# An Adaptive Domination Map Approach for Multi-Allelic Diploid Genetic Algorithms

**A. Sima UYAR**
Istanbul Technical University
Computer Engineering Department
Istanbul - Turkey
e-mail: etaner@cs.itu.edu.tr

**A. Emre HARMANCI**
Istanbul Technical University
Computer Engineering Department
Istanbul - Turkey
e-mail: harmanci@cs.itu.edu.tr

## Abstract

When working in dynamic environments, adapting to a change in the environment becomes important. One way to deal with such change using genetic algorithms is adding diploidy with a dominance mechanism. Most domination mechanisms found in literature are developed for use with a binary encoding of genes. However in some applications, it provides better performance, ease of implementation and flexibility to represent the genes with more than two alleles. The main aim of this study is to introduce an adaptive domination mechanism for multi-allelic, diploid genetic algorithms that is able to handle different types of change in non-order-based problems. A simulation of a simplified dynamic load balancing of jobs with resource requirements on processing units with capacities is used for testing. Change is introduced by allowing any number of capacities of processing units to change or by allowing the addition or deletion of processing units and jobs to the system. Promising results that promote further study are obtained.

**Keywords:** diploid genetic algorithms, dynamic environments, multi-allelic representations, genotype to phenotype mapping, adaptive domination mechanism.

## 1 INTRODUCTION

When working in dynamic environments, adapting to a change in the environment becomes important. One way to deal with such change using genetic algorithms is adding diploidy with a dominance mechanism. Discussions on diploid genetic algorithms and various applications can be found in (Branke, 2002), (Collingwood, 1996), (Goldberg, 1989), (Greffenstette, 1996), (Hadad, 1997), (Kim, 1996), (Lewis, 1998), (Ng, 1995), (Ryan, 1994), (Smith, 1992).

If a diploid representation of individuals is used, a mechanism for mapping the genotype onto the phenotype is needed. Usually this mechanism is domination. Most domination mechanisms found in literature are developed for use with a binary encoding of genes. However in some applications, it provides better performance, ease of implementation and flexibility to represent the genes with more than two alleles. There has not been much study focused on applying domination in diploid chromosome structures with multi-allelic gene representations.

In literature most papers that deal with non-binary gene representations, mainly focus on genotype to phenotype mapping approaches for order-based problems. One such study (Yoshida, 1994) for solving a TSP problem uses a meiosis like approach for obtaining a haploid chromosome from two chromosomes through recombination operators. Another approach (Yilmaz, 2002) for solving the TSP problem with an order-based, integer representation applies the recombination operators directly on the diploid genotype. Both of these approaches do not incorporate a domination mechanism and are mostly suitable to be used for order-based problems. Another approach which may be used with multi-allelic representations in non-order-based problems for dynamic environments is given by Ryan in (Ryan, 1996) and is based on the natural mechanism of incomplete dominance. Ryan calls this approach the degree of N-*ness* where N is the number of the last possible phenotype. The alleles can take on any form. The number of alleles required to give N different phenotypes is calculated and each allele is assigned a numeric value chosen in such a way as to allow suitable cutoff points to obtain the different phenotypes. A major difficulty may arise in assigning

suitable values to alleles when the number of alleles increase.

## 2 THE PROPOSED DOMINATION APPROACH

In (Uyar, 2002a) by the authors of this paper, a binary representation is used for genes and an approach loosely based on the idea of the penetrance of a gene (Weaver, 1997) is developed for mapping the genotype of an individual onto a phenotype. Instead of having alleles which are strictly either dominant or recessive, levels of penetrance, which will henceforth be called the domination factors for alleles, are defined. In determining the domination factor of an allele, a weighted proportion based on relative fitnesses of individuals having that allele is calculated. The introduction of proportions being weighted according to relative fitness values in a population, artificially incorporates the effects of environmental factors into the calculation of the dominance value of an allele. To represent the domination factors of all the loci, an array, having the same length as the chromosomes and composed of real numbers in the interval `[0.0,1.0]` is used. Each value on the array shows the domination factor of the allele `1` over the allele `0` for the corresponding location on the chromosomes. The domination array changes along with the individuals through generations and is recalculated at the end of each generation using Equation 1.

$$Dom[i] = \frac{\sum_j ph_{ij} * f_j}{\sum_j f_j}, i = 1, 2, .., l \quad j = 1, 2, ...s \quad (1)$$

where $ph_{ij}$ is the phenotypic value of the $j$th individual at the $i$th chromosomal location, $f_j$ is the fitness value of the $j$th individual, $l$ is the chromosome length and $s$ is the number of individuals.

Extending this approach to multiallelic representations, where genes may take on more than two alleles, requires the use of a domination matrix instead of an array. The number of columns in the matrix is determined by the length of the chromosome and the number of rows is determined by the number of possible alleles, i.e. each column corresponds to a locus on the chromosomes and each row corresponds to an allele. Each entry in the matrix shows the domination value of the corresponding allele in relation to the other alleles for that location on the chromosomes. To recalculate the domination matrix at the end of each generation the pseudocode given in Figure 1 is used.

**Algorithm 1** Pseudocode for Domination Matrix Calculation

```
begin
  for i=MinAllele to MaxAllele do
  begin
    for j=1 to LastGenePos do
    begin
      Val=0;
      for n=1 to LastIndividual do
      begin
        if (individual[n].phenotype[j]=i)
          Val=Val+individual[n].fitness;
      end;
      DomMatrix[i,j]=Val/TotalFitness;
    end;
  end;
end;
```

Assume that the population consists of four individuals with the current chromosome, phenotype and fitness values as given in Figure 1.

The variables in the pseudocode in Algorithm 1 are taken as *MinAllele=A*, *MaxAllele*=D, *LastGenePos*=5, *LastIndividual=4* and *TotalFitness=51*. The example domination matrix given in Figure 2 is obtained.

For example, for the population given in Figure 1, the domination value of the allele B for locus 1 is calculated as in Equation 2.

$$DomMatrix[B, 1] = \frac{12 + 8 + 1}{12 + 30 + 8 + 1} = \frac{21}{51} \quad (2)$$

And the domination value of the allele C for locus 1 is calculated as in Equation 3.

$$DomMatrix[C, 1] = \frac{30}{12 + 30 + 8 + 1} = \frac{30}{51} \quad (3)$$

Since the allele A and allele D are not seen at locus 1 on any of the phenotypes of the individuals, their corresponding values for the current example population are 0. This means that in the next generation if an individual has allele B at locus 1 on its first chromosome and allele C at locus 1 on its second chromosome or vice versa, the corresponding phenotype value will become B with probability 21/51=0.41 and C with probability 30/51=0.59. An allele which has a corresponding 0.0 value in the domination matrix is not expressed in the phenotype if the other allele for the same locus has a non-zero domination value. If both alleles have 0.0 domination values, then either one is expressed in the phenotype with equal probability.

```
individual 1:  A B C B D  (chromosome 1)
               B B A D A  (chromosome 2)
               A B C B D  (phenotype)
               fitness = 12

individual 2:  A C B C D  (chromosome 1)
               A C B D B  (chromosome 2)
               A C B C D  (phenotype)
               fitness = 30

individual 3:  B B B A C  (chromosome 1)
               B A C C A  (chromosome 2)
               B B C C C  (phenotype)
               fitness = 8

individual 4:  A B D A C  (chromosome 1)
               B B C C C  (chromosome 2)
               B B C A C  (phenotype)
               fitness = 1
```

Figure 1: Example Population

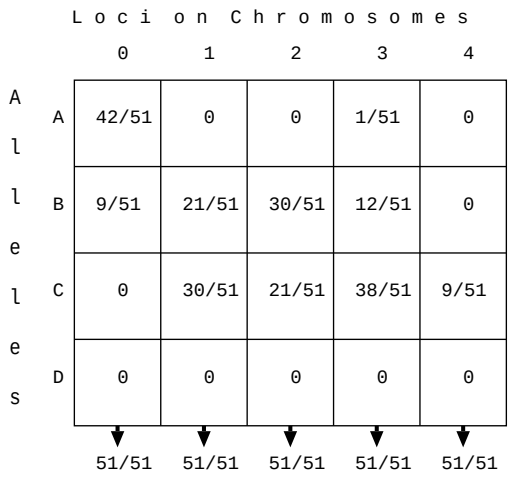


Figure 2: Example Domination Matrix

# 3 THE TEST PROBLEM

A simplified simulation model of a dynamic load balancing of jobs on processing units (PU) is used as a test case problem for the proposed domination approach. The objective is to minimize the total load imbalance throughout the system. The definition of the total load imbalance will be given in section 3.2. This paper focuses on the genotype to phenotype mapping approach developed for these types of problem domains however, a more detailed exploration of the genetic dynamic load balancing approach can be found in (Uyar, 2002b) by the same authors of this paper.

## 3.1 THE SIMULATED SYSTEM

The simulated system consists of a number of PUs fully connected via a network. One processor is dedicated to genetic load balancing operations. Load information from all PUs are sent to the central load balancing processor when a change occurs. This information is used by the genetic load balancer (GLB) to find a better distribution of jobs on the PUs. When a more efficient load distribution is found, all PUs in the system are notified to initiate the necessary job transfers. Job transfers among PUs cause overheads which penalize the performance value for that distribution in proportion to the number of job transfers needed and the sizes of the jobs to be transferred. If the improvement in the system performance through the new job distribution (including the penalty introduced by the cost of job transfers to achieve that distribution) is not above a certain threshold, no job transfers are carried out.

To simplify the simulation, some assumptions are made about the system. These assumptions are:

- All PUs in the system are equipped with the same type of resources with different capacities.
- All jobs may be migrated.
- If a new job arrives that will cause the current system load to exceed the total capacity of the system, the job is denied entry.
- A new job that is allowed entry into the system starts execution on a PU determined randomly.
- There are no constraints. The total load on a PU may exceed its capacity.
- At the beginning of job execution, the average resource (CPU, I/O, Memory) requirements per unit time for each job are determined randomly.

It is assumed that actual resource requirements do not deviate too much from the average values. The load value assigned to the job is a function of average requirements per unit time for all types of resources.

- The migration cost value assigned to a job is a function of the costs of packing and unpacking loads at the host and target PUs respectively and the communication overheads as a result of job transfers over the network from host to target PUs.
- The system is not initially empty and is always at least moderately loaded.

The PUs in the system are represented by their *PU numbers*, *capacities* which are assigned randomly at the time they join the system and a value that shows their *current loads*. A job in the system is represented by its *job number*, its *average resource requirement per unit time* and the *overhead* it brings to migrate that job. In addition to these representational information, the central genetic load balancer unit also keeps track of where each job is located and what each PU's current load is. This information is updated each time a change occurs in the system. The actual system load distribution is updated only if the distribution found by the GLB will increase the system performance by a threshold percent. To prevent thrashing before the GLB converges at least partially, a predefined number of generations are required to pass until the solution candidate found by the GLB is considered for application to the system. The general flow chart for the system execution is given in Figure 3.

### 3.2 FITNESS EVALUATION

The fitness of an individual shows how balanced the load distribution represented by the phenotype of that individual is. To calculate the fitness, the amount of load per unit capacity called the *unit load* (UL) is determined as in Equation 4.

$$UL = \frac{Total\,System\,Load}{Total\,System\,Capacity} \tag{4}$$

For each PU, the amount of load for that PU under ideal conditions, which will be called *ideal load* (IL) is calculated. The ideal load for the *ith* PU which has a capacity of $Capacity_i$ is calculated as in Equation 5.

$$IL_i = UL * Capacity_i \tag{5}$$

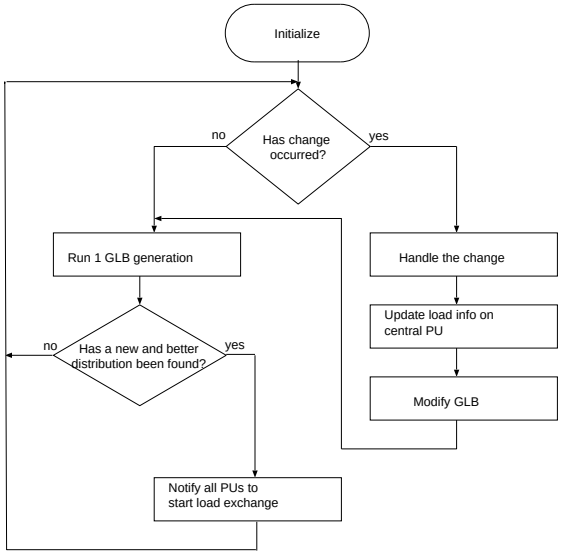


Figure 3: General flow chart for system execution

The *load imbalance* (LI) of a PU is the absolute value of the difference between the *actual load* (AL) of a PU and its *ideal load* (IL). The aim of the load balancer is to minimize the total load imbalance in the system. To normalize the total imbalance value, it is divided by the curent total load in the system. The normalized load imbalance is given by Equation 6.

$$LI_N = \frac{\Sigma_i |AL_i - IL_i|}{Total\,System\,Load} \quad i = 0, 1, ..., NoOfPUs \tag{6}$$

In calculating the fitness of an individual the migration overhead is used as a penalty. The actual fitness value $f_a$ of an individual is calculated as in Equation 7.

$$f_a = LI_N \tag{7}$$

In this implementation the migration costs determined randomly for each job at start of job's execution is a real value in the [0,1] interval. The total penalty value for a distribution candidate is a function of the sum of all the migration costs for the jobs that will be transferred. When calculating the penalty value, in the worst case the new distribution candidate will require all the jobs to be transferred. Assuming there're $n$ jobs in the system, each with maximum possible migration costs (1.0), the upper bound for the migration costs will be $n$. In the best case, no job transfers will be required, giving the lower bound for the migration costs as 0.0. The penalized fitness value $f_p$ for an individual is calculated as in Equation 8 where $C_A$ is the

actual migration costs and $C_M$ is the upper bound on the migration costs.

$$f_p = f_a * \frac{C_M}{C_M - C_A} \tag{8}$$

### 3.3 THE GENETIC LOAD BALANCER ALGORITHM

The GLB algorithm works on an event driven simulation of the defined system. Events occur with interarrival times according to a Poisson arrival process. There are four types of events in the system, namely the *new job event*, the *finished job event*, the *new PU event* and the *removed PU event.*

The basic steps of the diploid genetic algorithm used is given in Algorithm 2 and explained briefly in the following sections.

**Algorithm 2** The Diploid Genetic Algorithm

```
begin
  initialize population;
  initialize domination matrix;
  do
    selection;
    crossing-over;
    mutation;
    domination matrix recalculation;
  until end_of_generations ;
end.
```

In the *initialization* step, each of the genes on the two chromosomes of the individual is initialized randomly to have a value in the possible allele set. All the locations on the domination matrix is initialized to 0.0 to allow equal probability of domination to all possible alleles.

The *reproduction* phase consists of selection of the mating pairs via tournament selection method with tournament sizes chosen as two and a random pairing of the individuals in the mating pool.

Two point *crossing-over* occurs within the genotype of each parent. Each offspring individual receives one chromosome determined randomly from each parent. Offspring replace their parents and there is no overlapping between generations.

The *mutation* operator changes the value of a gene from one allele to another in the set of allowed alleles. This operator acts on the genotype of the individuals.

At the end of each generation, the new *domination map is recalculated* using the approach given in detail in Section 2. This new domination map is used in the next generation to obtain the phenotypes of the individuals from their genotypes.

There are no stopping criteria. The algorithm runs on indefinitely until the whole simulation system is stopped through the issue of an appropriate signal.

## 4 EXPERIMENTS

For a dynamic, single knapsack problem where only the weight constraint changes in time, tests and comparisons were performed and results for the binary representation case with only one knapsack were reported in detail in (Uyar, 2002a) by the same authors. It was shown in that study that the proposed approach for the binary case where the severity of the change and length of the change steps are determined randomly, outperformed Ryan's additive diploidy approach (Ryan, 1994) modified in (Lewis, 1998) to suit dynamic environments. However it was also seen that even the proposed approach failed to give good performance in the test cases where the change was highly severe. It was concluded that these types of change require very high levels of diversity than diploidy alone is able to provide. It can be seen that these conclusions would hold true for the simplified dynamic load balancing of jobs using the proposed GLB and with only weight constraints changed. In the scope of this paper, the more general case where also knapsacks are added or removed will be explored.

In the solution approach to this problem, the chromosome length is again the same as the number of jobs (items) in the system and each gene represents on which PU the corresponding job is located. PUs are identified by integer values. Assuming there are 3 PUs, the allele values may be 1, 2, 3.

Three main types of change are implemented for the testing phase of the proposed domination approach.

- PUs may be added / removed
- Jobs may be added / removed
- Load capacities of individual PUs may change

For the first two types of change where jobs or PUs are added or removed, the change is explicitly made known to the GLB through an event mechanism. However, when a change (increase or decrease) in individual PU capacities occurs, the GLB is not explicitly notified. Ryan's approach (degree of Nness) needs the change to be detected. However the proposed approach adapts automatically through the adaptation incorporated in

the domination mechanism, not requiring to be made aware of this type of change.

Adding or removing PUs to the system means that the number of alleles are changed. Ryan's approach (degree of Nness) is not equipped to deal with this type of change because it requires the re-determination of values assigned to each allele and cutoff values for phenotypic expressions and restart after such a change (Ryan, 1996). When the approach proposed in this study is used, in the case of a change in the number of alleles, an addition or a deletion of a row to the domination matrix is sufficient and the computational costs only increase or decrease linearly.

Adding or removing jobs from the system requires a modification to the representation of the individual. Since each gene location represents a job, a new job causes an increase in the chromosome length and a removed job causes a decrease.

When the load capacities of individual PUs change, it only affects the objective value calculation and the GLB adapts to the change automatically through the adaptive domination mechanism.

The changes in the system are implemented as random events occurring according to a Poisson arrival process. Addition/removal of jobs and PUs cause events to occur while a change in a PU capacity is not implemented as an event. For simplicity and for ease of analyzing the results, only one type of change is allowed at one step. Multiple events occurring simultaneously will not change the behaviour of the algorithm but will make the implementation more complicated and will prevent a clear analysis of results.

The system parameters used for the simulation are given in Table 1. The genetic algorithm parameters used in the experiments are given in Table 2. The individuals are initialized using the default parameter settings for the simulated system.

The types of change, the instances (generation numbers) they occur and the system state after the change are given in Table 3.

Table 1: System Parameters used in Simulation

| | |
|---|---|
| Default Number of Jobs | 32 |
| Default Number of PUs | 5 |
| Maximum Number of Jobs | 1024 |
| Maximum Number of PUs | 128 |
| Maximum Job Load | 540 |
| Maximum PU Capacity | 4096 |
| $\lambda$ for Poisson Distribution | 500 |
| Prob. for New Job Event | 0.64 |
| Prob. for Removed Job Event | 0.16 |
| Prob. for New PU Event | 0.1 |
| Prob. for Removed PU Event | 0.1 |
| Generations Before Update | 50 |
| Acceptable Perf. Improvement | 25% |

Table 2: The Genetic Algorithm Parameters

| | |
|---|---|
| Population Size | 250 |
| Initial Chrom. Length | 32 |
| Crossover Probability | 0.9 |
| Mutation Probability | 0.01 |
| Initial Dom.Values | 0.5 |
| Initial Allele Set | {0,1,2,3,4} |

Table 3: Change Types and Instances

| Gnr. | Change | No.of Jobs | No.of PUs |
|---|---|---|---|
| 0 | Initialize | 32 | 5 |
| 450 | New Job | 33 | 5 |
| 800 | Chg. Cap. | 33 | 5 |
| 1150 | New Job | 34 | 5 |
| 2000 | Rem. PU | 34 | 4 |
| 2450 | Chg. Cap. | 34 | 4 |
| 2900 | New PU | 34 | 5 |

## 5 TEST RESULTS

The program is run 20 times. One representative run for the general behaviour of the system is selected and a plot of the best fitnesses obtained at each generation is given in Figure 4. The x-axis shows the number of generations and the y-axis shows the best and the average current fitnesses for each generation. The values on the y-axis have been obtained by multiplying the actual fitness values by 100 for ease of plotting. The

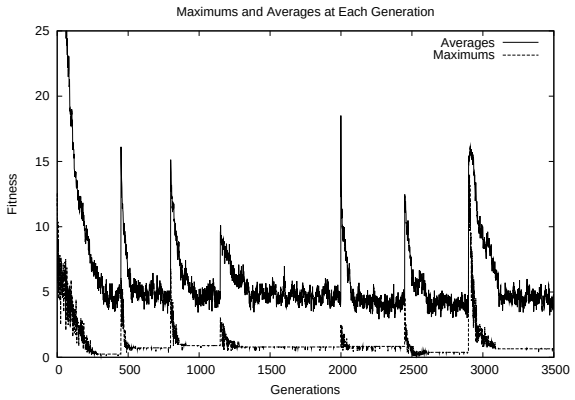


Figure 4: Plot of Current Best and Averages

Table 4: Average Best Fitnesses and Standard Deviations over 20 Runs

| Interval | Avg. of Best Fit. | Std. Dev. |
|---|---|---|
| 0 - 449 | 0.3 | 0.16 |
| 450 - 799 | 0.5 | 0.19 |
| 800 - 1149 | 0.6 | 0.32 |
| 1150 - 1999 | 0.6 | 0.15 |
| 2000 - 2449 | 0.2 | 0.11 |
| 2450 - 2899 | 0.2 | 0.11 |
| 2900 - 3500 | 0.5 | 0.24 |

above plotline in the figure is that of the current averages and the below plotline is for the current best fitnesses.

It is expected that after each change instance, the population will try to move towards a fitness close to 0.0. The objective is to minimize the load imbalance in the system. The lowest possible value for the objective function is 0.0, however due to individual unit capacities and job weights, a perfect 0.0 may not be physically possible. The algorithm tries to find the best possible distribution of the jobs on the PUs. This can be seen from the plot. After a change, the population tries to move towards the minimum. Due to the higher rate of mutation, the population does not fully converge and the average for each population does not fully approach the minimum. This is because the higher mutation rate preserves diversity in the genotype. However the minimums move close to 0.0 much quicker. This is mainly due to the guiding factor of the genotype to phenotype mapping mechanism, speeding up the phenotypic convergence rate.

The averages of best fitness values (multiplied by 100) over 20 runs for each change interval and the corresponding standard deviations are given in Table 4.

The averages of the global best fitnesses for each change interval (all multiplied by 100), averaged over 20 runs, gets very close to zero in a short time (around 50 generations) and the standard deviations obtained from the 20 runs shows that the obtained results are acceptable and independent of the initial population.

## 6 CONCLUSION

The test problem chosen for this study is representative of a domain of non-order based problems where a multi-allelic representation is used and where different types of random change occurs in the environment. The change in the environment can be categorized (Branke, 2002) using the following criteria:

- frequency of change
- severity of change
- predictability of change (is there a pattern?)
- cycle length / cycle accuracy (are there re-visited environmental states?)

All types of change used in this study are not periodic, not predictable, occur randomly with moderate severities and with moderately long change steps.

When working in dynamic environments, for the design of an appropriate evolutionary algorithm, the following aspects (Branke, 2002) should be considered:

- are the occurences of the change explicitly known to the system or do they have to be detected?
- is the genetic representation affected by the change?
- is the change equivalent to a change in the optimization function, the problem instance or some restrictions?

The types of change introduced to the environment for this study were selected to incorporate different aspects of change effecting genetic algorithm design. All of the chosen change types act by changing the problem instance. The objective and evaluation functions stay the same and there are no restrictions. Some of the changes are made explicitly known to the GLB but others occur without a notification mechanism. The GLB does not require any change detection. Some types of change used require a modification of the genetic representation while some don't.

As can be seen from the results of the experiments, when using a diploid representation with an adaptive domination mechanism to handle different types of change in the environment, if a multiallelic representation for individuals is chosen, the proposed approach works well in following the change.

In the first type of change where only knapsack capacities were altered, the dominance scheme allows for automatic adaptation without the need to detect the change. The main factor in this automatic adaptation is the re-calculation of the domination matrix at the end of each generation using the current relative fitness proportions of the individuals.

In the second type of change where the increase or decrease in the number of PUs causes a change in the allele count, it was required only to add or remove a row to the domination matrix. This increases the computational costs linearly. The survey conducted on diploid genetic algorithms for dynamic environments revealed that there are no other domination approaches in literature that addresses this second type of change efficiently with a small amount of modification.

For the third type of change where the number of jobs were increased or decreased, a modification of the individual representation was required. This type of a change requires the change instance to be explicitly known or detected due to the representation modification requirement.

The fact that the proposed approach is able to handle different types of change with acceptable increases in computational costs, makes it suitable for use in problem cases with similar properties to the one used in this study. Further experiments are being carried out on different test problem domains.

## References


Branke J., (2002), *Evolutionary Optimization in Dynamic Environments*, Kluwer Academic Publishers, ISBN: 0-7923-7631-5.

Collingwood E., Corne D., Ross P., (1996), "Useful Diversity via Multiploidy", AISB Workshop on Evolutionary Computation.

Goldberg D. E., (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, ISBN: 0-201-15767-5.

Greene F., (1996), "A Method for Utilizing Diploid/Dominance in Genetic Search", First IEEE Conference on Evolutionary Computation.

Hadad B.S., Eick C.F., (1997), "Supporting Polyploidy in Genetic Algorithms Using Dominance Vectors", in *Proceedings of the Sixth International Conference on Evolutionary Programming*, Vol. 1213 of LNCS Springer Verlag.

Kim Y., Kim J.K., Lee S., Cho C., Hyung L., (1996), "Winner Take All Strategy for a Diploid Genetic Algorithm", First Asia-Pacific Conference on Simulated Evolution and Learning.

Lewis J., Hart E., Graeme R., (1998), "A Comparison of Dominance Mechanisms and Simple Mutation on Non-Stationary Problems" , *in Proceedings of Parallel Problem Solving from Nature 1998*, No. 1498 in LNCS Springer Verlag.

Ng K.P., Wong K.C., (1995), "A New Diploid Scheme and Dominance Change Mechanism for Non-Stationary Function Optimization", Sixth International Conference on Genetic Algorithms.

Ryan C., (1994), "The Degree of Oneness", 1994 ECAI Workshop on Genetic Algorithms, Springer Verlag.

Ryan C., (1996), *Reducing Premature Convergence in Evolutionary Algorithms*, PhD Thesis, National University of Ireland.

Smith R.E., Goldberg D.E., (1992), "Diploidy and Dominance in Artificial Genetic Search", *Complex Systems,* Vol. 6 pp 251-285.

Uyar A. S., Harmanci A. E., (2002a), "Performance Comparisons of Genotype to Phenotype Mapping Schemes for Diploid Representations in Changing Environments", Recent Advances in Soft Computing 2002.

Uyar A. S., Harmanci A. E., (2002b), "Application of an Improved Diploid Genetic Algorithm for Optimizing Performance through Dynamic Load Balancing", *Advances in Simulation, Systems Theory and Systems Engineering*, WSEAS Press Electrical and Computer Engineering Series, pp. 423-428, ISBN: 960-8052-70-X.

Weaver R. F., Hedrick P. W., (1997), *Genetics 3rd. Ed.*, Wm. C. Brown Publishers.

Yilmaz A. S., Wu A. S., (2002), "The Effect of Diploidy on Integer Representations", 2002 Genetic and Evolutionary Computation Conference: Late Breaking Papers, pp. 496-503.

Yoshida K., Adachi N., (1994), "A Diploid Genetic Algorithm for Preserving Population Diversity", Parallel Problem Solving from Nature (PPSN III), pp. 36-45.