A Study of the Fitness Function in the Genetic Algorithms

Aguilar Jose
Dpto. de Computación. Facultad de Ingeniería
Universidad de los Andes. Av. Tulio Febres.
Mérida, Edo. Mérida-Venezuela
Telf: (58.74)440002 Fax:(58.74)402979
email: aguilar@ing.ula.ve

ABSTRACT

Scaling of objective function values has become a widely accepted practice in Genetic Algorithms. This is done to keep appropriate levels of competition throughout a simulation. That is, objective function values must be scaled to prevent takeover of the population by superindividuals. The process of scaling an objective function derives in a fitness function. The fitness function is the only chance that we have to communicate our interest to the Genetic Algorithms. In this paper, we realize a study about the fitness functions in the Genetic Algorithms to solve combinatorial optimization problems.

1. Introduction

The mechanisms of a simple Genetic Algorithm (GA) are surprisingly simple, involving nothing more complex than copying strings and swapping partial strings. The explanation of why this simple process works is much more subtle and powerful. GA starts with an initial random population that represents solutions of the problem. The evolution of this population is, in effect, a blind random search of the search space of the problem. The Darwinian principle of reproduction and survival of the fittest and the genetic operators are used to create a new offspring population of individual from the current population. The reproduction operation involves selecting an individual from the current population based on its fitness and allowing it to survive by copying it into the new population. The individuals are copied into the new population according to their fitness function values.

Since De Jong's studies [2, 3, 6, 7], scaling of objective function values has become a widely accepted practice. Without scaling, early on there is a tendency for that few superindividuals dominate the selection process. To avoid this problem, objective function *(Of)* values must be scaled up to accentuate differences between population members to continue to reward the best performers. The process of scaling an objective function derives in a fitness function *(Ff)*.

In nature, fitness (the number of offspring that survive to reproduction) is a tautology. Large numbers of offspring survive because they are fit, and they are fit because large numbers of offspring survive. Survival in natural population is the ultimate and only taskmaster of any import. By contradistinction, in GA we have the opportunity and perhaps the duty to regulate the level of competition among members of the population to achieve the interim and ultimate algorithm performance we desire. This is precisely what we do when we use the *Ff*. Any and all boundary condition in our *Ff* will be ruthlessly exploited by the individuals in the population.

In this paper, we realize a study about the *Ff* in the GAs to solve combinatorial optimization problems. We present the solution of the graph partitioning problem using different *Ffs*. This work is organized as follows: in section 2, the theoretical basis of the GA is reviewed. Then, we present the problem of the definition of a *Ff* in the GAs. Then, we present the *Ffs* that we will study and evaluate. Section 5 presents the graph partitioning problem. Then, we test and compare the *Ffs*. Remarks concerning the future work and concluding are provided in section 7.

2. The Genetic Algorithms

This is an optimization algorithm based on the principles of evolution in biology. A GA follows an "intelligent evolution" process for individuals based on the utilization of evolution operators such as mutation, inversion, selection and crossover [2, 3, 6, 7]. The idea is to find the best local optimum, starting from a set of initial solutions, by applying the evolution operators to successive solutions so as to generate new and better local optima. The procedure evolves until it remains trapped in a local minimum. The general procedure is the following:

Generation of individuals which represent potential solutions
Repeat until system convergence
  Evaluation of every individual
  Selection of the best individuals for reproduction
  Reproduction of the individuals using the evolution operators
  Replace the worst old individuals by the new individuals

3. The Fitness Function Problem

In GA, each individual in the population is measured in terms of how well it performance in the particular problem environment. This measure is called the fitness measure. Any *Ff* for GAs must be able to evaluate solutions to the problem and produce a score which distinguishes a better solution of another. That is, the *Ff* must associate a measure of quality to every individual.

The computational advantage obtained using GA, with respect to random search, is due to the fact that the search is directed by the *Ff*. The favoring of one solution over another defines the direction that the evolutionary processes in GA encourage individuals in the population to survive. The individuals in the initial generation of the process will generally have exceedingly poor fitness. Nonetheless, some individuals in the initial population will turn out to be somewhat more fit than others. These differences in performance must then be exploited.

A central aspect of fitness function is the concept of *partial credit.* A *Ff* needs to score an individual on how well it performs on the problem to be solved. However, the *Ff* needs to do more than this, it needs to score the individuals in such a way that they can be compared and a more successful individual can be distinguished from a less successful individual. This is a distinction

which is not common in computer science. Usually, we write a program which determines a solution for a given problem, but rarely we are interested in evaluating how well a problem was solved. Another aspect is the *granularity* of the *Ff*. A *Ff* which has only a few values overall may not provide enough information about the problem to enable the evolutionary process to proceed towards the solution.

In addition, there are other factors to consider when designing a *Ff*. One of these factors is how to aggregate the results of *multiple fitness tests*, in those areas where the overall fitness of an individual in the population is determined by its performance on several separate fitness tests. In those cases, each individual in the population is run over a number of different fitness tests. These fitness tests sometimes represent a sampling of different values of an independent variable or a sampling of different initial conditions of a system. Simply adding the results of the individual fitness tests together may not be appropriated, and we need to realize an analyzes about as to combine the results of every fitness test.

In general, the design of a *Ff* depends of the problem that we are studying. For example, the fitness of an individual in an optimization problem may be measured in terms of the absolute value of the difference between the solution proposed by the individual and the optimum solution. In a problem of optimal control, we can define a *Ff* as follows: the fitness of an individual may be the amount of time it takes to bring the system to a desired target state. If one is solving a recognize patterns problem, the fitness of a particular individual may be measure by some combination of the number of instance handled correctly and the number of instance handled incorrectly. On the other hand, if one is trying to find a good randomizer, the fitness of a given individual might be measured by means of entropy satisfaction of the gap test, satisfaction of the run test, or some combination of these factors. For some problems, it may be appropriated to use a multiobjective fitness measure incorporating a combination of factors such as correctness or efficiency.

4. The Fitness Functions

Normally, the *Ff* is the same than the *Of* of the problem. Nevertheless, scaling of *Of* values, to derive in a *Ff*, has become a widely accepted practice. A review of current scaling procedures is presented in Forrest [2, 3, 6, 7]. In this section, we present different *Ffs*.

a. The Ff is the same than the Of:

In this case, we use the *Of* values as the *Ff* of every individual:

$$Ff(i) = Of(i) \qquad \forall\, i \in H \qquad (1)$$

Where, H: set of individuals in the current population

b. Linear Scaling

One useful scaling procedure is linear scaling. Linear scaling requires a linear relationship between the *Of* and the *Ff*

$$Ff(i) = a\, Of(i) + b \qquad \forall\, i \in H \qquad (2)$$

The coefficients *a* and *b* are chosen to enforce equality of the Of_{av} and Ff_{av} values. In this way, this condition ensures that the average population members receive one offspring copy on average and the best receive the specified multiple number of copies. To control the number of offspring given to the population member with maximum fitness, we choose the other scaling relationship to obtain a scaled fitness

$$Ff_{max} = c\, Of_{av}$$

where c is the number of expected copies desired for the best population member.

Using these conditions we can calculate linear scaling coefficients *a* and *b*. This simple scaling helps prevent the early domination of extraordinary individuals, encouraging a healthy competition among near equals.

c. Study of the Solution Space in the Current Population

In this case, we define the *Ff* depending of the values of the *Ofs* in the current population. If these values are very different we try to prevent the domination of superindividuals in defining the fitness of an individual in terms of the absolute value of the difference between the *Of* and the Of_{av} value, otherwise, we try to find quickly the local optimal using superindividuals.

$$Ff(i) = \begin{cases} |\, Of(i)) - Of_{av} \,| & \text{if } (\max_{i\in H}(Of(j)) - \min_{i\in H}(Of(j))) \text{ is very large} \\ \{Of(i) - \min_{i\in H}(Of(j))\} & \text{otherwise} \end{cases} \quad (3)$$

We propose to use these fitness functions with a roulette wheel approach to selection the individuals for reproduction. In this case, we create a weighted roulette wheel where each current individual in the population has a roulette wheel slot sized in proportion to its fitness. To reproduce, we simply spin the roulette wheel thus defined the same number of time than the number of individuals in the current generation. That is, each time we require an offspring a simple spin of the roulette yields the reproduction candidate. In this way, more highly fit individuals have a higher number of offspring in the succeeding generation. Once an individual has been selected for reproduction, a replica of the individual is made. This individual is then entered into a mating pool for further genetic operators action (for generate the new population).

5. An example: the Graph Partitioning Problem

The problem consists in dividing a graph in several subgraphs, so as to minimize a given *Of*. The graphs are sets of nodes joined by arcs. In a very general way, to place the problem on a mathematical formulation, the following definition is necessary:

$\prod=(N,A)$ where,

$\prod$ is a directed graph,

N is a set of n nodes on which we can associate a weight function Q : N -> R. In our study Q(i)=1 $\forall$ i=1,.., n,

$A = a_{ij}$, are node pairs that define the arcs. It's known as adjacency matrix, and it defines the arc weight of $\prod$. In our study $a_{ij} = 1 \ \forall \ i,j = 1, ..., n$

The problem consists in dividing the graph in *K* different subgraphs $\prod=\{\prod_1,..., \prod_K\}$, according to certain constraints. The classic constraints are:

- The number of nodes of the different subgraphs ($N_{\prod_k} \ \forall \ k = 1, ..., K$) must be balanced.

- The arcs with extremities in different subgraphs must be minimal.

The *Of* associates a real value to every subgraph configuration. We propose the *Of* following:

$$F = \sum_{i,j \in D} a_{ij} + b \left(\sum_{k=1}^{K} \left(N_{\Pi_k} - n/K \right)^2 \right) / K \qquad (4)$$

where $D=\{i \in \prod_k \; \& \; j \in \prod_l \; \& \; l \neq k\}$

The balance factor (b) defines the importance of the interconnection cost with respect to imbalance cost, and. The graph partitioning problem is reduced to find a subgraph configuration with minimum value for the *Of*:

$$F1 = MIN(F)$$

5.1 Resolution with Genetic Algorithms

The GA applied in our problem follows the next procedure: we define a space of research of *n* vectors where everyone represents an individual, and every individual represents a possible solution. Each vector has *n* elements and every element has a value among *1...K*, according to the subgraph to which it belongs. Furthermore, we use the equation (4) to determine the cost of every individual. We begin with an initial population of individuals randomly defined and we choose the individuals with minimal cost for generating new individuals using the genetic operators. Since the population is constant, we substitute the worst individuals of the current population by the best individuals generated. The procedure stops if we exceed a given number of generations without finding a better solution. To explain the genetic operators that we use, we consider the next vectors:

vector A: 1 3 2 2 vector B: 3 2 1 1

For the vector A, the notation means that node 1 belongs to subgraph 1, node 2 to subgraph 3, etc. Similarly for the vector B. The application of the *crossover* operator is the combination between two vectors. The points which cut the individuals are randomly chosen. For example, the application over the previous vectors gives:

1 3 : 1 1 and 3 2 : 2 2

The *mutation* operator is the random change of a part of a vector. The points that determine the part in which the mutation will be done are randomly chosen. For example, the application over the vector A gives:

1 : 2 1 : 2

In this work, we used the crossover operator and then the mutation operator according to the PM probability, that is PM is the probability of use the mutation operator after the crossover operator.

6. Performance Evaluation

The random graphs used are defined for the average number of nodes (*n*) and the average degree of the successor nodes of a node (*d*). For each graph, the successors of a node are chosen randomly from a uniform distribution in the interval [1, d]. The execution time is in seconds. We compare these results with the best method of the work [5, 8, 10], that is Simulated Annealing (SA). We have used a SUN SPARCstation II.

The parameters of the simulations are the following: the total number of subgraphs (K), the mean number of nodes per graph (n), the mean number of successors per node (d) and the balance factor (b). The performance criteria studied are: execution time of the heuristics and *Of* value of the solutions. We generate 50 random graphs for the set of parameters where n = {10, 20}, K=2 and d=2, and 20 random graphs for the set of parameters where n = {10, 20, 50, 100}, K=5 and d=5.

The number of simulations depends on the accuracy demanded for the confidence intervals (σ) of the estimation of the *Of* (4). The number of simulations per a given set of parameters is either (depending of which occurs first): 30 simulations or the number of simulation required to obtain a given standard deviation of the *Of*. Due to space limitations, the results presented in this section were chosen because they are representative of the phenomena studied. We fix $\sigma = 0.1$ for large graphs, and $\sigma = 0.05$ for graphs of little size.

We obtain the optimum solutions for graphs where $n \leq 20$ and $K < 5$. With these results, we study other performance criteria for graphs of little size: the average performance (Sop_l), the percentage of optimum solutions (δ_l) and the relative error (E_l) of each heuristic, where *l* is the

number of times that we execute the heuristic to obtain these values. These criteria are calculated as follows:

- Sop_l is the mean value of the solutions for a given set of parameters, for (l= 1, ..., 10).
- δ_l is the percentage of optimum solutions of a heuristic, for (l= 1, ..., 10).
- E_l is the mean relative error of the solutions of a heuristic compared to the optimum solution,

$$E_l=\Sigma_{i,l} (S_{il} - S_i^{opt}) / S_i^{opt} \quad \text{for } l=1, ...,10 \text{ and } i=1,...,50$$

where S_i^{opt} is the optimum solution of the graph *i* and S_{il} is the solution of the heuristic for the graph *i*.

The heuristics using the equations 2 and 3, for graphs of *200* or more nodes, need a very large time to reach the suboptimal solutions (Figure 5.2), but they give the best results (Figure 5.1). For graph of little size (of *100* or fewer nodes), the difference between the results of the heuristics is little (Figure 5.1). Otherwise, the heuristic using the equation 2 gives the best results. The large execution time for the heuristics using the equation 2 is due to that we must make complex calculation to obtain the value of the *Ff* for every individual. For the equation 3, the heuristics need a lot of time to converge because they try to combine superindividuals with bad solutions to study all solution space.

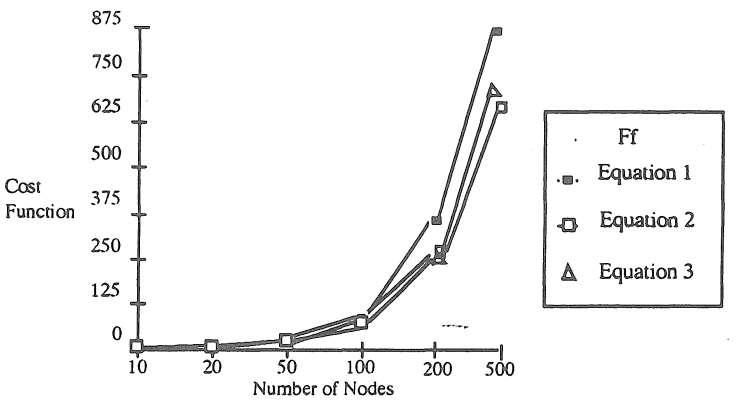

Figure 5.1. Simulation Results for
K= 2, b = 1 and d = 2

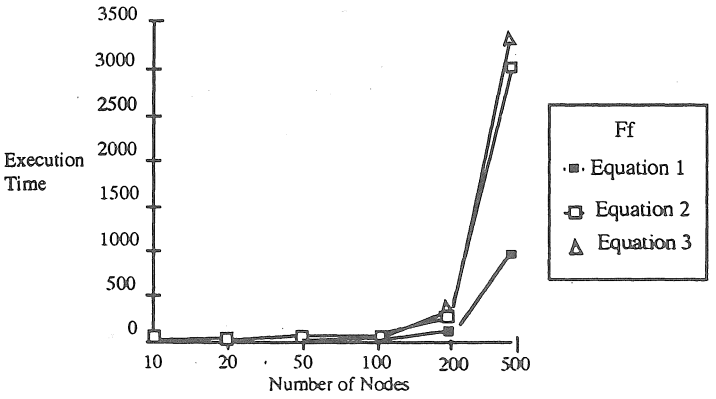

Figure 5.2 Execution Times of the heuristics
for K= 2, b = 1 and d = 2

Sop and δ are approximately the same for every heuristic, but the heuristics using the equation 1 give the worst results (Table 5.1). The heuristics using the equations 2 and 3 are the heuristics with the least mean relative error. In general, we obtain good results. The equation 1 is inefficient because it is going to reproduce almost all time the best individuals.

Ff and SA	E_5	δ_5	So_5	E_{10}	δ_{10}	So_{10}
Equation 1	0.52	0.5	3.55	0.42	0.7	3.25
Equation 2	0.30	0.8	2.8	0.20	0.9	2.50
Equation 3	0.30	0.8	2.75	0.23	0.9	2.65
SA	0.35	0.8	2.8	0.26	0.9	2.50

Table 5.1. Others performance criteria
for n=20, d=2, b=1 and K=2

7. Conclusions

In the GAs, the regulation of the number of copies is especially important in small population. At the start of the GAs run it is common to have a few extraordinary individuals in a population of mediocre colleagues. If we select the best, the extraordinary individuals would take over a significant proportion of the finite population in a single generation. This is undesirable and a leading cause of premature convergence. On the other hand, in a run the population average fitness may be close to the population best fitness. If this situation is left alone, average members and best members get nearly the same number of copies in future generations, and the survival of the fittest necessary for improvement becomes a random walk among the mediocre. In both cases, at the beginning of the run and as the run matures, fitness scaling can help.

In our study all the *Ffs* give good results, but the GAs using the equations 2 and 3 give the best results because they take both cases into account. The execution times for these heuristics are large because we realize complex calculation to obtain the *Ff* values for every individual. We remark than the quality of the fitness functions will depend of the selection technique that we use. We have proposed to use a roulette wheel approach as selection mechanism. Future research will apply these *Ffs* in other combinatorial optimization problems, and will implement the GA on parallel machines.

Reference

1. KERNIGHAN, B. and LIN, S. "An efficient heuristic procedure for partitioning graphs", The Bell System Technical Journal, February 1970.
2. MUHLENBEIN, H., SCHLEUTTER, G. and KRAMM, D. "Evolution algorithms in combinatorial optimization", Parallel Computing, Vol 7, No 2, pp 65, 1988.
3. GOLDBERG, D. "Genetic algorithms in search, optimization and machine learning", Addison-Wesley, 1989.
4. TALBI, E. and BESSIERE, P. "Un algorithme génétique massivement parallèle pour le problème de partitionement de graphes". Rapport de recherche. Laboratoire de Génie Informatique. Grenoble-France. 1991.
5. AGUILAR, J. "Combinatorial Optimization Methods. A study of graph partitioning problem", Proceedings of the Panamerican Workshop on Applied and Computational Mathematics, PWACM, Caracas, Venezuela, 1993.

6. KOZA, J. "Introduction to Genetic Programming", Advances in Genetic Programing (K. Kinnear Jr editor), The MIT Press, 1994.
7. KINNEAR, K. "Alternatives in Automatic Function Definition: A comparison of performance", Advances in Genetic Programing (K. Kinnear Jr editor), The MIT Press, 1994.
8. AGUILAR, J. "L'allocation de tâches, l'équilibrage de la charge et l'optimisation combinatoire", PhD thesis. Rene Descartes University, Paris, France, 1995.
9. AGUILAR, J. "Evolutionary Learning on Recurrent Random Neural Network", Proceedings of the World Congress on Neural Networks, Washington, D.C., USA, 1995.
10. AGUILAR, J. "A Hybrid Approach based on Genetic Algorithms and Simulated Annealing for Combinatorial Optimization Problems", Proceedings of the XXI Latin American Conference on Informatics-PANEL 95, Porto Alegre, Brazil, 1995.