Designer Genetic Algorithms: Genetic Algorithms in Structure Design

Sushil J. Louis
Department of Computer Science
Indiana University, Bloomington, IN 47405
louis@iuvax.cs.indiana.edu

Gregory J. E. Rawlins
Department of Computer Science
Indiana University, Bloomington, IN 47405
rawlins@iuvax.cs.indiana.edu

Abstract

This paper considers the problem of using genetic algorithms to design structures. We relax one constraint on classical genetic algorithms and describe a genetic algorithm that uses differential information about *search direction* to design structures. This differential information is captured by a masked crossover operator which also removes the bias toward short schemas. We analyze performance and present some preliminary results. Further, consideration of this problem suggests a partial solution to the identification of the *deception problem*.

1 INTRODUCTION

The problem of designing structures is pervasive in science and engineering. The problem is:

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

As an example of the design problem, consider the *combinational circuit design problem*: Given a set of logic gates, design a circuit that performs a desired function. Two instantiations 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 1 and 2). 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.

Design is traditionally considered a creative process and so difficult to automate. Expert systems that seek to codify knowledge are currently too brittle and not

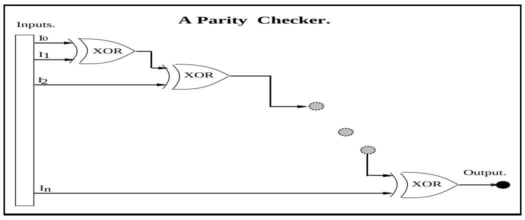

Figure 1: An n-bit parity checker.

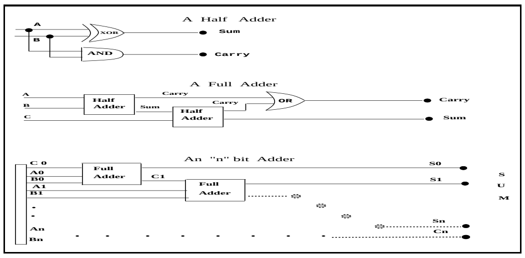

Figure 2: An n-bit adder.

applicable across a broad range of domains. However, *natural selection* has been spectacularly successful in producing a broad range of robust structures that are efficient at performing a broad range of functions. Its success is evident from the abundance and diversity of life on this planet.

Genetic Algorithms (GAs), based on natural selection should enjoy similar success in solving the problem of design. However when naively applied, their performance is less than encouraging. The difficulties lie in the enormous size of the problem, the interdependence among parts in the structure, and the biases inherent in current GAs. This interdependence is called *epistasis* and is an important aspect of well-designed structures. For example, if we use the elements of a two-dimensional array to represent gates in the textbook solution to the parity problem, the XOR gates must lie on the diagonal. Such interdependence is crucial in the design of a correct circuit [16].

The problem of structure design points out the major problem of choosing a good representation for GAs. A classical GA will do well, only if we artfully choose just the right encoding (non-epistatic), in essence, helping the search process. We solve this problem by using the fitness difference between parents and children to indicate good directions to bias search. Such directional information is easily available but cannot be explicitly stored and used in nature. Classical GAs mimicking nature also do not use this information. We however, can and do explicitly use this directional information to bias search toward high-performance schemas of *arbitrary length* thus reducing the dependence on encoding. In addition, we may identify deception by using this information alone or in conjunction with other methods and take remedial action. This approach is further developed in section three.

The next section 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 three. Preliminary results, presented in the fourth section, indicate the usefulness of DGAs. The last section covers conclusions and directions for future research.

2 CLASSICAL GAs

A genetic algorithm, first defined by Holland, is a randomized parallel search method modeled on evolution [15]. GAs are being applied to a variety of problems and are becoming an important tool in machine learning and function optimization [10]. Their beauty lies in their ability to model the robustness, flexibility and graceful degradation of biological systems. However, there has been little research on their applicability to design problems; much of the GA literature concerns function optimization. Any reference to design invariably means optimization of design parameters. In such problems the initial structure is fixed and the object is to optimize some associated cost [1, 17]. For example, in Goldberg and Samtani [9] 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 initial structure itself.

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. The basic algorithm, where $P(t)$ is the population of strings at generation t, is given below.

$t = 0$
initialize $P(t)$
evaluate $P(t)$

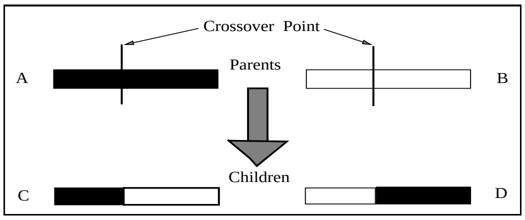

Figure 3: Crossover of the two parents A and B producing children C and D.

while termination condition false **do**
  select $P(t+1)$ **from** $P(t)$
  recombine $P(t+1)$
  evaluate $P(t+1)$
  $t = t + 1$

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 probabilistically chooses a bit and flips it. Crossover (CX) 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 3 shows how crossover mixes information from two parent strings A and B, producing offspring C and D 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.

Holland's *schema theorem* is fundamental to the theory of genetic algorithms [15]. 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 "*" is a "don't care" symbol; positions 2, 3 and 5 can be either a 1 or a 0. The *order* of a schema is the number of fixed positions in the schema, while the *length* is the distance between the first and last specific positions. The order of 1**0*1 is 3 and its length is 5. The *fitness* of a schema is the average fitness of all strings matching the schema.

The schema theorem proves that relatively short, low-order, above-average schemas get an exponentially increasing number of trials in subsequent generations. Long, low-order, high-performance schema do not play a significant role in biasing genetic search.

2.1 GAs for STRUCTURE DESIGN

Using a genetic algorithm to design a structure is like playing with a child's construction kit. Given some low level building blocks, we have to put them together so that they perform a certain function. A GA used for design manipulates low-level "tools," or building blocks, playing with their arrangements, until it finds the required structure. But there are three problems:

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. A GA (or any search method) will perform poorly in optimizing a function that is zero at all points but one [3].

Second, 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 plays an important part in determining the suitability of classical genetic algorithms to structure design. Phenotypic epistasis may not be reflected in the genotype (unless it is very carefully encoded) and so will seriously degrade GA performance.

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.

2.2 CROSSOVER BIAS

Long schemas tend to be disrupted by CX more often than shorter ones. Let H be a schema, $\delta(H)$ its 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. However, in epistatic domains, schemas of arbitrary length need to be preserved. If the encoding does not ensure that low-order schemas are short the GA will not make progress.

One way out of this is to use inversion. Inversion rearranges the bits in a string allowing linked bits to move close together. Inversion-like reordering operators have been implemented by Goldberg and others [8, 21] with some success. 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 nor the resources available to nature.

Another approach is to use a new crossover operator like punctuated crossover or uniform crossover. Punctuated crossover (PX) 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 [18, 19].

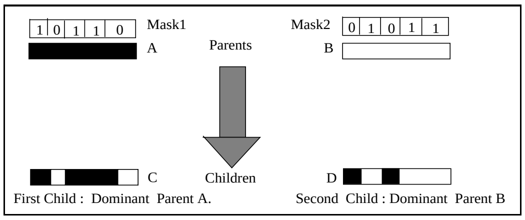

Figure 4: Masked crossover.

Uniform crossover (UX) 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 its length. Experimental results with uniform crossover suggest that this property is useful in some problems [20]. However, in design problems we would like *not* to disrupt highly fit schemas whatever their length.

None of these operators uses directional information. In the next section, we define a masked crossover operator that removes the bias toward short schemas by using directional information to efficiently bias search.

3 MASKED CROSSOVER

We define an operator that uses the relative fitness of the children with respect to their parents, to guide crossover. The *relative* fitness of the children indicates the 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 minor modifications [16].

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 4 and defined below:

copy A **to** C **and** B **to** D
for i **from** 1 **to** string-length
    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

MX tries to preserve schemas identified by the masks. Call A the *dominant parent* with respect to C; C inherits A's bits unless B feels strongly ($Mask2_i = 1$) and A does not ($Mask1_i = 0$). 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:

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

This probability of disruption does *not* depend on $\delta(H)$. Intuitively, 1's in the mask signify bits participating in schemas. MX preserves A's schemas in C while adding some schemas from B at those positions that A has not fixed. A similar process produces D. In addition, MX can combine overlapping schemas with less disruption than UX. This allows creation of schemas that would be impossible with one point crossover.[1] To ensure that the semantic interpretation of mask bits is correct, we modify masks in subsequent generations. Modifying masks will change the probability of disruption. Using fitness information to guide mask modification in subsequent generations, we would like to decrease the probability of disruption of highly-fit schemas independent of length. 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 depends on the fitness of the child relative to that of its parents. We define three types of children:

Good child: more fit than best parent.

Average child: fitness between that of the parents.

Bad child: less (or equally) fit than worst parent.

With two children produced by each crossover, and three types of children there are a six cases, with associated interpretations and possible actions on the masks (see figure 5).

[1] A simple example: the string 111 cannot be produced from 101 and 010 by one point crossover

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 5: Mask rules for the six ways of pairing children.

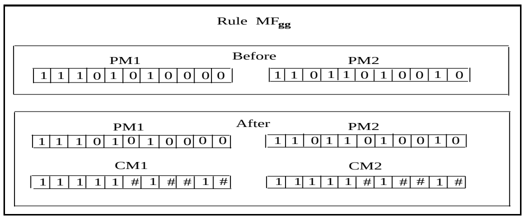

Figure 6: Mask rule MF_{gg}: Example of mask propagation when both C1 and C2 are good

3.1 MASK RULES

This section specifies 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 want to encourage search in the convex subspace defined by these loci. The idea is to search in areas close to one parent with information from the other parent providing some guidance. *Note that in MX, this is done without regard to length.* A mask mutation operator that flips a mask bit with low probability acts during mask propagation. We provide two representative mask functions rather than all, to give an intuitive understanding of their form. These are MF_{gg}, used when both children are good and MF_{bb} used when both children are bad (for more details see Louis and Rawlins [16]).

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 $C2$. For the figures in this section, the "#" represents positions decided by tossing a coin.

1. MF_{gg}: Both children are good.

   Summary: Encouraging behavior. Parents' masks are OR'd to produce the children's masks, ensuring preservation of the contributions from both parents (see figure 6).

   Action:

   - $CM1$ and $CM2$: OR the masks of $PM1$ and $PM2$. If there are any 0's left in $CM1$, toss

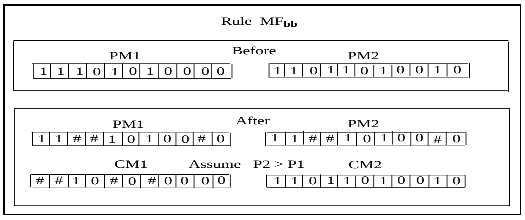

Figure 7: Mask rule MF_{bb}: Example of mask propagation when both C1 and C2 are bad.

a coin to decide their value.

- $PM1$ and $PM2$: No changes except for those produced by mutation.

2. MF_{bb}: Both children are bad.

   Summary: Discouraging behavior that 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 7.

   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 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 $PM2$.

With mask propagation through mask rules, directional information is explicitly stored in the masks and used by the crossover operator to bias search. Using the fitness of children relative to that of their parents we preserve and recombine high performance schema and disrupt low-performance schema, independent of schema length. The main features of MX are, storage and use of directional information, and independence from length of schemas. We think of masked crossover as a golden mean between the disruptiveness of UX and the bias toward short schemas of CX. Compared to the size of the search space when using inversion, $2^l!$, a genetic algorithm using MX searches only 2^{2l}.

Many sets of 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.

MX presents a problem when using classical selection procedures. The classical strategy replaces the original population with the new children produced, but does 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 [6]. 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. We use this *elitist* selection strategy to guarantee convergence. Another problem which may occur is that although MX preserves schemas of arbitrary length, the fitness information itself may be misleading. Such problems are called *deceptive* [10]. When fitness information is misleading we expect a GA using MX 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 GA (CGA) in the crossover operator (MX) and in the selection strategy (elitist) used.

Identifying and overcoming deception, is an important area of research. 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 [13, 14]. We suggest an approach satisfying both criteria, using designer genetic algorithms.

Deception can be statically identified using the ANODE algorithm suggested by Goldberg [11, 12]. Recent results indicate that the Nonuniform Walsh-Schema Transform (NWST) [2] 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.

4 RESULTS

We compare a designer genetic algorithm's performance 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 for masks and genotypes is 0.04. These numbers were found to be optimal through a series of experiments using various population sizes and probabilities. The graphs in this section plot average fitness over ten runs.

Each genotype is a bit string that maps to a two-dimensional structure (phenotype) embodying a circuit. 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 $S_{i-1,j-1}$. 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 GA maximizes the sum of correct responses (For more detail see Louis and Rawlins [16]). It is also possible to use only a subset of the possible inputs, reverting to the complete set only when the population converges prematurely. This results in significant savings in time.

We compare the performance of a classical GA using elitist selection with a DGA on a 2-bit adder problem. The graph in figure 8 shows 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. As explained earlier, since MX uses fitness information to bias search, it is more easily mislead than traditional crossover. Even if a problem is deceptive, it does not mean that no solutions can be found. Figures 9 and 10 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

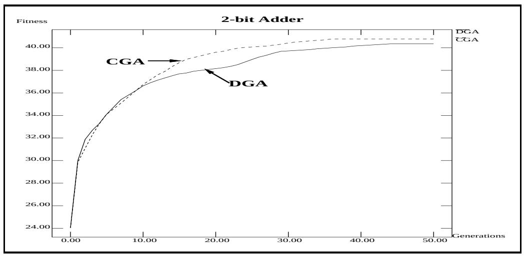

Figure 8: Performance comparison of average fitness per generation of a classical GA versus a DGA on a 2-bit adder.

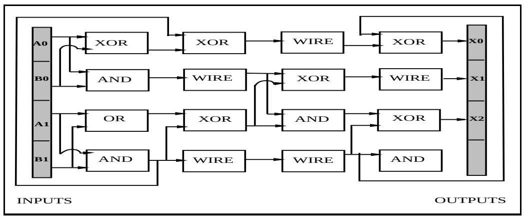

Figure 9: A 2-bit adder designed by a designer genetic algorithm.

because it does not affect the output. Although we have not done a rigorous study of the types of solutions found by both algorithms, we see that the circuit designed by the DGA depends on long schemas. For example S_{03} gets its input from S_{32} which is 11 units away, where 11 is large when compared with 16, the length of the genotype. This is in marked contrast to the CGA circuit.

We now consider the parity problem. The encoding described above will violate the "principle of meaningful building blocks" with regard to the solution to the parity problem as shown in figure 1 [10]. Since diagonal elements of S (the phenotype) are further apart in the string, any good subsolutions (highly fit, low-order schemas) found will tend to be disrupted by traditional crossover. MX however, will find and preserve these subsolutions as its performance is independent of 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,

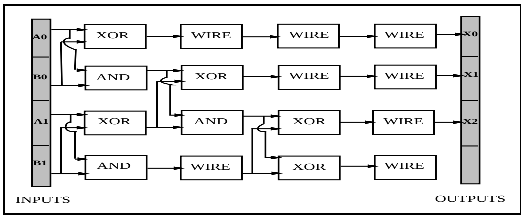

Figure 10: A 2-bit adder designed by a classical genetic algorithm.

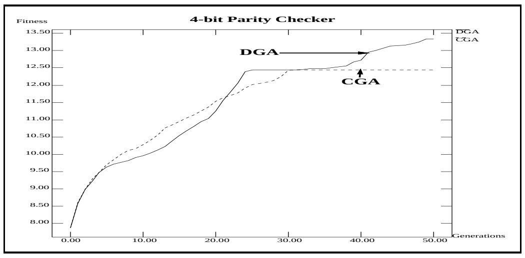

Figure 11: Performance comparison of average fitness per generation of a classical GA versus a DGA on a 4-bit parity checker.

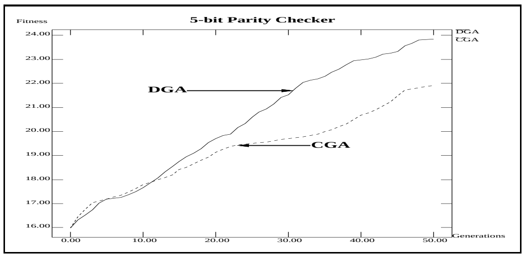

Figure 12: Performance comparison of average fitness per generation of a classical GA versus a DGA on a 5-bit parity checker.

modulo the number of rows). Although this reduces the size of the search space, CX disrupts low-order schemas and therefore performs worse than the DGA. Figure 11 shows this for a 4-bit parity checker. The difference in performance gets larger as the problem is scaled in size. Figure 12 compares the 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 figure 13). However, as the problem becomes more epistatic, MX does better than CX. 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% [7].

In the comparisons above we ignored the effect of selection. Figure 13 compares 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 figure clearly shows the importance of selection strategy. Finally, figure 14

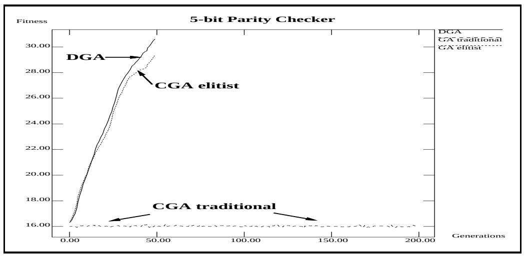

Figure 13: Performance comparison of average fitness per generation of a traditional CGA, an elitist CGA, and a DGA.

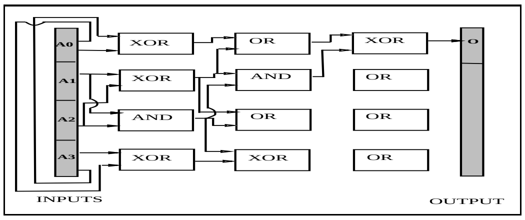

Figure 14: A Circuit designed by a DGA that solves the 4-bit parity problem.

shows a 4-bit parity checker produced by the DGA.

5 CONCLUSIONS

We have shown that a designer genetic algorithm relaxes the emphasis on schema of short length. This increases the domain of successful GA applications since a GA programmer no longer needs to follow the principle of meaningful building blocks. Using masked crossover mitigates the problem of epistasis while elitist selection is crucial to good performance on design problems. The increase in cost in using a DGA is by at most a constant factor per generation. This comes from the cost of sorting a population of size n ($n \log n$), and a constant cost for mask propagation. Comparing the performance of the two GAs on a problem also gives significant insights about properties of the search space.

Selection plays the largest part in biasing genetic search. We can think of a genetic algorithm as a search process at two levels. At the selection level, search is biased by fitness information. However, at the recombination level, search is essentially random (in classical GAs). Using fitness information to bias search at the recombination level allows a DGA to do better as is indicated by our results. However, if the fitness information is misleading, a GA will be led astray. The DGA will be misled at both levels, in contrast to the the CGA which will be mislead only at the selection level.

Experiments with the standard test suite of five functions first used by DeJong show no significant difference in performance [4]. In the experiments we used the same elitist selection strategy for both the DGA and the classical GA. These results were to be expected as DeJong's criteria for choosing his functions were not based on the epistatic or deceptive properties of these functions.

This paper uses a simple representation and only considers binary masks which piggyback on their associated strings. Mathematical analysis of the effects of mask rules on these simple masks is being done. We are also looking at more general representations and non-binary masks. Finally, identifying and handling deception dynamically, forms the thrust of our current research.

References

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

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

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

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

[13] 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.

[14] 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.

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

[16] Louis, Sushil. J., and Rawlins, Gregory J. E. "Using Genetic Algorithms to Design Structures," Technical Report No. 326, Department of Computer Science, Indiana University, 1990.

[17] 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.

[18] 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.

[19] 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

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

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