Investigation of New Operators for a Diploid Genetic Algorithm

Şima Etaner UYAR*, A. Emre HARMANCI

Istanbul Technical Univ., Computer Engineering Dept.
Maslak TR 80626 Istanbul, Turkey

ABSTRACT

This study involves diploid genetic algorithms in which a diploid representation of individuals is used. This type of representation allows characteristics that may not be visible in the current population to be preserved in the structure of the individuals and then be expressed in a later generation. Thus it prevents traits that may be useful from being lost. It also helps add diversity to the genetic pool of the populations. In conformance with the diploid representation of individuals, a reproductive scheme which models the meiotic cell division for gamete formation in diploid organisms in nature is employed. A domination strategy is applied for mapping an individual's genotype onto its phenotype. The domination factor of each allele at each location is determined by way of a statistical scan of the population in the previous generation. Classical operators such as cross-over and mutation are also used in the new reproductive routine. The next generation of individuals are chosen via a fitness proportional method from among the parents and the offspring combined. To prevent early convergence and the population overtake of certain individuals over generations, an age counter is added. The effectiveness of this algorithm is shown by comparing it with the simple genetic algorithm using various test functions.

Keywords: Genetic algorithms, evolution, diploidy, meiosis, domination, convergence

1. INTRODUCTION

Genetic Algorithms are a class of stochastic, global optimization algorithms which model the biological principles of Darwin's theory of evolution. This theory centers around the principle of natural selection which mainly states that those individuals that have a certain characteristic which gives them some advantage above the others are more likely to survive and reproduce. If this characteristic is inheritable, then some of these individuals' offspring will be born with it and thus have the advantage over the others. After a few generations, the number of individuals with the favorable trait will increase in the population. So individuals that have an advantage over the others and adapt better to their environments reproduce and leave more offspring for the next generations.

In an optimization problem, the aim is to find the best solution. This may not be possible in all cases, so usually a near best solution is accepted. Biological evolution may be seen as an ongoing global optimization process which keeps on searching for the optimal individual for a certain environment. In that sense, some mechanisms found in nature that lead to this natural global optimization process can be modelled to solve the artificial optimization problems. Genetic algorithms act on this principle. However it is not always necessary to model all the natural mechanisms. While for certain classes of problems, a simple subset of these would work well, for some others, a more complicated algorithm that models more natural operators may lead to better solutions.

1.1. Biological Background

Genetic Algorithms model some of the mechanisms found in nature. So most of the terminology will be borrowed from biology and related fields. Therefore, a brief introduction[2] for these will be given.

Some simple microorganisms (and some stages of life of more advanced organisms) have a *haploid* structure, i.e. they have one set of chromosomes defining all the characteristics. However, most complicated organisms in nature have a *diploid* or *multiploid* structure. In the diploid case, each characteristic is represented by two different genes located on a pair of chromosomes, called *homologues*. *Genes* are the smallest unit of hereditary information and two or more genes that occupy the same location (*locus*) on homologue chromosomes are called *alleles*. All the genes located on all the homologue chromosomes constitute the *genotype* of an individual and the subset of these that are expressed,

*Correspondence: Email: etaner@cs.itu.edu.tr; Telephone: (+90 212) 2856471; Fax: (+90 212) 2853679

i.e. that are visible, is called the *phenotype*. An allele is expressed if it is *dominant*. The only way a *recessive* allele may be expressed is when both alleles have the same value. A *mutation*, which is an abrupt change of a gene from one allelic form to another may or may not change the phenotype. This depends on whether the mutation brings out a dominant allele or not.

Meiotic cell division is the type of cell division used in diploid organisms to produce four haploid, reproductive cells (called *gametes*) from one cell. It consists of two stages with five phases each. In the beginning phase of the first stage, chromosomes replicate, producing two identical *chromatids* (called *sister chromatids)* joined at the center. In the next phase homologue chromosomes, each of which are made up of two identical chromatids, come together to form a *tetrad*. *Cross-over* may occur during this phase between the corresponding segments of non-sister chromatids. At the end of the first stage homologues seperate and two cells are formed. In the following phases of the second stage sister chromatids are also seperated, forming four cells each with half the number of chromosomes as the original cell. This marks the end of the cell division.

After each mating parent has completed its cell division, each will have formed four gametes. One gamete from each parent comes together to form the diploid cell of the offspring.

If an organism has the same value for corresponding genes located on homologue chromosomes, the organism is said to be a *homozygote* for that trait and a *heterozygote* if it has different values. If an organism is homozygote for a certain trait, then that trait is seen in the phenotype as given by the genes but if it is a heterozygote than the allele which is to be expressed is determined depending on which allele is dominant over the other.

Inheritance deals with the transmission of characteristics from parents to offspring. The mechanisms mentioned above enable this transmission.

1.2. The Simple Genetic Algorithm

The basic genetic algorithm as described in Goldberg's book "*Genetic Algorithms in Search, Optimization and Machine Learning*"[1] is a simple but powerful search and optimization tool. It is composed of three main operators: reproduction, cross-over and mutation. Individuals that make up the population have a haploid, binary representation, i.e. each gene value may either be a 1 or a 0. In the beginning step of the algorithm, individuals' genes are randomly initialized to either value and their fitness values are calculated. How the binary string is decoded to form the actual parameters needed to calculate the fitness value depends on the encoding and the problem. For example, if a function has three parameters, each ranging between 0 - 7, these can be represented using the binary representation of each value. Thus, 3 bits (characters) are needed per parameter and the individual's chromosome can be 9 characters long. Each group of 3 characters of this string can seperately be used to represent each parameter. For example, if the chromosome is "*011101100*" and if the parameters, x_1, x_2 and x_3 are as given above, then the first three genes will decode to give $x_1 = 3$ and the second three genes will decode to $x_2 = 5$ and the last three to $x_3 = 4$.

A fitness proportional selection mechanism (a roulette wheel selection) is used to choose the individuals that are going to take part in the reproduction phase. Each individual occupies as much space in the roulette wheel as its fitness. Assuming there are m individuals in the population, the wheel will be turned m times to choose m individuals to reproduce. In this method, the expected number of representatives of each individual is proportional to its fitness, i.e. if f_{Tot} is the sum of the fitnesses of all the individuals and f_i is the fitness of the ith individual, the expected number of times individual i will be selected by the wheel is given by $(m * \frac{f_i}{f_{Tot}})$.

After the selection step, these m individuals are paired off at random for mating. The new individuals are formed by performing cross-over operations with a predefined probability between mating pairs. In the simple genetic algorithm, a one-point cross-over method is used. A cross-over site is chosen randomly and segments beginning at the cross-over site and ending at the end of the chromosome are exchanged, forming two new chromosomes which may or may not be identical to the original mating chromosomes. These new individuals (the offspring) replace their parents, thus there's no overlapping between generations.

As the last operator, mutation occurs with a very low, predefined probability at each gene location, independent of the others. It causes a sudden change in the allele value of the gene location at which it occurs. After this step, the new fitness values are calculated and the loop starts over from the reproduction phase. This loop is repeated until a specified number of generations is reached. The algorithm may be written roughly as in Algorithm 1.

Algorithm 1 Simple GA

```
begin
  Initialize_Population(no_of_indivs);
  do
    reproduce;
    cross-over;
    mutate;
  until end_of_generations;
end.
```

2. DIPLOIDY IN GENETIC ALGORITHMS

In nature, most complex organisms have a diploid chromosome structure. This means that the organism has two genes for each characteristic, located on two chromosomes. Even though this seems like redundant information, it is nature's way of keeping a genetic memory. This way, genetic information which may be useful in the future is not lost but is shielded from the harmful short-term effects of selection. The main mechanism which aids in shielding these characteristics is domination. While the organism has two alleles for the same characteristic, only one of them is expressed. The allele that is dominant over the other appears in the phenotype. However the recessive allele is not lost but is simply masked until a future time when it may become useful. A well known example of this in nature is the *Biston betularia*, the peppered moth.[2] In the 19th century, these moths were very widely seen in England and were mostly found on lichen covered trees and rocks. They had a very light coloring and this made them almost impossible to see against the background. Until 1845, all observed *Biston betularia* have been light colored. However, during this year, one black specimen was seen in growing industrial centers of Manchester. With the increased industrialization going on in England, the tree trunks were left bare and especially in heavily polluted areas, the ground, the tree trunks and even the rocks were almost black. During this period more black specimens of the *Biston betularia* were being found. By the 1950s, only very few light colored ones were seen and these were away from industrial areas. H. B. D. Kettlewell performed tests with both light and dark colored moths in both industrial and unpolluted areas. His tests confirmed his hypothesis that the black color protected the moths from being seen by the birds against the dark backgrounds, thus giving them an advantage over their light colored counterparts. Even though the light color has been dominant, the dark coloring was masked and came out when favorable conditions occurred. Recently, the pollution level in Great Britain has been monitored to show a decrease. The light colored peppered moths have already started to increase in number along with this. This also shows that the light coloring has not been lost either and has started reappearing with the emergence of conditions which favors light colored moths over the black ones.

Diploid representations for genetic algorithms have been discussed and summarized in both Goldberg's[1] and Holland's[3] books. There has also been some more recent implementations and papers introducing additive diploidy and polygenic inheritence,[4] another new approach to dominance and diploidy and its effects on early convergence,[5] multiploidy and dominance,[6] and a *winner take all* strategy for applying dominance.[7]

2.1. The Diploid Algorithm

The implemented algorithm contains most features of the simple genetic algorithm as described above and has some new operators and features added to it. The pseudo-code of the algorithm may be found in Algorithm 2.

2.1.1. The Representation

In the algorithm each individual is represented as in Fig. 1. Chromosome 1 and chromosome 2 are homologues and form the diploid chromosome structure of the individual. The phenotype shows which of the characteristics are expressed. In this implementation, the chromosomes and the phenotype are represented each by strings consisting of either a 1 or a 0 at each location. In Fig. 1 the black boxes represent a 0 and the white ones a 1. How the phenotype is obtained from the two chromosome strings will be explained in the following section. The age of an individual shows for how many generations it has survived. Its fitness value shows how fit the individual is, i.e. how well it adapts to its environment.

Algorithm 2 The Diploid GA

```
begin
  initialize;
  do
    select mating pool;
    form gametes;
    mate;
    mutate;
    for each dead parent, form new individual;
    select next generation;
    calculate new domination values;
  until end_of_generations ;
end.
```

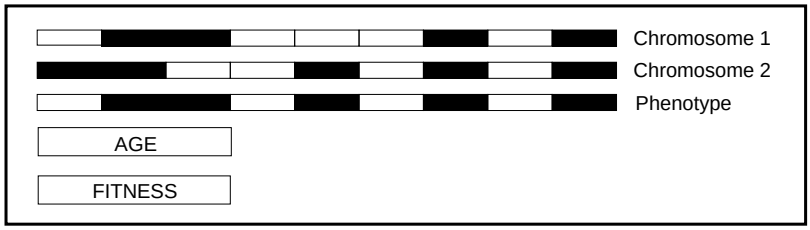

Figure 1. The Individual.

2.1.2. Domination Mechanism

The phenotype of the organism is made up of the characteristics that are expressed and the fitness of an individual is determined by its phenotype. Therefore a mechanism to map the genotype onto the phenotype is needed. This is a very important part of diploid genetic algorithms and there has been some research done most of which are explained in detail by Goldberg.[1]

In this study, the approach used involves a statistical scan of each generation. When determining the phenotype, the genotype elements corresponding to that location may either be equal or different. Using c_{1i} and c_{2i} to represent the ith location on chromosome 1 and chromosome 2 respectively and p_i to represent the corresponding ith location on the phenotype,

if $c_{1i} = 0$ and $c_{2i} = 0$ then $p_i = 0$
if $c_{1i} = 1$ and $c_{2i} = 1$ then $p_i = 1$

In the above cases where the two alleles for the genes on homologue chromosomes are the same, the corresponding phenotype equals that allele but in the case where they are different, i.e. where ($c_{1i} = 0$ and $c_{2i} = 1$) or ($c_{1i} = 1$ and $c_{2i} = 0$), a method to determine the phenotypic value is needed. In natural organisms, the allele to be seen in the phenotype is the dominant one, so an artificial mechanism to simulate this in artificial systems must be designed. In this implementation, a domination array composed of real numbers in $[0.0, 1.0]$ is used. The length of the array is the same as the chromosome length with each value showing the dominance factor of the allele 1 over the allele 0 corresponding to the same location on the chromosomes. For example, if the alleles on the two chromosomes are different for the ith location and if the ith entry in the domination array is $dom_i = 0.8$, the phenotypic value for that location will be 1 with probability 0.8 and 0 with probability 0.2.

The domination array evolves along with the individuals in each population and is calculated using Equation 1.

$$Dom_i = \frac{\sum_j p_{ij} * f_j}{\sum_j f_j}, i = 1, 2, .., length \quad j = 1, 2, ...size \tag{1}$$

where p_{ij} is the phenotypic value of the jth individual at the ith location, f_j is the fitness value of the jth individual, $length$ is the chromosome length and $size$ is the population size (the total number of individuals in the population).

Equation 1 is evaluated at the end of each generation using the phenotype and fitness values of the individuals in that population. So the dom_i value will be higher if individuals with the allele 1 in the ith location have higher fitnesses compared to those that have allele 0. Since the domination array is one of the driving forces of the population, it is expected that the values corresponding to locations on the phenotype that should be 1 in the optimal solution, should approach 1.0 and 0.0 for the case where the optimal value should be 0.

2.1.3. Initialization

The initialization step is similar to the one in the simple genetic algorithm with a few additions. Each of the the genes on the two chromosomes of the individual is initialized randomly to be a 0 or a 1. All the locations on the domination array is initialized to 0.5, meaning that neither allele is dominant in the beginning. After this step, the phenotypes of the individuals are determined using the initial domination array and the fitnesses are calculated based on the phenotypic values. The individuals' age counters are initialized to 0.

2.1.4. Mating Pool Selection

Those individuals that are going to take part in the reproduction phase are selected via a roulette wheel selection mechanism. The method used is exactly the same as the one described in the simple genetic algorithm selection phase.

2.1.5. Gamete Formation

Gametes in natural, diploid organisms are the haploid reproductive cells that go into reproduction. One haploid gamete from each mating pair comes together to make up the diploid cell of the offspring. In most cases in nature, gamete formation is the result of a cell division process called meiosis.

The artificial implementation of the meiotic cell division is given in Algorithm 3 and the mechanism is shown in two major steps in Fig. 2.

Algorithm 3 Artificial Implementation of Meiotic Cell Division

```
procedure meiosis;
  begin
    make chromosome_1 chromatid_1;
    copy chromatid_1 into chromatid_2;
    make chromosome_2 chromatid_3;
    copy chromatid_3 into chromatid_4;
    choose one from chromatid_1 and chromatid_2 randomly;
    choose one from chromatid_3 and chromatid_4 randomly;
    if flip (probability_of_crossover) then
      cross-over chosen chromatids;
    if flip (probability_of_crossover) then
      crossover remaining chromatids;
    make each chromatid a gamete;
  end;
```

In this implementation, a two point cross-over approach is used. In two point cross-over, two cross-over sites are selected randomly and the chromosome segments remaining between these sites are exchanged. How this type of cross-over works is shown in the bottom part of Fig. 2. Unlike in the simple genetic algorithm, cross-over occurs between non-sister chromatids of the same individual during meiotic cell division.

As can be seen in Fig. 2, using this type of cell division adds diversity to the population. Two of the gametes from the four that have been formed are selected randomly to go into the mating phase.

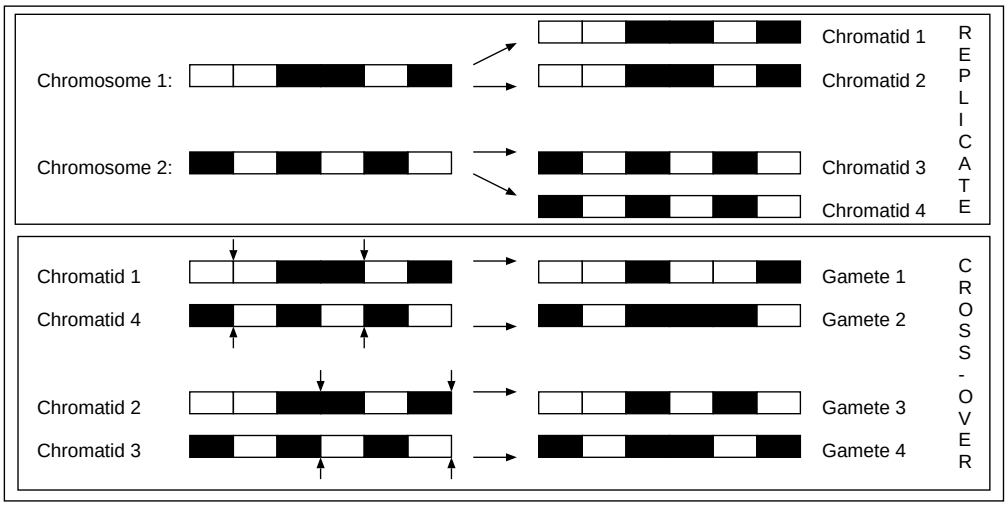

Figure 2. Meiotic cell division.

2.1.6. Mating Phase

After each mating parent goes through a meiosis-like process, there are four gametes from each parent, ready to go into mating. In this implementation, each mating produces two offspring which do not replace their parents at this step. In order to have two offspring, two gametes from each parent are selected at random and each gamete from each parent goes to each one of the offspring. For example, for the case where a binary representation for gametes is used, the formation of the offspring is seen in Fig. 3.

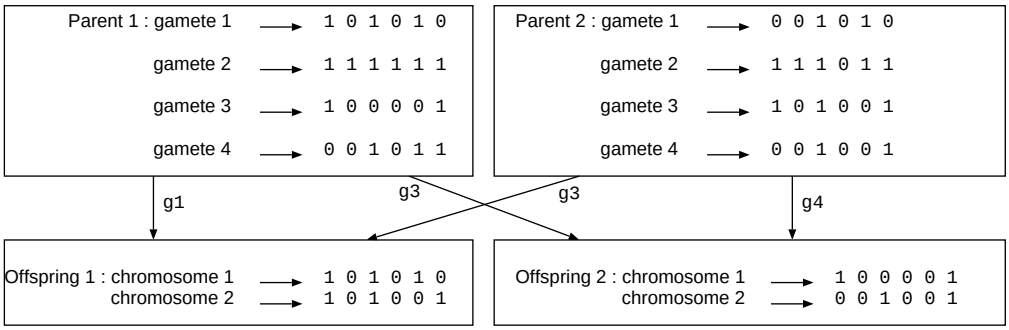

Figure 3. Mating and the formation of offspring.

2.1.7. Mutation

The effect of mutation is the same as it is in the previously explained simple genetic algorithm. In the diploid case the genotype and the phenotype of an individual are different and mutation acts directly on the genotype, i.e. on each gene of the two strands of chromosome. Each mutation at each location is independent of those in other locations and the probability of its occurrance is the same for all genes.

2.1.8. Aging and Death

Since the offspring do not replace their parents and the next generation of individuals is selected from among the parents and the offspring via a fitness proportional selection metod, those individuals that have high fitnesses may get selected through many generations. This may lessen diversity and may cause early convergence. To prevent this, an aging mechanism is introduced. As seen in the structure of the individual in Fig. 1, each individual has an age counter. Each time the individual survives into the next generation, this counter is incremented by one. At the end of each generation, some individuals die of old age and new individuals are initialized randomly to fill their place.

The probability of an individual to die is proportional to the square of its age. The probability of the ith individual to die is calculated as in Eq. 2

$$DeathProb_i = k.age^2 \tag{2}$$

where k is a problem-dependent constant in the real number set $[0.0, 1.0]$ and should be determined at the beginning of the algorithm by taking into account the number of generations the algorithm is to be run, the population size and the maximum age the individuals may live to be. At the end of each generation, each individual is checked to see if it dies or not. If a decision is made to kill an individual, then a new individual is initialized to keep the population size constant.

2.1.9. Determining Who Survives

Unlike in the simple genetic algorithm, the offspring do not replace their parents and they survive together till the end of the generation. Since the population size is kept constant in all generations, half of the individuals must be selected to survive into the next generation. A fitness proportional selection method, similar to a roulette wheel selection method, is used to determine these individuals. However each individual may be selected only once and once it is selected, it is removed from the current population and is copied into the next. The probability of the ith individual being selected as the pth individual to survive may be given as in Eq. 3.

$$SelProb_i = \frac{f_i}{\sum_j f_j - \sum_r f_{sel[r]}}, j = 1, 2, ..., size \quad r = 1, 2, p-1 \tag{3}$$

where f_i is the fitness of the individual, $\sum_j f_j$ denotes the sum of the fitnesses of all the individuals in the population and $\sum_r f_{sel[r]}$ denotes the sum of the fitnesses of all the $(p-1)$ individuals selected before this individual. This selection is repeated as many times as the size of the population.

3. TEST FUNCTIONS AND RESULTS

The proposed diploid algorithm is tested using various test functions and the results are compared with those obtained using an implementation of the simple genetic algorithm on the same function. In the following sections, the results of two of these comparisons will be given. In each test case, the algorithms are run 100 times and the average of the results are given over 100 runs. The algorithms are run each time with the same set of parameters but with different initial populations. The parameters chosen for the algorithms are given in Table 1.

Parameters	Haploid	Diploid
Number of Generations	1000	1000
Population Size	250	250
Cross-Over Probability	0.9	0.9
Mutation Probability	0.009	0.009
Aging and Dying Factor (k)	-	0.001

Table 1. Parameters used in both algorithms

The results will be compared based on the algorithms' online and offline performances. The online and offline performance of an algorithm as defined by DeJong are explained in Goldberg's book.[1] The offline performance is designed to measure convergence and the online performance is designed to measure ongoing performance of the algorithm. In an offline application, a simulation of the system may be used and the algorithm may be run on the simulation to achieve the best results and then these best results can be applied to the real system. However in an online application, the results of function evaluations are the results of actual experimentation on the real system, so in such applications, the time it takes to reach an acceptable solution becomes more important then getting the best solution. DeJong defined the online performance $x_e(s)$ of strategy s on environment e as given in Eq. 4

$$x_e(s) = \frac{1}{T}\sum_{1}^{T} f_e(t) \tag{4}$$

where $f_e(t)$ is the fitness function value for the environment e on trial step t, i.e. online performance is the average of all function evaluations up to and including the current trial. He defined offline performance $x_e^*(s)$ of strategy s on environment e as given in Eq. 5

$$x_e^*(s) = \frac{1}{T}\sum_{1}^{T} f_e^*(t) \tag{5}$$

where $f_e^* = best\{f_e(1), f_e(2), ..., f_e(t)\}$, i.e. the offline performance is the running average of the best performance values up to a particular time.

Another comparison will be made by giving a table of best, worst and average results from each run in 100 runs and the standard deviation of these runs. In each table *Mean Fitness* will denote the average of the best fitnesses achieved over 100 runs of the algorithm, σ will give the standard deviation of the best results obtained from 100 runs, *Best Fitness* and *Worst Fitness* will denote the best and the worst results obtained over all the 100 runs, the *Percentage* will denote the percentage of the standard deviation to the mean fitness value and *Average Step* will denote the average of the number of generations needed to achieve the best result in each of the 100 runs.

3.1. Test Case 1

The function used in this test will be explained below. It is a function which is considered easy for the haploid, simple genetic algorithm. The main aim in this test is to show that the proposed algorithm performs as well as, if not better than, the simple algorithm in such cases.

3.1.1. The Problem

The problem is to maximize the number of genes that have the value 1 in the case where the chromosome length is 32. So the best individual will be the one that has a chromosome with all 32 genes having a value of 1 and the worst one would have all 0s. The fitness value of an individual will be equal to the number of 1s its chromosome has.

3.1.2. The Results

As will be seen in Fig. 4 and Fig. 5, the online and offline performances of both algorithms are plotted on a common set of axis respectively. In each case the x-axis represents the generation count and the y-axis represents a fitness value. In both graphs the above plot line belongs to the diploid algorithm. The plots and the results in Table 2 show that the proposed diploid algorithm performs as well as the simple genetic algorithm which uses a haploid representation.

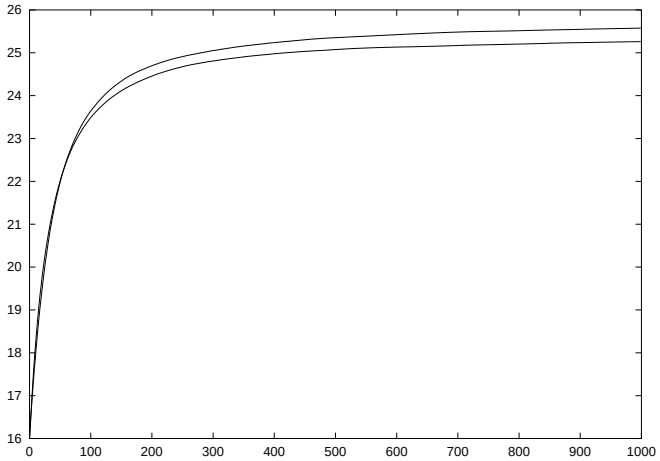

Figure 4. Online performance averaged over 100 runs

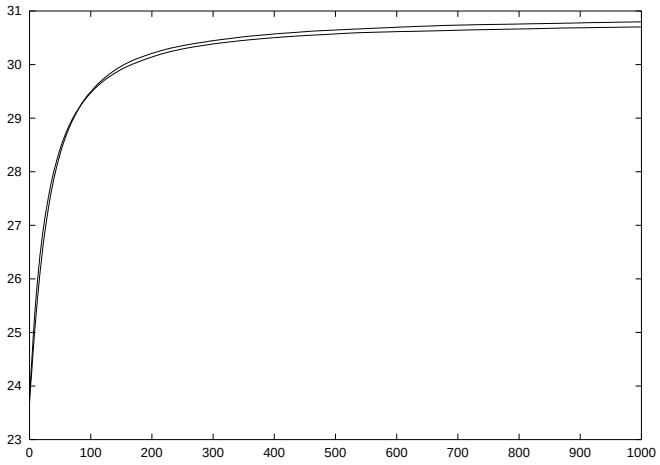

Figure 5. Offline performance averaged over 100 runs

Test 1	Simple, haploid algorithm	Proposed diploid algorithm
Mean Fitness	32	32
σ	0.0	0.0
Percentage	0.0	0.0
Best fitness	32	32
Worst fitness	32	32
Average step	29	30

Table 2. Results from Test 1

3.2. Test Case 2

The function used in this test will be explained below. It is a function which is considered hard for the haploid, simple genetic algorithm because the fitness function changes in time. It will be shown that the proposed algorithm performs better than the simple algorithm as expected.

3.2.1. The Problem

The chromosomes are again made up of 32 genes. The fitness function oscillates every 30 generations between trying to maximize the decimal value represented by the chromosome and trying to minimize it. A 32 bit chromosome string is taken as a binary number and its decimal equivalent is calculated. When trying to maximize this value, the decimal number itself becomes the fitness and when trying to minimize it, the decimal value is subtracted from the biggest decimal number that can be represented by 32 bits and this becomes the fitness value of the individual.

3.2.2. The Results

Fig. 6 shows a plot of the maxima and Fig. 7 shows a plot of the averages of fitnesses of all the individuals obtained at each generation, averaged over 100 runs. For clarity reasons, the x-axis representing the generation count, shows only 250 generations. The y-axis represents fitness values. In both plots, the thicker plot line belongs to the diploid algorithm, while the thinner one belongs to the simple genetic algorithm. It can be seen that the diploid algorithm recovers and finds the maximum much more quickly than the haploid one during fitness function transitions. It takes a little longer for the average fitness value of the population to recover but it still is faster in the diploid case. As expected, these differences are due to the fact that the diploid chromosome structure acts as a genetic memory. Characteristics that were useful before the fitness change were not forgotten and when the fitness evaluation method switched back, they emerged again, causing the algorithm to recover more quickly whereas the haploid algorithm had to search for them all over from scratch.

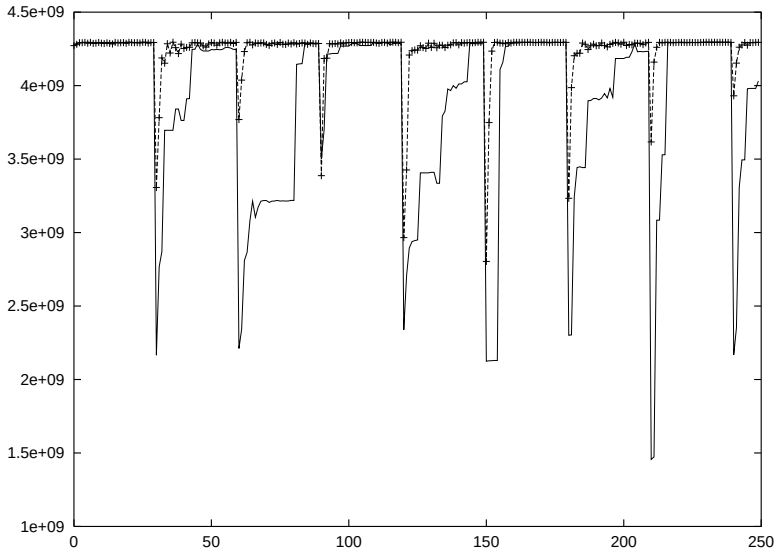

Figure 6. Test 2: Maximums obtained for each generation, averaged over 100 runs

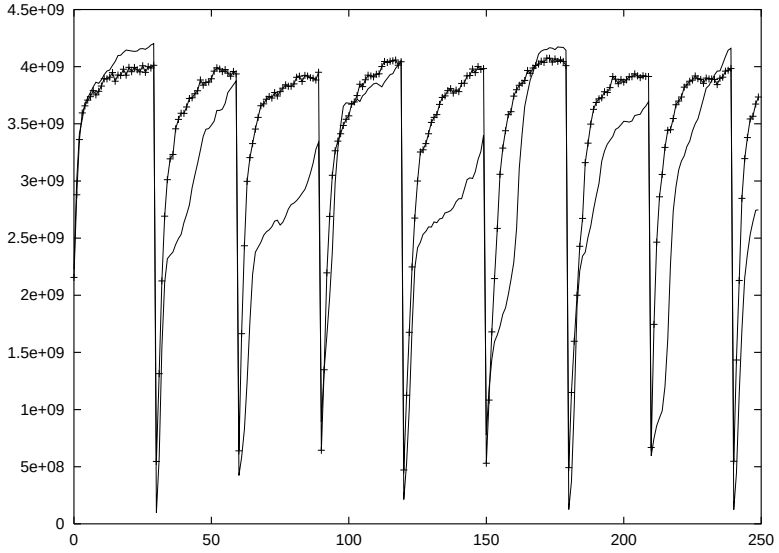

Figure 7. Test 2: Averages obtained for each generation, averaged over 100 runs

As will be seen in Fig. 8 and Fig. 9, the offline and online performances of both algorithms are plotted on a common set of axis respectively. In each case the x-axis represents the generation count and the y-axis represents a fitness value. The better plot line in each belongs to the diploid algorithm. These plots have been calculated from the data in Fig. 6 and Fig. 7 using Eq. 5 and Eq. 4 respectively and are as expected. It must be noted that in these graphs, the results for all 1000 generations are given.

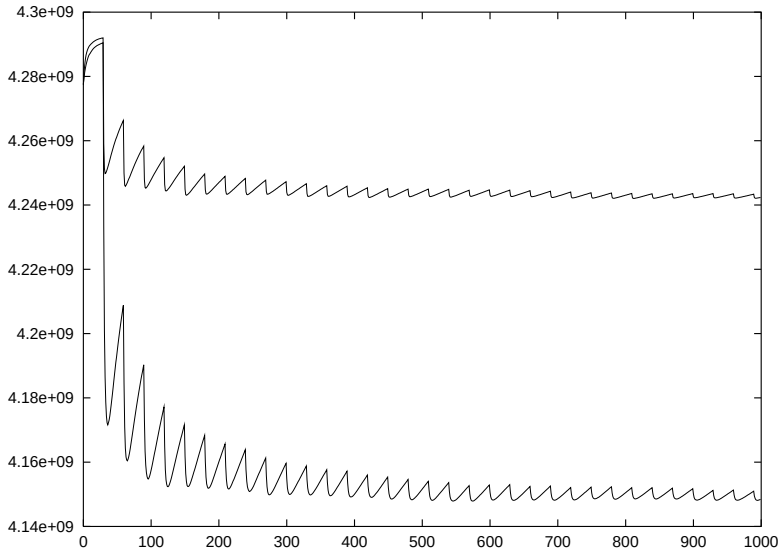

Figure 8. Test 2: Offline performance averaged over 100 runs

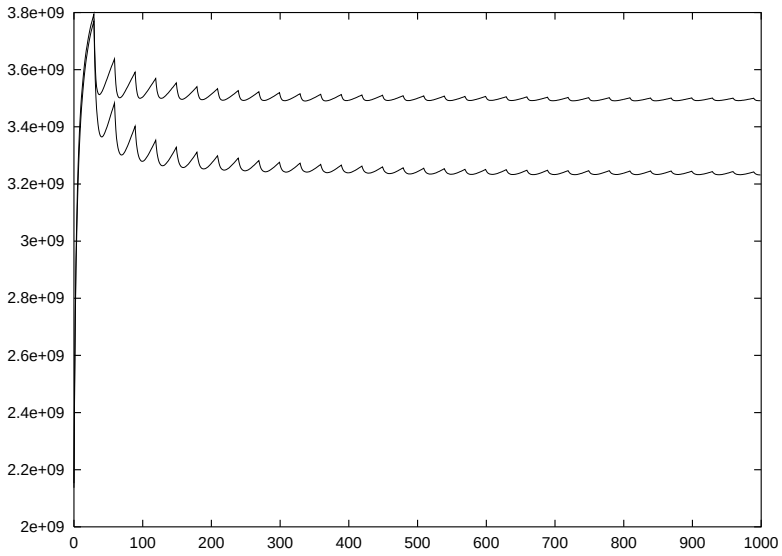

Figure 9. Test 2: Online performance averaged over 100 runs

The results in Table 3 show that the diploid algorithm is much more reliable than the haploid one for this problem. The standard deviation between maximums obtained at different runs is %0.000061 of the mean fitness in the diploid case, while it is %0.000814 in the haploid case. Based on these, it can be said that the diploid algorithm will give good and reliable results even when only one run is made. This also brings out another important aspect of the algorithm. The haploid algorithm has a higher standard deviation between found optima which shows that it is more dependent on the initial population. However the fact that the standard deviation in the diploid case is lower, shows that no matter what the initial population might be, the diploid algorithm produces acceptable results with low deviations between them. So the diploid algorithm is fairly independent of the starting population in this test case .

Test2	Simple haploid algorithm	Proposed diploid algorithm
Mean fitness	4294950144	4294966272
σ	34971.7	2636.2
Percentage	0.000814	0.000061
Best fitness	4294967296	4294967296
Worst fitness	4294702080	4294958080
Average step	147.3	259.5

Table 3. Results from Test 2

In the beginning generations of the second test problem, those individuals that have a higher decimal value are more at an advantage than the others. However after 30 generations, the fitness function changes (similar to an environmental change in nature) and those that have smaller decimal values gain advantage and remain that way for the next 30 generations. The gene combinations which made the individuals favorable in the first 30 generations are not lost however and are remembered in the genotype. When the fitness function changes back to the first case, they become favorable again and thus are once again expressed. This is very much like the *biston betularia* example given in previous sections.

4. CONCLUSION

In this paper, the proposed diploid algorithm has been compared with an implementation of the simple, haploid algorithm using two test functions. The results obtained from these two test cases show that the proposed diploid algorithm works well in both cases and better in the the case of a changing fitness function as expected. Comparisons with other diploid representations found in literature are currently being done.

REFERENCES

1. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
2. H. Curtis, N. S. Barnes, *Invitation to Biology (3rd Ed)*, Worth Publishers Inc., 1981
3. J. H. Holland, *Adaptations in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control and Artificial Intelligence*, MIT Press, 1998.
4. C. Ryan, "The Degree of Oneness", in *Proceedings of the 1994 ECAI Workshop on Genetic Algorithms*, Springer Verlag, 1994.
5. F. Greene, "A Method for Utilizing Diploid/Dominance in Genetic Search", in *Proceedings of the First IEEE Conference on Evolutionary Computation*, 1996.
6. Y. Kim, J. K. Kim, S. Lee, C. Cho, L. Hyung. "Winner Take All Strategy for a Diploid Genetic Algorithm", in *Proceedings of the First Asia-Pacific Conference on Simulated Evolution and Learning*. 1996.
7. E. Collingwood, D. Corne, P. Ross, "Useful Diversity via Multiploidy", *AISB Workshop on Evolutionary Computation,* 1996.