomes more epistatic, masked crossover does better than traditional crossover because it uses differential information about child fitness to bias search independent of schema defining length. Hypothesis testing using the student's t test on our experimental data proves that MX is significantly better than CX at a confidence level greater than 95% [11].

In the comparisons above we ignored the effect of selection. Figures 23 and 24 compare the performance of: 1) a GA using traditional crossover and selection, 2) a GA using traditional crossover and elitist selection, and 3) a DGA on a 5-bit parity problem. The same parameter set as in the previous examples is used although we set the number of gate types to six, increasing the number of possible solutions. This was done in the hope of coaxing better performance from the GA using traditional selection and crossover. The figures clearly show the importance of selection strategy.

The next two figures show examples of correct circuits for the 4-bit parity problem. A DGA produced the circuit in figure 25 and a classical genetic algorithm produced the circuit in figure 26. The unconnected gates in the last column do not contribute to the output,

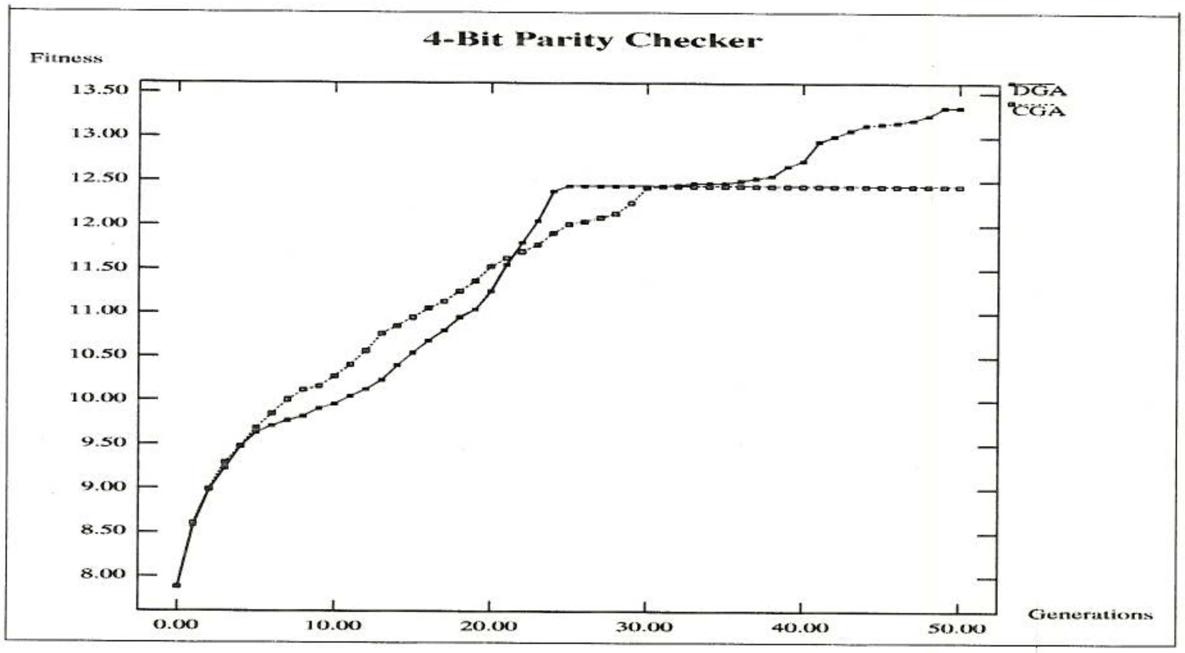


Figure 20: Performance comparison of average fitness per generation of a classical GA versus a DGA on a 4-bit parity checker.

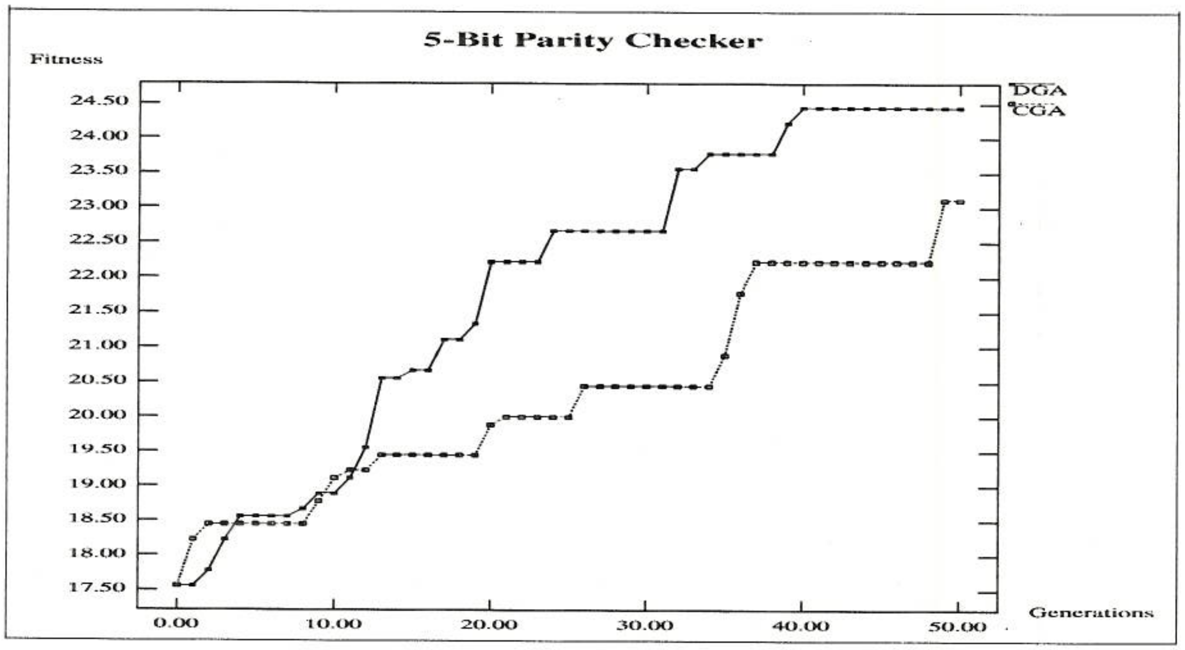


Figure 21: Performance comparison of maximum fitness per generation of a classical GA versus a DGA on a 5-bit parity checker.

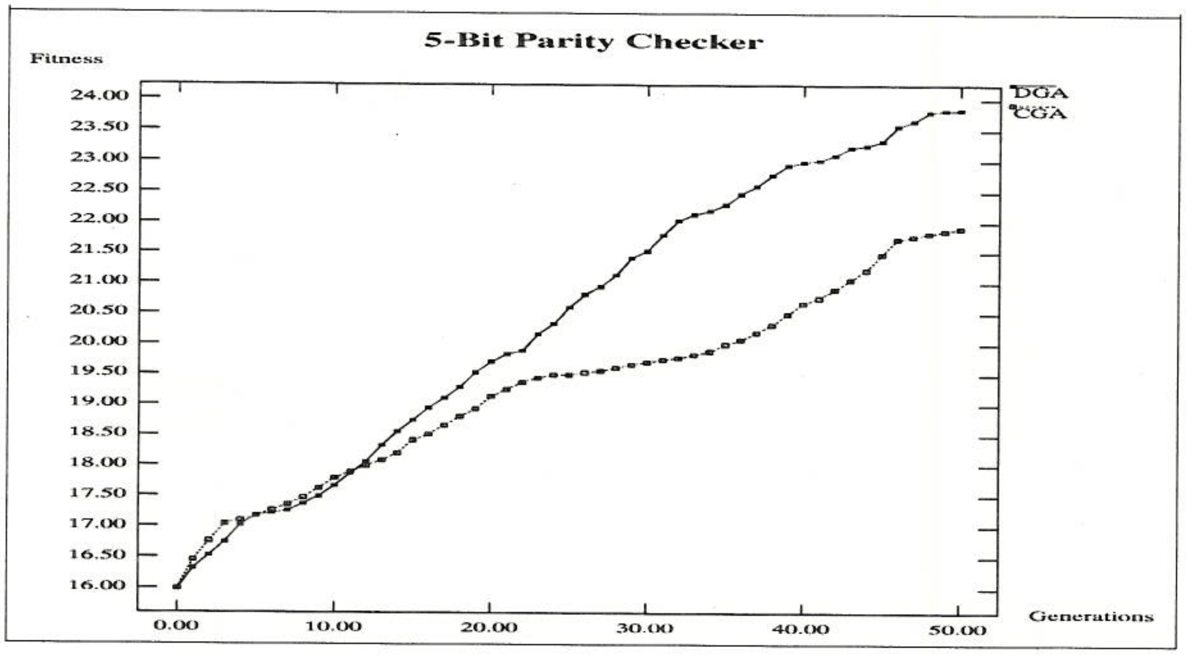


Figure 22: Performance comparison of average fitness per generation of a classical GA versus a DGA on a 5-bit parity checker.

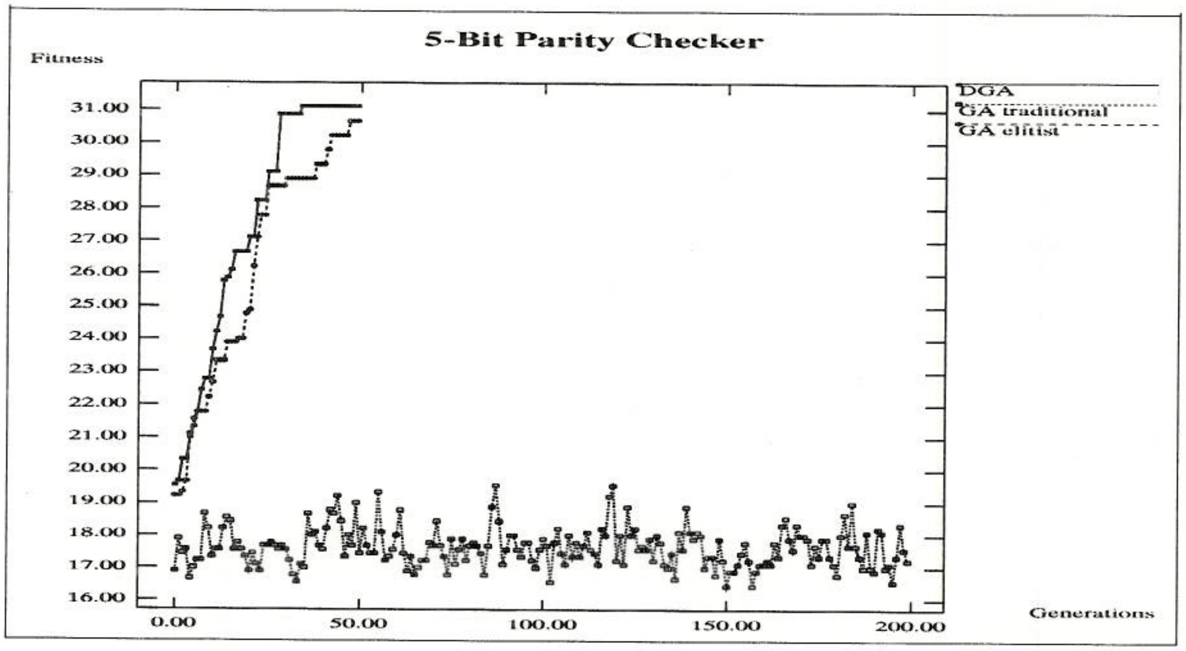


Figure 23: Performance comparison of maximum fitness per generation of a classical GA using traditional selection, a classical GA with elitist selection, and a DGA on a 5-bit parity checker.

therefore their connections have been left out. Both algorithms had the full complement of gates available.

5.1 Conclusions

To show the effectiveness of a designer genetic algorithm we need to satisfy two conditions:

1 Show that there exist encodings on which a DGA outperforms a classical GA.

2 Show that there do *not* exist encodings on which a classical GA outperforms a DGA.

The first condition is met by our results. Experiments with the standard test suite of five functions first used by DeJong empirically support the second condition[8]. Masked crossover and elitist selection result in an increase in the computational cost. We can calculate the increase in cost per generation as the sum of two costs.

- Cost of sorting the population. Sorting can be done in $O(n \log n)$ on the average.
- Cost of mask propagation. This is a constant depending on the length of the string.

A designer genetic algorithm increases the domain of application of genetic algorithms by relaxing the emphasis on schema of short defining length. We see that using masked crossover mitigates the problem of epistasis while elitist selection is crucial to good performance. The increase in cost in using a DGA is by at most a constant factor per generation. Comparing the performance of the two GAs on a problem also gives significant insights about properties of the search space. If the performance difference is large, it indicates that highly fit, low-order schema are mostly of large defining length and/or the number of solutions in the search space is low. The number of solutions is called the *footprint*. As the footprint decreases, performance difference gets larger.

Deception also plays a role in determining performance. Traditional crossover may do better on deceptive problems because it uses no information about search direction. However, since we now have a good tool with which to recognize and isolate deception we can take remedial action using messy genetic algorithms at only those points that require such action. By combining designer genetic algorithm with messy genetic algorithms we hope to extract the best properties of each.

6 Applications and Further Research

Application areas include VLSI layout design, network architecture design and of course, function optimization. Current applications of GAs in industry are mostly in the area of function optimization. Although GAs have enjoyed some success, their implementations are ad-hoc at best and do not make use of the full power of genetic search. The basic problem is again one of representation. As has been stressed before, GAs work best when the encoding follows the principle of "meaningful building blocks." Unfortunately finding such an encoding

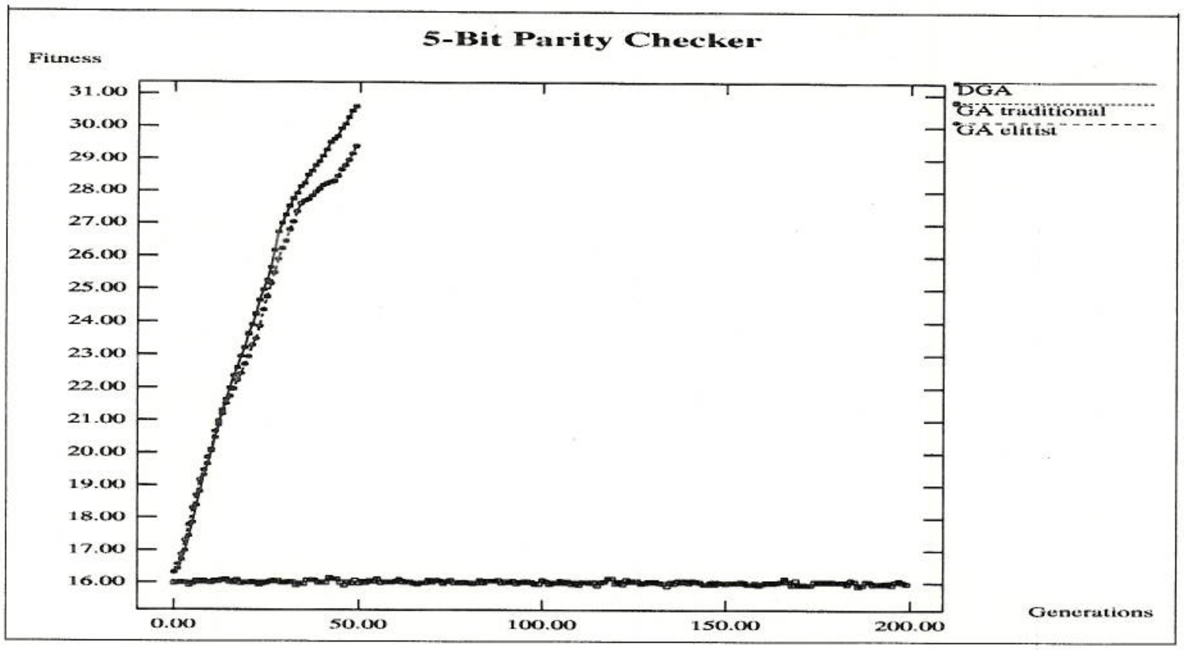


Figure 24: Performance comparison of average fitness per generation of a classical GA using traditional selection, a classical GA with elitist selection, and a DGA on a 5-bit parity checker.

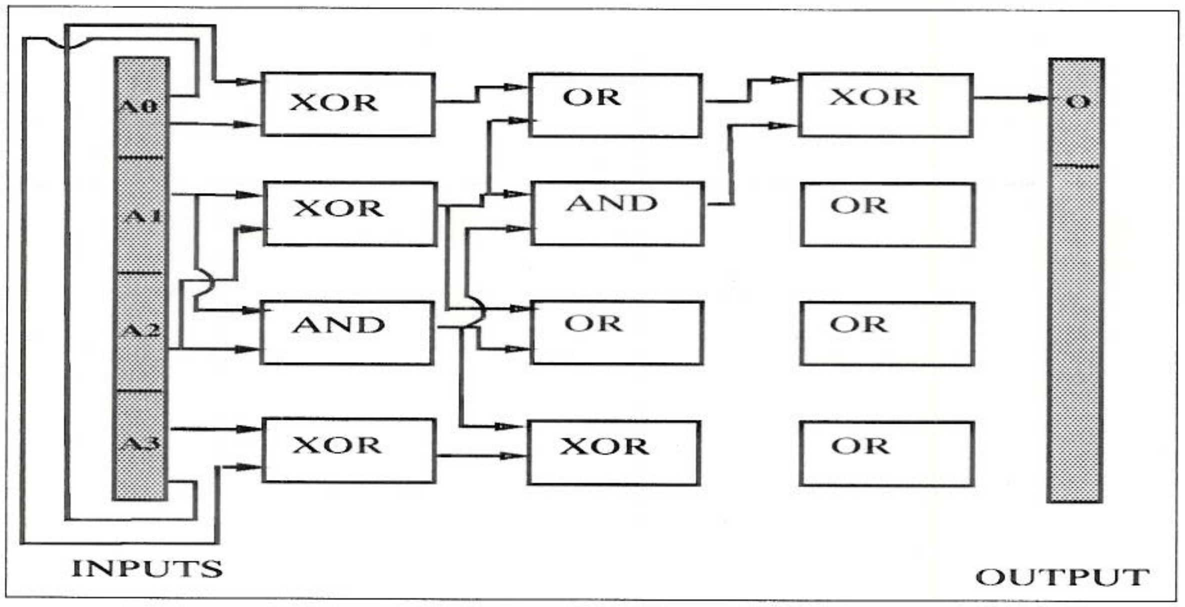


Figure 25: A Circuit designed by a designer genetic algorithm that solves the 4-bit parity problem.

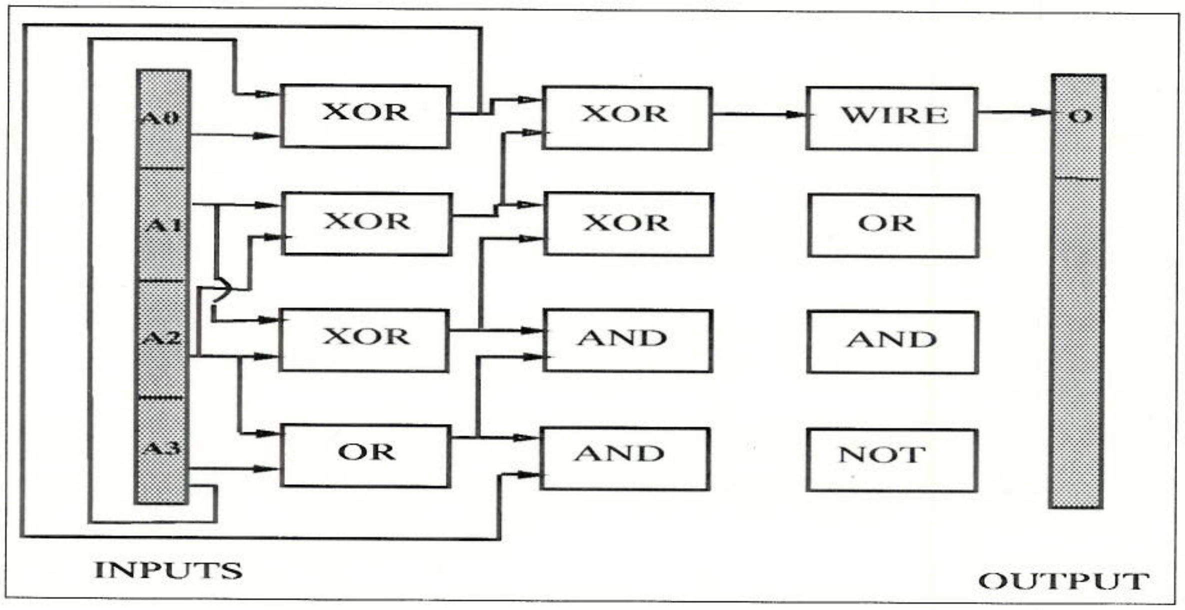


Figure 26: A Circuit designed by a classical genetic algorithm that solves the 4-bit parity problem.

is not trivial and is something of an art [14]. Most industrial implementations such as the systems developed at Lockheed and General Electric [21, 2] do not pay attention to this principle and lead to mixed or bad results. DGAs however are not constrained by such principles and do not care about the aesthetic sense of GA programmers. As such they promise to exploit the full potential of genetic search, increasing the domain of successful application.

Another application area envisioned for a DGA is in the design and analysis of protein structure. A protein is a sequence of amino acids. This one dimensional sequence is transformed into a three-dimensional shape that determines the function of a protein. The functional shape arises out of the complex interactions among the constituent amino acids and the environment. Since the number of amino acids in a protein may be in the hundreds, predicting the three dimensional structure of a protein and thereby its function is extremely difficult and remains one of the open problems in microbiology. DGAs seem tailormade for this problem since the phenotype or structure of the protein is very epistatic. Amino acids very far apart on the one-dimensional sequence may interact with each other in the three dimensional functional structure. Current work already follows a similar evolutionary approach as summarized in a recent Scientific American article[1].

Much work remains to be done before the DGA becomes a tool in an application's toolbox. This proposal only considers binary masks which piggyback on their associated strings, a constraint that needs exploration. Non-random initialization of masks to simulate traditional or uniform crossover is an interesting area yet to be explored. we can use an independent population of masks to chart directional trends and allow non-binary masks to assign weights to alleles. Future research will therefore be focused on global and non-binary masks and their possible combination. Various selection schemes need to be weighed for optimal performance. The use of special operators, for niche formation and speciation [7], dominance and diploidy [14], and the conditions for their use in designing structures need to be evaluated. Identifying and overcoming deception, is an important area of research, not only for structure design but also for the field of genetic algorithms in general. We hope to combine DGAs with MGAs as described in the previous section, to synthesize an algorithm that is impervious to many of the problems that plague current GAs. Finally, mathematical analysis must be done to put the DGA on a firm foundation.

References

[1] Beardsley, Tim., "New Order" in *Scientific American,* October 1990, Vol 263, 18-24.

[2] Bramlette, Mark F., and Cusic, Rod., "A Comparative Evaluation of Search Methods Applied to Parametric Design of Aircraft." In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kauffman, 1989, 213-218.

[3] Bridges, Clayton L. and Goldberg, David E., "The Nonuniform Walsh-Schema Transform", in *Workshop on the Foundations of Genetic Algorithms and Classifier Systems* Morgan Kauffman, (to appear) 1991.

[4] Culberson, Joseph C., and Rawlins, Gregory J. E., "Genetic Algorithms as Function Optimizers." Unpublished Manuscript, Indiana University, Department of Computer Science. 1990.

[5] Dawkins, Richard., *The Selfish Gene.* Oxford University Press, 1976.

[6] Dawkins, Richard., *The Blind Watchmaker.* Longman Scientific and Technical, Longman Group UK Limited. 1986.

[7] Deb, Kalyanmoy. and Goldberg, David E. "An Investigation of Niche and Species Formation in Genetic Function Optimization" In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kauffman, 1989, 42-50.

[8] De Jong, K. A., "An Analysis of a the Behavior of a class of Genetic Adaptive Systems." Doctoral Dissertation, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor.

[9] Eshelman, Larry J., Carauna, A. and Schaffer, J David. "Biases in the Crossover Landscape" In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kauffman, 1989, 10-19.

[10] Eshelman. L. J., "The CHC Adaptive Search Algorithm: How to have Safe Search When Engaging in Nontraditional Genetic Recombination." in *Workshop on the Foundations of Genetic Algorithms and Classifier Systems* Morgan Kauffman, (to appear) 1991.

[11] Freund. John. E., *Statistics A First Course*, Prentice-Hall, 1981.

[12] Goldberg, D. E., and Lingle, R., "Alleles, loci and the Traveling Salesman problem." in *Proceedings of the an International Conference on Genetic Algorithms and their Applications,* 1985. 154-159.

[13] Goldberg, David E., and Samtani, M. P., "Engineering Optimization via Genetic Algorithm." in *Proceedings of the Ninth Conference in Electronic Computation,* 1986, 471-482.

[14] Goldberg, David E., *Genetic Algorithms in Search, Optimization, and Machine Learning* Addison-Wesley, 1989.

[15] Goldberg, David E., "Genetic Algorithms and Walsh Functions: Part I, A Gentle Introduction", in *Complex Systems*, 3, 1989, 129-152.

[16] Goldberg, David E., "Genetic Algorithms and Walsh Functions: Part II, Deception and its Analysis, in *Complex Systems*, 3, 1989, 153-171.

[17] Goldberg, David E., Korb, Bradley., and Deb, Kalyanmoy. "Messy Genetic Algorithms: Motivation, Analysis, and First Results", TCGA Report No. 89002, Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms, 1989.

[18] Goldberg, David E., Deb, Kalyanmoy, and Korb, Bradley. "An Investigation of Messy Genetic Algorithms," TCGA Report No. 90005, Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms, 1990.

[19] Holland, John. H., *Adaptation In Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press. 1975.

[20] Jog, Prasanna D., Suh, Jung Y., and Gucht, Dirk Van., "The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem." In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kauffman, 1989, 110-115.

[21] Powell, D. J., Tong, S. S., and Skolnik, M. M., "EnGENEous Domain Independent, Machine Learning for Design Optimization." in *Proceedings of the Third International Conference on Genetic Algorithms and their Applications,* 1989, 151-159.

[22] Schaffer, David. J., and Morishima, Amy., "An Adaptive Crossover Distribution Mechanism for Genetic Algorithms" in *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Erlbaum Associates, 1987, 36-40.

[23] Schaffer, J David, and Morishima, Amy. "Adaptive Knowledge Representation: A Content Sensitive Recombination Mechanism for Genetic Algorithms" In *International Journal of Intelligent Systems* John Wiley & Sons Inc., 1988, Vol 3, 229-246

[24] Schaffer, J. David., Eshelman, Larry J., and Offut, Daniel "Spurious Correlations and Premature Convergence in Genetic Algorithms" in *Workshop on the Foundations of Genetic Algorithms and Classifier Systems* Morgan Kauffman, (to appear) 1991.

[25] Syswerda, Gilbert., "Uniform Crossover in Genetic Algorithms" In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kauffman, 1989, 2-8.

[26] Smith. D., "Bin Packing with Adaptive Search." in *Proceedings of an International Conference on Genetic Algorithms and Their Applications.* 1985. 202-206.