A Simple Representation Technique
to Improve GA Performance

Steven L. Keast
Department of Computer Science and Software Engineering
Auburn University
Auburn, AL.
keastsl@auburn.edu

November 13, 2003
Abstract

This paper proposes a new phenotype-to-genotype encoding representation technique that helps in several areas of Genetic Algorithm (GA) performance. First, it makes it possible to eliminate the problem of illegal gene values from being generated from the crossover operator in most situations. It does this by encoding the traits within the chromosome in such a way as to guarantee no illegal traits can be created. Next, it can increase the variability of offspring, even if the parents are identical, thus increasing the likelihood that the GA will not get stuck on sub-optimal solutions. Finally, it increases the GA’s performance by allowing a more rapid search through the hypothesis space. This is achieved by the prior mentioned genotype encoding technique and the variation in offspring that is seen due to this encoding scheme.

A new operator called Cloning is introduced that is a key contributor to the performance of the GA. Cloning creates new individuals that have gene values identical to the hypotheses from which they were cloned, but with dramatically different gene representations. This is possible due to the technique used to encode the individual traits. By cloning hypotheses, later crossover operations create offspring that can be very different than their parents, thus increasing the speed in which the hypothesis space is searched.

Finally, this paper discusses the test results of the GA and characterizes its performance. Areas covered include investigating the GA’s performance when Tour size, population size and cloning rate are varied.

Introduction

One of the key problems faced by genetic algorithms is how to represent the phenotype as a genotype – that is, to encode an individual in terms of specific gene values. Making a poor decision on the phenotype representation has significant consequences. It can permit the GA to get stuck on a sub-optimal solution. This can occur if the population converges on a less-than-perfect hypothesis. If the population becomes the same non-ideal chromosome, further progress of the algorithm will be difficult. Another potential issue is the creation of chromosomes that have invalid gene values. This can occur during crossover or mutation. To handle this problem the individual can simply be tossed, or it can be further modified to create a valid individual. In either case, processor cycles are lost and the search time is increased.

One way the representation problem has been addressed has been the simulation of biological processes. Genotypes have been expressed using techniques such as variable length encoded strings, non-coded segments (introns) and redundant genes. GAs using these techniques have shown improved performance in searching the hypothesis space for the target solution.

Breeder Genetic Algorithms (BGAs) [6] take a different tack in solving the problem of representation. They use floating point numbers to encode gene values, and add small random floating point numbers to these numbers for the mutation operation. Researchers such as Salomon have shown that BGAs can reduce the search effort from $O(l^4 \ln l)$ for simple binary encoding to $O(l \ln l)$ [6]. Salomon also describes in [6] the issue of the Hamming distance between adjacent numbers and how this distance affects the probability of
finding a solution. He suggests the use of Gray Codes for gene values as one means of increasing the probability by minimizing the distance between adjacent numbers.

This paper offers a simple new representation technique that solves the problems of the creation of illegal offspring and of getting stuck on sub-optimal solutions. Gene values, rather than being represented by Gray Codes, simple binary strings, or floating point numbers are represented by 32-bit binary values. The paper also introduces a new operator, Cloning, that replaces the traditional mutation operator used in GAs today. Cloning creates new individuals that are identical to their clone/parent, yet with dramatically different 32-bit gene representations.

This paper will cover the following areas:

♦ Statement of the Problem – A more detailed discussion of the GA issues this new representation technique tries to resolve.
♦ Related work – A look at recent work in both the biological representation and BGA-type algorithms.
♦ The New Genotype Representation Technique and GA Enhancements – A presentation of the new representation technique and GA implementation requirements to take advantage of it. A new operator called Cloning is also introduced that helps improve the performance of the GA.
♦ GA Design for Testing the Representation Scheme and Cloning Operator – A discussion of the design of the specific GA used to generate all test results reported later in this paper.
♦ Target Problem for GA Testing and its Common Fitness Function – Outlines the target problem and the fitness function used for all GA test results.
♦ Test Results – Reviews all test results gathered on the GA's performance.
♦ Conclusions – Summarizes the paper's key points.

Statement of the Problem

Genetic algorithms can have problems in a few areas. First the algorithm can get stuck on a sub-optimal solution and have difficulty progressing past this point. The issue here is the population of candidate hypotheses converges on a common hypothesis allele and this allele isn’t the optimum solution. Once the population has the same allele, no amount of selection and crossover operations will make any difference. Identical parents will produce identical offspring. The only solution to this problem is for the mutation operator to resolve the issue by randomly making a better hypothesis than that which is available in the current population and allowing the GA to move forward.

Another problem is what to do with illegal hypotheses (allele value is not valid) that are created by the crossover and mutation operators. This problem is caused by having traits within the allele that have a total number of value options that are not a base 2^n number. For example, a trait could have a total of five possible values. This would require three bits to encode, permitting a total of eight possible values. This means the mutation and crossover
operators have a 3 in 8 chance of creating an invalid value for this trait. One easy solution to this problem is to simply kill off the illegal offspring and try again. This, however, slows the algorithm’s progress requiring more time to find the optimum hypothesis. One could also randomly select a valid value to fix an illegal offspring fixing the problem but this again requires additional CPU cycles and delay algorithm progress. A better solution would be to have a GA that didn’t create illegal offspring, thus eliminating the problem.

Related Work

Although much of the work that has gone into the representation problem has dealt with applying a specific solution to a unique problem, there is also much work to date that deals with a more generalized study of representation and how it affects GA performance. This work can be broken into two broad areas of study: GAs that simulate biological processes in an effort to improve performance, and GAs that propose representation techniques that give the GA an advantage in searching the hypothesis space efficiently.

Biological GAs

Biologically-based GAs break the representation problem down typically into the following key components [1]:

♦ Genomes vary in length due to insertions, deletions and recombination.
♦ Genes on a genome can be position-independent.
♦ Genomes can contain non-coding sequences (Introns [3]).
♦ Genomes can have competing or duplicate genes.
♦ Genomes have overlapping reading frames.

The biological representation of the phenotype tries, through the simulation of known biological processes, to solve problems that arise during GA operation and improve GA performance. One area addressed by this technique is the use of Introns [1], non-coded sequences within the genotype, to solve the problem of producing illegal offspring from the crossover and mutation operators. By putting in a guard-block (the “Intron”) between valid gene sequences, gene regions can be protected from these operations [1]. The addition of Introns can have one other major benefit – GA operation can be improved by as much as a factor of 10 [3].

Besides non-coded sequences, biological-based GAs also can vary the length and the position of the gene within the genome [5]. Variable length and position representations require the genome to be scanned to determine the phenotype value. The actual gene value within the genome is delineated with START and STOP codes [5] and these codes can be found on multiple reading frames. The crossover and mutation operators are much the same as a standard GA but due to the variability of gene size and position within the parents, the crossover operator must match up closely in both parents (same region on the genes) before recombination is performed [5]. The mutation operator typically uses a fairly low probability of 0.001 in many cases for best performance.
The use of duplicate genes is an attempt to mimic biological processes where the same gene that produces a specific protein, for example, can be duplicated in multiple locations within a particular genome. Traditional GAs usually specify gene values in only one location within the representation. By allowing for duplication, the work area is increased and more potential candidates are available to more quickly scan the search space for a solution.

Biological GAs, by using representations that more closely follow natural processes, can have improved performance over a more standard representation technique. The key to this representation is that it allows for a “considerable degree of self-adaptation” [5]. The genome with variable length and position genes, duplicate genes and non-coded sequences allows for far more variability in future generations compared to other representation forms.

**Designed-for-Performance GAs**

Several different techniques are used to speed up GA performance outside the scope of biological process simulation. One idea is the messy GA (mGA) [2] proposed in 1988 and first published in 1989 by Goldberg, Korb and Deb [2]. The mGA represents genes as a pair of numbers with the gene value defined as a binary or floating-point number and the gene name included as one component of the pair. An example of a mGA gene could be (1 7) that defines gene 1 with a value of 7. The chromosome of a mGA is a collection of gene values that can include multiple defines for the same gene and certain gene types could even be missing from the chromosome.

**GA Problems**

Salomon points out in [7] several issues involved with GA design. The main concern is the revisiting of already sampled points in the search space due to the random application of “variation operators” [7]. By revisiting these points, GA performance is reduced and less predictable. The paper also looks at the Scheme Theorem, Breeder GAs and probabilistic estimates of GA performance arguing that all misstate the computational complexity of a GA.

After establishing the shortcomings in current GA thinking, a more deterministic approach is proposed that yields predictable GA results (in the author’s opinion). The central concept is that the application of the mutation operator needs to be done in a deterministic way and the crossover operator is not needed. In the pseudo code supplied by the author, the mutation operator is applied in what the author calls an “optimization step” on each of a series of nested for loops. Due to the nature of the operator’s application, it is deterministic.

Although [7] seems far from the representation techniques proposed in this paper, one result of the new representation technique is to generate a great deal of variability in the offspring. This could be seen as having the same effect as is seen in [7] by its author’s application of the mutation operator in a deterministic fashion.
The New Genotype Representation Technique and GA Enhancements

As Nicholas J. Radcliffe pointed out in [4], maximizing the length of the chromosome “will give rise to the greatest degree of intrinsic parallelism” and “achieve maximum processing efficiently” from the GA. The binary representation of gene values by itself can go a long way in increasing the chromosome length. Whether coding of traits is in Gray Codes or a simple binary representation, the chromosome length initially increases quickly as the number of values for a gene increases.

New Genotype Representation and Chromosome Implementation

The proposed representation technique tries to take advantage of the property of increased chromosome length leading to increased GA efficiently. Gene values within the chromosome are encoded as 32-bit integers with all numbers within the integer being legal values used to determine a gene’s value. By using such a large number for each gene, chromosome length is far larger than in most typical GAs. It should also be pointed out that 32-bit integers don’t have to be the upper limit of a gene’s length for this representation. Other representations can be done that exceed the 32-bit length suggested in this paper and further increases in efficiency might be possible.

Genes within the chromosome are initially assigned values that are created by a 32-bit random number generator. The actual gene value is calculated by taking the modulo of the 32-bit integer as follows.

\[
\text{GeneValue} = (32\text{BitInteger} \mod \text{TotalNumberOfGeneValues})
\]

If, for example, a gene represents eye color and the legal values for eye color are blue, green, brown and hazel, the variable TotalNumberOfGeneValues will be set to 4. The total number of ways any gene value could be represented by the 32 bit integer is calculated by dividing the size of a 32-bit integer by the variable TotalNumberOfGeneValues \(\left(2^{32} / \text{TotalNumberOfGeneValues}\right)\). The effect of using such large numbers to represent each gene means the search space is greatly increased.

A chromosome can be defined with a structure. For example, an individual that has four genes would be characterized with the following structure:

```c
struct individual {
    int    geneA;
    int    geneB;
    int    geneC;
    int    geneD;
};
```

Although the new representation technique consumes more memory than a technique that allows only one number per gene value and packs the genes together in an attempt to minimize memory space requirements, computers today have much larger memories than even just a few years ago. Using more memory to define a chromosome isn’t an issue today.
Also accessing genes that are defined as members of a structure is far easier to do than dealing with shifting and sorting a compressed binary string optimized for size.

Initial Population Generation

The initial population is created in the same fashion as typical GAs – using a random number generator to create the values for each gene within an individual’s chromosome. The only difference is that the random number for this representation must be 32 bits in length.

Offspring Variability Increases Speed through the Search Space

The key to the speed of a GA using this representation technique is the variability of the offspring caused by the crossover operator. Whether the crossover operator used is single or double, as long as the crossover operation can slice through any location within the gene, this representation will create children that have a high probability of varying from their parents. Note that this representation technique has no advantages when uniform crossover is used.

To illustrate how variability can be achieved in offspring, the following two individuals are composed of four genes with a total of thirteen value options per gene. Using the modulo operation on each gene, both individuals have gene values of 6, 1, 11, 11.

Individual A: 0x0b294f20, 0x34ec43e0, 0x547e6372, 0x27a3785d
Individual B: 0x05c915b8, 0x24397f9f, 0x33b407fe, 0x2de40cd6

Using single point crossover to slice through the middle of gene two of each chromosome and then recombining the parents to create two children, the following new individuals will be created.

Child A: 0x0b294f20, 0x34ec7f9f, 0x33b407fe, 0x2de40cd6
Child B: 0x05c915b8, 0x243943e0, 0x547e6372, 0x27a3785d

Child A now has the gene values of 6, 8, 11, 11 and Child B is 6, 7, 11, 11. In a more traditional representation technique, slicing through a gene from parents having the same value in that gene would produce no variation for that gene value in the children.

By creating variability in the children with the crossover operator, a mutation operator isn’t required with this representation. Also the problem faced by many GAs requiring the use of mutation to move the population off of a sub-optimal solution occurs naturally with this representation.

Cloning – a New Operator to Replace Mutation

A GA using the new representation technique doesn’t use the mutation operator. Instead it uses a new function called cloning to help nudge the algorithm forward when it might otherwise get stuck on a sub-optimal solution. Cloning simply recreates an individual that has exactly the same gene values (modulo values) as the original individual but with very
different integers representing the values. To illustrate how cloning works, the following two individuals are comprised of four genes with each having thirteen values per gene. Individual B was cloned from Individual A using the cloning operator.

Individual A: 0x0b294f20, 0x34ec43e0, 0x547e6372, 0x27a3785d
Individual B: 0x05c915b8, 0x24397f9f, 0x33b407fe, 0x2de40cd6

Although the two individual seems very different, if the modulo 13 of each gene is taken, they both decode to the following values. This means both individuals will have exactly the same fitness even though they appear to be dramatically different.

6, 1, 11, 11

In summary: Cloning is a way to create greater diversity in the generations that follow from the crossover operations that are applied to the parent chromosomes. When a gene is sliced, the chances of the recombined genes from two parents creating the same modulo value in the offspring gene is far less than in traditional representation techniques. In a traditional encoding scheme, two parent genes that have the same value will always produce offspring that have this same value in this gene no matter where a crossover operation might happen within the gene. By cloning and replacing individuals in the population with their clones, variability can be generated in succeeding generations and the search space more easily covered with fewer crossover operations.

The "Even-Valued Gene" Problem

One major problem arises from using the new phenotype representation technique; genes within the genotype that have an even number of possible values don’t converge towards a solution. This is called in this paper the “even-valued gene” problem, and it does have a major impact on how the algorithm performs.

Two techniques are proposed to solve this problem. The first uses the gene as an encoded number requiring decoding before the actual gene value is obtained. The second technique modifies genes that have an even number of values with one additional value that is illegal. This solves the problem but also creates offspring that have possible illegal gene values that need to be dealt with.

To illustrate the problem, the following graph shows the results of the new representation technique trying to solve the sample problem defined in the section “Target Problem for GA Testing and Its Common Fitness Function” below. The number of iterations through the GA loop was limited to 1,000. Without the limit placed on the number of iterations, the loop in many cases would go on forever.
The first idea proposed to solve this problem was encoding the gene value. This technique does a very simple operation on a gene’s representational integer to find the final gene value; the gene is rotated by the number in its bits D0 through D2 with eight added to create the final amount by which the integer is rotated.

\[ \text{GeneValue} = (\text{GeneValue \ rotate ((GeneValue \ & \ 0x7) + 8))} \]

By using this idea, genes that have an even number of values can still be represented by integers that are of odd and even types. When these integers are combined through the crossover operator, the result will create children that have a reasonable amount of diversity. The following graph shows the results of applying this simple solution to the sample problem and the resulting improvement in the algorithm’s performance.
Another option is to not allow even-valued genes in a chromosome. Although this idea is against the original intent of the algorithm to not create illegal values in children chromosomes that need to be dealt with, it does offer another solution to the problem. The implementation of this technique is to simply add one additional illegal value to all genes that have a total number of value options that are an even number. When crossover is performed, children that are created that have illegal valued genes are simply discarded. The following graph shows the results of modifying the GA with this scheme to solve the problem.
This solution to the problem actually performed better than the earlier discussed encoding technique for most test cases. The one drawback to this implementation is the crossover operation can create children that have illegal gene values. The occurrence of this happening on the test problem was captured and graphed below. As can be seen from the following graph, the percentage of children discarded due to illegal gene values drops as the number of total gene values increases.

![Illegal children From Crossover Operator](image)

Both techniques to solve this problem were tested and the results captured in the “Test Results” section later in this paper.

**GA Design to Test Representation Scheme and Cloning Operator**

The GA modifications proposed in this paper deal with the chromosome representation problem and a new operator called *Cloning*. To test these additions, a complete GA needs to be coded that includes these enhancements. The following outlines the basic design of this GA and how certain parameters that control the GA’s operation will be varied during the testing phase of this project.

**Hypothesis Representation**

Each individual will be represented by a structure with one 32-bit integer assigned to each trait within the allele. For example an individual that is represented by four traits would be defined as follows:
typedef struct
{
    unsigned int    traitOne;
    unsigned int    traitTwo;
    unsigned int    traitThree;
    unsigned int    traitFour;
} INDIVIDUAL;

During the test phase of this project, the number of traits will be varied to determine two things. First by varying the number of traits, the number of hypotheses being searched is varied as well. The search performance will be compared to the size of the search space to see how the GA scales to the problem size. The number of traits will also be varied to determine if the algorithm has any biases towards the number of values in a trait.

Population

The population is the group of potential parent hypotheses that are available for the selection process. The initial population will be randomly generated. During testing of the GA, the population size will be varied to see how population size impacts the performance of the GA.

Selection Technique

Tournament selection will be used for all GA testing. The Tour size will be varied to determine GA performance as a function of Tour size. The selection process will always only select two parents for breeding.

Crossover Operator

In all cases single-point crossover will be used to create two offspring from the selected parents. A maximum of one offspring can survive. The most-fit offspring is determined and it can replace the worst individual in the population if its fitness is better than that individual’s fitness.

Cloning Operator

The cloning operator will replace the mutation operator that is usually a standard feature of a GA. The cloning operator will be applied on the basis of algorithm stagnation. The stagnation is measured as a function of offspring not replacing the worst parent in the population. The number of times this replacement does not occur is counted and at some defined level, the cloning operator is applied. The level the at which the cloning operator is applied at will be varied during the test phase to see how GA performance is affected by this variable.
Target Problem for GA Testing and its Common Fitness Function

The problem given to the enhanced GA is searching a hypothesis space for a random four-digit, variable-base number. The number's base is changed to test issues dealing with the number of values associated with a gene. The even-valued gene problem, for example, is easily seen when the number’s base is divisible evenly by two. The number of digits in the number will be fixed at four.

The fitness function will measure the distance a hypothesis is from the goal number by comparing each digit in the hypothesis with the corresponding digit in the goal number. The absolute value of the difference for each digit is added together to produce the fitness of any individual. The goal number is reached when the fitness function returns a fitness value of zero.

Test Results

The performance of the GA was gathered by running a number of test runs and then averaging the results. As was pointed out in [9] “performance measurements are generally taken as average over sets of runs”, so averaging GA results is key to producing accurate data. In all test cases the GA was run on a problem for a total of 500 passes. The GA was also limited to 1,000 iterations on a search before the search was aborted. The failure to find the optimum hypothesis wasn’t gathered in the test results and the number 1,000 was used as the search iteration number on all failures.

Tests were focused on gathering data on various GA performance variations caused by varying specific parameters intrinsic to the GA’s design. Parameters such as population size, tour size and problem size were varied with the GA’s ability to find the solution determined by the number of iterations through the crossover operator. The Cloning operator rate was also varied and its affects on search performance determined.

All results will show two graphs; one for each solution to the even-valued gene problem. In most cases the two solutions perform almost identically on the problem but a lot of variation can be seen when the problem size is varied, creating several searches that bring the problem out of the algorithm.

The test results also include a test run on the GA to evaluate its robustness in dealing with small populations. Although this test has little to do with GA performance, it does demonstrate the GA’s ability to move towards a solution even when the GA is configured in a way that would stop other GAs and leave them stuck on a sub-optimal result.
Population Size Varied

This test varies the population size while keeping the problem size, cloning rate and Tour percentage fixed. The Tour size does vary as a function of population size because its size is fixed at 10% of the population size. The cloning rate was 100%. This means the result of the crossover operation is cloned if the child created by the crossover operation doesn’t improve upon the fitness of the parents. The problem size was fixed as a four-digit base 33 number. The selection of a base 33 number means the even-valued gene problem will not be apparent in the resulting data.

As can be seen from the graphs below, population size doesn’t have a lot of impact on GA performance. In fact, smaller initial population sizes do slightly better than larger ones. One possible reason for this behavior is that the GA is more focused when the population size is small and able to create faster forward progress with fewer individuals.
Problem Size Varied

This test varies the problem size while keeping the population size, Tour percentage and cloning rate constant. The population was fixed at 50 individuals, the Tour percentage was set at 10% of the population size (in this case 5), and the Cloning rate was set at 100%. The problem size was controlled by generating a target four-digit number whose base was varied from base 4 to base 32. Creating target numbers with even bases will create the even-valued gene problem and the difference in the two techniques to resolve this problem is apparent in the graphed data below.
The performance of the GA can be defined to be $O(n \ln n)$. The encoded scheme to resolve the even valued gene problem shown in the first graph doesn’t meet this limit on even based numbers due to the solution not fully fixing the anomaly. The scheme that creates illegal gene values to fix the problem does stay within the $O(n \ln n)$ limit.

**Tour Percentage Varied**

This test varies the Tour size by changing the Tour percentage while keeping the population size, problem size and cloning rate fixed. The population size was set at 50, the Cloning rate was 100% and the problem size was fixed at a four-digit base 33 number.

One surprising result seen in the graphs below is that the performance degrades as the Tour percentage is increased to 100% of the population size. At Tour percentages from 10% to 40% of the population size, the GA performance is roughly the same. Above 40% however the iterations required through the crossover operator to find the target hypothesis increases rapidly. At the 100% Tour percentage level, the search time is increased by roughly 30% over the search time for a 10% Tour percentage.

An explanation for this is that higher Tour sizes have a tendency to select the same individuals from the population over and over again. At smaller Tour sizes, it is more likely more individuals from the population will be selected and this will create more diversity in the offspring, moving the GA forward more quickly to a solution.
Clone Rate Varied

This test varies the cloning rate while keeping the population size, Tour size and problem size constant. For this test the population size was fixed at 50, the Tour percentage was set at 10% and the problem size was a four-digit base 33 number.

The cloning rate isn’t a probability for applying the Cloning operator but instead is a number representing the number of iterations through the crossover operator without any children replacing individuals in the population. For this test the rate was varied from 1 to 100 misses before the operator was applied.

One thing is obvious from the graphs below of the test results; the Cloning operator has a major impact on the GA’s performance. Cloning the parents at a 100% rate when a crossover operation doesn’t create children that have fitness better than their parents improves GA performance by 3-fold over cloning at the lowest rate tested (100 misses before Cloning is performed).
Performance vs. Cloning Rate
(Encoded Gene Value)

Performance vs. Cloning Rate (with illegal gene values)
Small Population Size Robustness Test

An early test done to validate the operation of the new representation technique was to use a population size of 2 and verify the GA could still find the target hypothesis. In tests done on GAs using a simple binary representation for the phenotype, small populations almost guaranteed the GA would get stuck on a sub-optimal solution. This also occurred very early in the GA’s processing of the problem and could only be resolved by the mutation operator. The ability of the new representation to deal with this problem shows how it can still quickly navigate through the search space even when it has few candidate hypotheses.

The following graphs show the performance of the GA using the new representation technique with a population size fixed at 2. The problem size was varied from a base 4 four-digit number to a base 32 four-digit number. The Tour percentage doesn’t apply in this test. The 2 individuals in the population are always the parents of the next generation.
One surprise in the test results is that the GA performs even better with population size of 2 compared to earlier tests that had a lower limit of 10. The even-valued gene problem is easily visible in the first graph and the extremes seen for the even-valued gene tests match those of earlier tests with higher populations. Overall there is no degradation caused by small population sizes and the GA shows no signs of getting stuck on sub-optimal.

Conclusions

Although the representation technique is a simple one, it creates a great deal of diversity in the offspring produced by the crossover operator and helps the GA quickly navigate through the search space. The addition of the Cloning operator improves the performance of the GA by as much as a factor of 3 and is a key component in the proposed GA enhancement scheme.

Testing shows the GA stays within the limit of $O(n \ln n)$ for iterations through the crossover operator growth based on input size. This is very similar to BGAs [6].

Additional work could look into the application of this representation on other problems including TSP. The problem chosen for this paper was searching for a target number whose search space was easily controlled by varying the base of the target number. Although this problem was convenient for testing the proposed GA, it isn’t a standard for GA performance testing and other problems should be attempted.
References


