Pipelined Genetic Architecture with Fitness on the Fly

F. Sáenz
Dp. Sistemas Informáticos y Programación
Universidad Complutense Madrid
Av Complutense s/n, E-28040 Madrid, Spain
fernan@eucmax.sim.ucm.es

A. Ibarra, J. Lanchares, J. I. Hidalgo
Dp. Arquitectura de Computadores y Automática
Universidad Complutense Madrid
Av Complutense s/n, E-28040 Madrid, Spain
ibaiba@dacya.ucm.es

Abstract

One of the main bottlenecks in Genetics Algorithms is the fitness evaluation for each individual. In this work, we propose a new fitness evaluation method, which will solve the bottleneck calculating a fitness on the fly.

1. Introduction

Finding a solution in the search space is a complex task when the number of variables and cardinalities of domains is large. Several heuristic techniques have been used as simulated annealing, evolution programs [4], tabu search, GRASP (Greedy Randomized Adaptive Search Procedure) and neural networks.

Genetic algorithms [2] are stochastic algorithms implementing a search procedure which model a biological phenomenon: genetic inheritance, and Darwinian survival.

In this work, we enhance the performance of genetic algorithms, making parallel evaluation of genetic operators through pipelining. For this goal, we firstly propose a novel pipelined genetic architecture to implement genetic algorithms which behaves differently from a usual sequential algorithm [3]. Thus, we implement the architecture and compare its efficiency with the sequential one, focusing on the total generations needed to obtain solutions.

2. Genetic Algorithms Background

Many interesting problems do not have reasonably fast algorithms to solve them. Most belong to NP-hard combinatorial optimisation problems, which are NP-complete decision problems requiring an algorithm of exponential complexity with problem size. Genetic Algoritms (GAs) technique is a powerful programming tool that try to solve this kind of problems.

Modelling a problem with GA requires two steps: firstly the coding of solutions, i.e. coding chromosomes in such a way that each one may represent a solution. Secondly the design a cost function, which measures the fitness of the individual, i.e. how good the solution is under a given criterion.

The fitness function is the link between the Genetic algorithms and the problem to solve. A fitness function takes a chromosome as input and returns a measure of its goodness.

GAs focus on local optimisation, but evolution allows them to jump to different points of the search space. According to evolution, best individuals are likely to survive and generate offspring, therefore transmitting their best features to new generations. In this procedure there are three basic genetic operators, involved in GA: *Selection*, *Crossover*, *Mutation* and the *Fitness* function.

These operations have different ways to be implemented, considering different criteria and different parameters. In the following list they are briefly discussed:

- *Selection.* This is a weighted random selection of two or more individuals whose genetic material will be propagated to the next generation. The weight induces the probability for best chromosomes to be selected.

- *Crossover.* From selected chromosomes, their genetic material is combined for generating the new ones. Typically, two chromosomes are mated. There are several ways to perform crossover.

- *Mutation.* After generating offspring, there is a random selection of new chromosomes for changing particular genes. This operation is quite important in the evolution process, because it allows GAs to explore other points in the search space, escaping from local optima. Obviously, it must be adjusted in order to avoid a pure random search.

- *Fitness.* Each chromosome is characterised by its fitness, which is problem dependent. There are no GA parameters related to it.

0-7695-1239-9/01 $10.00 © 2001 IEEE

GAs have been proven useful to tackle complex optimisation problems, even easily when coding is natural from the problem posed. Nonetheless, as any other stochastic procedure, there is no way of knowing whether optimum has been computed. Moreover, there is no clear stopping criterion for termination. However, when other approaches cannot be applied because of either restrictions (e.g., computation time in real time applications) or they are expensive (e.g., no-need of best solutions, but for reasonable ones), GAs are fully justified.

3. Pipelined Genetic Architecture

In this Section, we present the design of a pipelined genetic architecture. For other related parallel models, it has been shown and analysed the high speedup of hardware implementations when compared with software implementations. In this work, we propose a new approach which allows a better use of the pipeline.

3.1. Identifying Dependencies

Figure1 shows the diagram flow of our genetic algorithm.

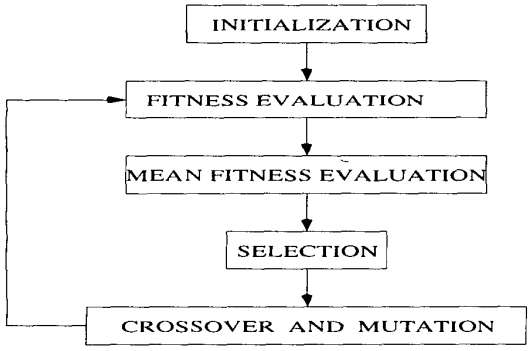

Figure 1. Structure of a Generic Genetic Algorithm.

As we can observe in our sequential genetic algorithm (Figure(1)), we can split our tasks into several parallel pipelines. Each of this threads can select an individual and continue with the standard GA sequence, i.e., one of the genetic operators (mutation or crossover) takes place. Then, we have a new chromosome with a new fitness value.

The next step will be evaluate the mean fitness for the complete population. But, we can only do that when we have evaluated the fitness for all chromosomes. This is our bottleneck, the computation of the mean fitness. During the next sections we explain how we are solve this problem.

In the next Section, we deal with the goal of eliminating these two bottlenecks.

3.2. Overcoming Dependencies Problems

Selection operation has been observed to be the main bottleneck in the pipelined architecture. Since selection relies on mean fitness computation, we propose to compute it on the fly, that is, computing an incremental mean.

In order to overcome the wait for chromosome fitness availability, the mean can be computed incrementally (on the fly) by feeding new fitness values to mean fitness computation. Therefore, there is always an available value for $\overline{f}$, which is updated whenever a new selection have to be performed. In such a way, the pipeline must not wait for mean fitness computation. The incremental mean is defined inductively as follows:

$$\begin{aligned} \overline{f}_0 &= \overline{f} \\ \overline{f}_j &= \tfrac{1}{j}\left(\overline{f}_{j-1} \cdot (j-1) + f_{i_j}\right) \end{aligned} \tag{1}$$

where $\overline{f}$ is the mean fitness computed initially, f_{i_j} is the chromosome fitness (from f_1 to f_n, $i_j \in 1, ..., n$) already feeded, and j, the number of chromosome fitness feeded.

This approach has two drawbacks:

- The number of fitness values previously feeded must be kept. This implies to have enough room available for storing.
- Since generations are not synchronous anymore, there is no clear notion of generation fitness. That is, should all f_{i_j} be equally weighted as Equation 1 implies?

In this work, we compute the mean on the fly, avoiding the storing of j, and we use an empirically obtained weight for each f_{i_j}, so that the pipeline has a continuos data flow, only limited by the performance of each pipeline stage. The architecture which supports this idea is depicted in Figure(2).

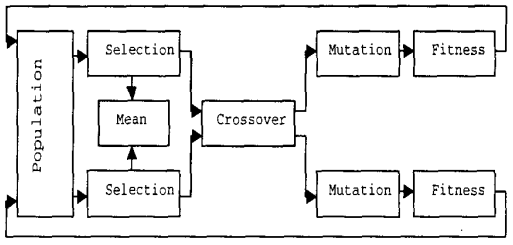

Figure 2. Architecture of the pipelined GA.

This approach has an additional advantage: there is no need of sorting chromosomes, as another implementations need [1] in order to select best chromosomes to be passed to the crossover module. This task is accomplished in our model by comparing the chromosome fitness to the current mean, so that selection acts as a filter for worst chromosomes.

3.3. Implementation of the Pipeline

The modified GA for embodying the pipeline consists of the following: firstly, the initial mean fitness is computed as the standard GA does; then, the pipeline itself comes, which consists of selection, crossover, and mutation operation, which are covered in forthcoming sections.

3.4. Genetic Operators

In this section we are going to describe the three basic genetic operators, with special attention to **Selection** operator. We have to notice that the selection is the crucial point in our pipelined strucure. The two other operator **Crossover** and **Mutation** are briefly described.

From a given population, all chromosomes are quasi-randomly selected (i.e., paired off) for crossover, mutation, or rejection. We have adopted the following criterion: If both chromosomes are good enough, they are selected for crossover. If both chromosomes are not good enough, they are selected for mutation. If only one of them is good, then it is replicated once and the two clones are selected for mutation.

Goodness of chromosomes $g(X_i, j)$ is determined by an empirically obtained parameter, the selection threshold st, which indicates the lower bound for the ratio of chromosome fitness, and the mean fitness. This parameter must increase as better solutions are found. The increasing is guided by another parameter, the increase of selection threshold Δst, which is also obtained empirically. The goodness is defined by:

$$g(X_i, j) = \begin{cases} true, & if \frac{f_i}{\overline{f}} \geq st(j) \\ false, & if \frac{f_i}{\overline{f}} < st(j) \end{cases}$$

The goodness is dependent not only on fitness, but also on "time" (here denoted by j)[1], since hopefully, as time passes by, better solutions are found and, therefore, one must be more strict in selecting chromosomes. Therefore, st is tuned correspondingly with Δst. This tuning is defined as:

[1] In fact, j denotes here a cpo (complete partial order) between generations of particular pairs of chromosomes. The sequence $j = 1, 2, 3, ...$ identifies the order in which chromosomes are selected, i.e., $X_{i_1}, X_{i_2}, X_{i_3}, ...; i_j \in 1, ..., n$.

$$st(j) = \begin{cases} st(j-1), \\ \quad if wmf(j) < wmf(j-1) \\ st(j-1) + \Delta st, \\ \quad if wmf(j) \geq wmf(j-1) \end{cases}$$

where, $wmf(j)$ is the weighted mean fitness at time (See below).

We discard the incremental mean approach, we compute a weighted mean with the value of chromosome fitness. The criterion for computing the weighted mean fitness is the following: If chromosome fitness is greater than the current weighted mean, then it must influence notably, and, conversely, if it is not, then it must slightly influence the current weighted mean. This slight influence is necessary in order to avoid to have chromosomes never selected because the current weighted mean fitness is too high. Both influence degrees are determined by empirically obtained parameters (n_i and s_i, respectively). The computation of the weighted mean fitness at time $j(wmf(j))$ is defined as:

$$wmf(0) = \overline{f}$$

$$wmf(j) = \begin{cases} f_i \cdot n_i + wmf(j-1) \cdot (1 - n_i), \\ \quad if f_i \geq wmf(j-1) \\ f_i \cdot s_i + wmf(j-1) \cdot (1 - s_i), \\ \quad if f_i < wmf(j-1) \end{cases}$$

The **Crossover** operator is carried out from a classical point of view. A parameter determines the probability of crossover. If two chromosomes have to be breed, then a gene is randomly selected for splitting each chromosome and exchange the resulting partitions.

The **Mutation** operator may occur when the chromosome must be mutated (decided in the selection stage), or when it is randomly determined in terms of a mutation parameter.

4. Performance Analysis of the Pipelined Genetic Architecture

This Section is devoted to the performance analysis of the proposed pipelined genetic architecture (hereinafter PGA) in terms of its adequacy in finding solutions. We are going to crosscheck results obtained with our genetic algorithm with the results obtained with the basic Holland's genetic algorithms, we therefore compare both relating quality of solutions and number of generations needed to reach them. We consider the classical Holland's genetic algorithm (hereinafter HGA) characterised by sequential behavior.

4.1. Comparing PGA against HGA

We have considered three benchmarks (Hill Climbing, Prisoner's Dilemma, and Task Problem, although only Pris-

oner's Dilemma results are presents), with three sets of seeds, and three population sizes, which results in twenty seven different runs for each genetic algorithm instance (HGA and PGA). All the runs have two hundred generations long, and the string length for the chromosome is sixteen bits. The selection threshold, is 1.0, and the increase for this factor is 0.001. Crossover probability is 60%, and mutation probability is 2%.

Prisoner's Dilemma has one global maximum and several local maxima. Progression to the optimum is therefore slower. For the sake of conciseness, we have shown in the following graphs the mean value of the three runs corresponding to the three different sets of seeds.

Figure(3) to Figure(5) correspond to the Prisoner's Dilemma benchmark.

We can observe from graphs above that PGA behaves clearly better than HGA. Convergence speed increases notably when the population size is augmented from 16 to 32 chromosomes. It also increases (in less degree) when we augment it from 32 to 64.

5. Conclusions

In this work, we have developed a general novel pipelined architecture, which shows important speed-ups in early simulation experiments.

Results have been acquainted from C and VHDL analysis implementations. The former give us qualitative data in terms of number of generations needed for achieving a solution, whereas the latter, in addition, can be used as a platform to investigate the implementation of critical regions of the parallel system in programmable logic devices, such as FPGAs. Moreover, by embodying the needed (and approximated) delays we can carry out a high-level simulation regarding speed-up in terms of simulation time. This could be an analysis step in comparing implementations of the parallel system in a multiprocessor architecture and workstation networks.

References

[1] I. M. Bland and G. M. Megson. *Implementing a Generic Systolic Array for Genetic Algorithms.* Dpt. of Computer Science, University of Reading, U.K., 1996.

[2] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, MA, 1989.

[3] J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, 1975.

[4] Z. Michalewicz. *Genetic algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1996.

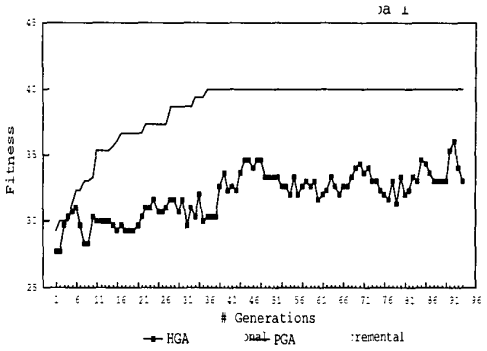

Figure 3. Mean fitness values for Prisoner's Dilemma with 16 chromosomes.

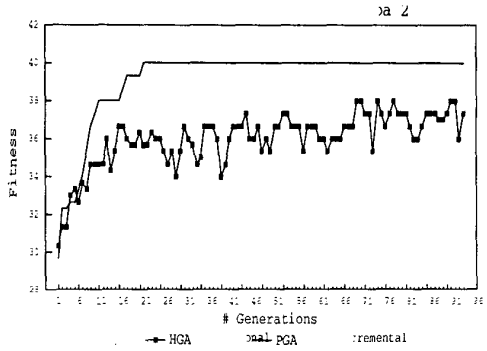

Figure 4. Mean fitness values for Prisoner's Dilemma with 32 chromosomes.

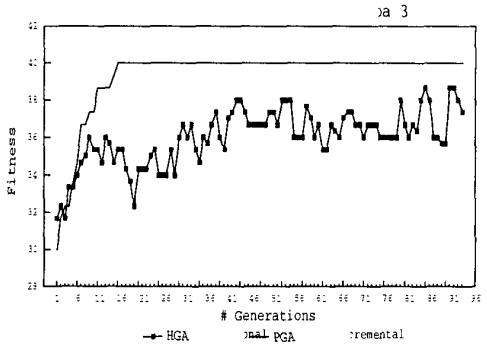

Figure 5. Mean fitness values for Prisoner's Dilemma with 64 chromosomes.