Design of a Programming Library for the implementation of GAs and PGAs using MPI

Aquiles Jose Manuel Mirón Enríquez, Darnes Vilariño Ayala
Benemerita Universidad Autónoma de Puebla
aquilesjm@gmail.com, darnes@solarium.cs.buap.mx

Abstract

A design for a programming library for implement sequential and parallel genetic algorithms is presented. The packages and classes designed are explained and represented by UML diagrams. This design is intended to develop flexible and reusable code that permits evaluate performance of genetic algorithms varying easily genetic and parallel parameters.

1. Introduction

All Genetic Algorithms (GAs) are search techniques that, based on principles of natural selection and genetics, are applied to solve successfully problems in many different fields as scientific researching, engineering design and business [F].

Particularly in this paper we are addressing a type of GAs, the Parallel Genetic Algorithms (PGAs) which use parallel computing to improve the performance of GAs. Genetic Algorithms are by nature easily paralleled, and because of this a considerable amount of research has been done on this topic [B].

The goal in this paper is to present the design of a programming library for implementing PGAs. The purpose of this library is to abstract the programmer from the details of the communications, migrations, and synchronization in the parallel aspect; and from generations breeding, crossover and selections methods in the genetic facet of a PGA, allowing at the same time, enough flexibility to optimize and adjust this genetic operators. The library design was originally developed to perform a series of tests of GAs and PGAs to solve a specific kind of combinatory optimization problems. Specifically the intention is to use a series of theoretical advances [A][C][D][E] both in sequential and parallel GA to propose a PGA (or GA) to solve this type of problems.

The paper is organized as follows. First, we offer a brief introduction to genetic algorithms and its main concepts; the third section explains parallel GAs and the different models that have been developed of PGAs. Next, the design of the programming library is presented, and the basic interactions between its components are explained. In the fifth section, the current state of the implementation is discussed along with the testing method used over the code; the paper ends with the conclusions and the future work with the library.

2. Genetic Algorithms

Genetic algorithms are stochastic search algorithms based on principles of natural selection and genetics. In this section we give a basic introduction to this algorithms and its associated terminology.

GAs generally represents the possible solutions to a search problem by strings of finite length. Such strings, also called *chromosomes* are the equivalent of individuals of a species. Each chromosome or individual, encode a solution, this is to say, the value in one given position of the string represent the value of a variable implied in the potential solution.

A group of chromosomes is called *population*. In a genetic algorithm the population or populations are evaluated to select the best individuals (depending on the algorithm, random or even worse individuals can be selected), to mate and reproduce to form other population. This procedure is repeated through a number of cycles or *generations*, with each population having better individuals (potential solutions) than its predecessor.

This is very similar to natural evolution, where the fittest individuals have more probabilities to spread their genes to the next generation. In genetic algorithms, or in any method that uses evolution to find the solution (namely, evolutionary algorithm), is necessary to define how to measure the fitness a

chromosome. This is accomplished by the definition of a *fitness function*, mathematically based (objective function), or such measure can be given by humans (subjective function) [I]. This fitness measure, once attained, is used to guide the evolution, favouring to bring up good solutions.

This process of selection and reproduction is carried out applying several functions, called genetic operators. These functions are basically: *selection*, *crossover* and *mutation* and are defined according to the type of chromosome and the characteristics of the problem.

Another important aspect to consider is defining when the GA is going to stop, in other words, the *stop condition*. This can be when a predefined number of generations are reached; when the best individual (or the average of the population) give an acceptable solution to the problem; or when the population has *converged*, this is, when all the individuals (or a percentage of them), are the same.

Defining a GA implies to define these operators, and the parameters of the GA itself, such as: size of the population, type of selection, probabilities of crossover and mutation, stop condition parameters (number of generations, fitness desirable, etc), type of chromosome to be used (binary, real, alphabetic, etc). Besides this, a GA usually require to incorporate problem knowledge, for example, when solving a combinatory problem, an individual who does not comply with the restrictions will never generate an acceptable solution, thus it has to be discarded or corrected; being required an additional function to do this.

Although the GAs can find solutions hard or impossible to obtain with other techniques, the time required to get such solutions can be an important constraint when they are applied. For this reason, one method to improve their performance, and therefore reducing time, is using parallel computing. In the next section the different models of parallel genetic algorithms are discussed.

3. Parallel Genetic Algorithms

The basic idea in parallel computing is to divide a task in subtasks in such way, that all or most of the subtasks are performed at the same time using multiple processors. In the case of GAs such division can be easily performed conceptually speaking. As GAs manages groups of individuals, and all of them have to be evaluated in the same way (by the fitness function), parallelizing a GA can consists on dividing the population in smaller groups, assigning each one of them to one processor and make them evolve independently (distributed); or sending the subgroups to different processor only for their fitness evaluation (centralized).

Depending upon the way the population is structured, the parallels GAs can be classified in three main types [B]: global population, master-slave communications; single population fine-grained and multiple populations coarse-grained.

There are two aspects of PGAs to consider in this classification: the internal structure of the populations (how individuals are related to each other) and the nature of communications between processors in the algorithm.

In a GA with global population, a chromosome can potentially mate with any other in the population, and because of this characteristic, this type of algorithms are called *panmictic* PGAs [AAA]. In such design, there is one processor (master) that controls the global population, selects, mates and mutates chromosomes, sending them to other processors (slaves) for their fitness evaluation.

The second type of PGAs, are those that have also a single population, but there is an structure between the individuals, that restricts the mating between them. Typically, the selection and crossover are restricted by "neighbourhoods", this is to say, chromosomes are logically organized by locations, and a chromosome can only mate with chromosomes in the same location o neighbourhood. To permit the interaction among all the individual, these neighbourhoods overlaps between them. These PGAs are appropriate in massive parallel computers, with an individual by processing unit. Such model is called *cellular* or fine-grained PGA.

Next, we have the algorithms that consist of several populations that evolve independently, exchanging individuals from time to time. This exchange of chromosomes, called migration, makes this model sophisticated, because it's needed to specify the source and destination, frequency, size and type of the migration, among other parameters. This type of algorithms, also called model island PGAs, are popular because they can be implemented using a computer network, without need of special hardware. But they are the most difficult to understand, because the effects of varying the migration parameters are not well understood yet. Furthermore, each one of the processors in the PGA can have subpopulations and migrations between them, making the algorithm even more complex. These algorithms are called *distributed* or coarse-grained PGAs.

This work is focused on the third model, proposing a system that allows sequential and distributed PGAs.

4. System Design

The goal of this paper is to present the design of a programming library useful to implement genetic algorithms both sequential and parallel. This system was originally planned to test several models of GAs, with the intention of proposing a PGA design suited to solve a specific type of combinatory optimization problem.

A main problem when distributed PGAs are designed is to set the parameters that control the migration between processors. This parameter selection constitutes a complex problem by itself, reason why it is needed a system that permits to change such parameters without much difficulty to the programmer, and to reuse generic components to build a specific PGA. The system designed is intended to obtain such flexibility and reusability.

4.1 System characteristics

As said before, the origin of this design was to solve a specific kind of problem, but in order to achieve that, it is necessary to have a way to perform variations in the genetic and parallel parameters of the algorithm, without rewriting the code. Thus, the purpose of the library to be developed can be extended to include other types of problems, as long as they require sequential or distributed parallel GAs. To achieve that, a set of desirable characteristics of the system can be stated as follows:

Allows to change the fitness function easily, capability by the programmer to add and change mechanisms that integrate problem knowledge, and permits to set easily the genetic (and parallel if appropriate) parameters of the GA.

4.2 UML design

The library classes are organized in four main packages, according to their functionality. In the following UML diagram, these packages and their relationships are shown.

The main classes of the library and their relationships are represented in the following UML diagram.

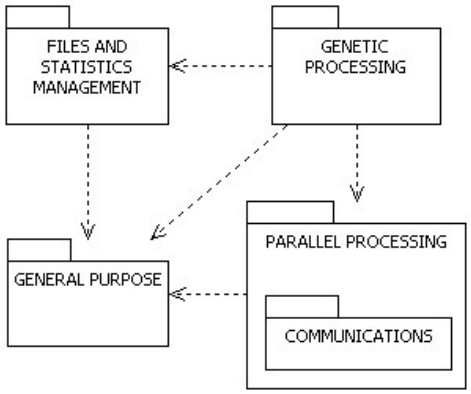

Figure 1. Library packages

The classes and relationships contained in each one of these packages are shown and explained below.

GENETIC PROCESSING: The classes contained in the package and their relationships are shown in the following class diagram:

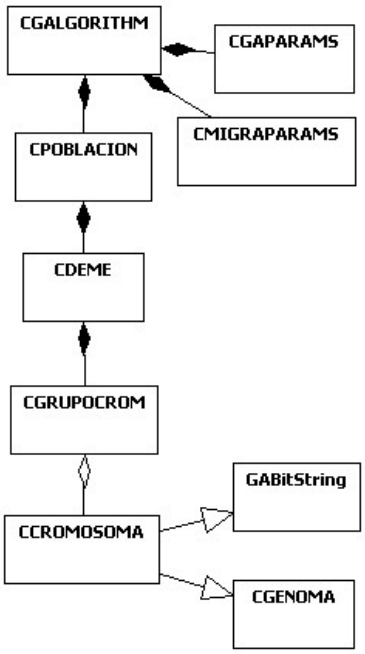

Figure 2. GENETIC PROCCESING classes

As figure 2 represents, the classes in this package abstract the different levels in a population: first we have the CGALGORTHM class, an object of this class represents a genetic algorithm running in a processor, a CGALGORITHM object will have a CPOPULATION, class which manages the population o populations evolving in the algorithm. There can be subpopulations or demes in a GA, and they are represented by objects

of the CDEME class. Every CDEME object will have a CGRUPOCROM object, which controls directly the individuals evolving. Such individuals will be objects instanced from the CCROMOSOMA class. CCROMOSOMA class inherits from the CGABitString and CGENOMA classes. CGENOMA has the methods and attributes genetic related, and CGABitString the bit string management functionality. CGAPARAMS and CMIGRAPARAMS are two auxiliary classes that encapsulate the parameters genetic and parallel respectively.

FILES AND STATISTICS MANAGEMENT: This package contains the classes which are capable to write/read files, and to do statistics control. The next figure illustrates them.

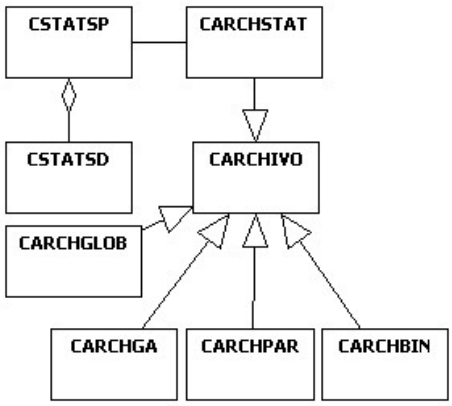

Figure 3. FILES AND STATISTICS MANAGEMENT classes

In the figure above, the classes belonging to this package are shown. CARCHIVO is the parent class that defines the basic file functionality, and another five classes are derived from it. CARCHSTAT writes to disk the log of the evolutionary process and the statistics associated. CARCHGA reads from disk the file containing the genetic parameters to be implemented. CARCHPAR do the same thing with a file that contains the parallel parameters. And CARCHBIN is a class with the purpose to get the restriction matrix from a file; is a class intended for the specific case of solving a combinatorial problem. CSTATSD retrieves and controls the statistics by subpopulation and CSTATSP do the same with the total of the population. CARCHGLOB is used to write the general information of the evolution on disk.

PARALLEL PROCESSING: This package encapsulates the classes in charge to do the migration between subpopulation and processors, and therefore of the communications in the COMMUNICATIONS sub package. This is presented in the next figure.

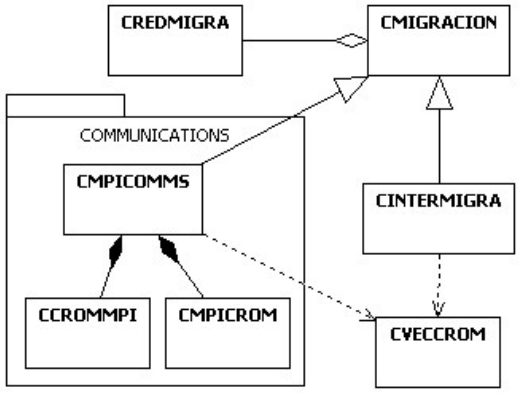

Figure 4. PARALLEL PROCESSING classes

CMIGRACION is the parent class of CMPICOMMS and CINTERMIGRA classes. CMIGRACION defines the abstract methods to do the migration. CINTERMIGRA is the derivate class that is responsible for the migration among subpopulation. CMPICOMMS is the class that makes the communication and migration between different processors possible. That is the reason because CMPICOMMS is contained in the sub package COMMUNICATIONS. Both CINTERMIGRA and CMPICOMMS inherited from CMIGRACION its relationship with the class CREDMIGRA. An object CREDMIGRA contains a representation of the migration network between demes or populations. The sub package COMMUNICATIONS contains all the classes that manages the MPI communications, being CCROMMPI and CMPICROM the classes that convert from MPI data to genetic objects and vice versa.

GENERAL PURPOSE: This package contains two classes used for general purpose by several other classes.

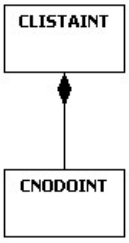

Figure 5. GENERAL PURPOSE classes

CLISTAINT is a class that manages integers in a bounded list, and CNODOINT is the container of the integer, and the elements from that list.

5. Implementation

Currently the development of the library is being implemented as part of a Master degree thesis. The progress of the library is estimated in 60 %, being the GENETIC PROCCESSING the package with more progress with a 90 % of implementation.

6. Conclusions and future work

A design for a programming library for PGAs development is proposed, focusing in sequential and parallel island model PGAs. It is intended to test this library with a series of combinatorial optimization problems, applying several theoretical advances in the field.

Further work is test this library to resolve specific problems, in particular a multidimensional knapsack problem is intended to be solved by this library, developing both a secuential and parallel version.

7. References

[AAA]Alba E., Tomassini M. (2002). Parallelism and Genetic Algorithms. IEEE Transactions on Evolutionary Computation 6 (5), pp. 443-462.

[A] Cantú-Paz, E., & Goldberg, D.E. (1997a) “Modeling idealized bounding cases of parallel genetic algorithms”. In Koza, J., Deb, K., Dorigo, M., Fogel, D., Garzon, M., Iba, H., & Riolo, R. (Eds.), Genetic Programming 1997: Proceedings of the Second Annual Conference (pp. 353-361). San Francisco, CA: Morgan Kaufmann Publishers.

[B] Cantu-Paz, E. (1998). A survey of parallel genetic algorithms. Calculateurs Paralleles, Reseaux et Systems Repartis. Vol. 10, No. 2. pp. 141-171. Paris: Hermes.

[C] Cantú-Paz, E. and Goldberg, D.E. (1999). “On the scalability of parallel genetic algorithms”. Evolutionary Computation. 7(4), 429-449.

[D] Cantú-Paz, E. (1999b). “Designing Efficient and Accurate Parallel Genetic Algorithms”. PhD thesis. University of Illinois at Urbana-Champaign.

[E] Cantu-Paz, E. (2001). “Migration policies, selection pressure, and parallel evolutionary algorithms” Journal of Heuristics. 7(4), 311-334

[F] Goldberg D. E. (1994). "Genetic and evolutionary algorithms come of age". Communications of the ACM, vol. 37, no 3, p. 113–119.

[G] Goldberg, D. E., (1989b) “Genetic Algorithms in Search Optimization and Machine Learning”, Addison-Wesley, Reading, MA.

[H] Holland, J. H., (1975) “Adaptation in Natural and Artificial Systems”, University of Michigan Press, Ann Arbor, MI.

[I] Sastry, K., Goldberg, D.E., Kendall, G. (2005). In Burke, E. and Kendall, G. (Eds), Introductory Tutorials in Optimization, Search and Decision Support Methodologies. Berlin: Springer.