A Java-based Distributed Genetic Algorithm Framework

Gabi Escuela, Yudith Cardinale and Jorge González

Universidad Simón Bolívar
Departamento de Computación y TI
Apartado 89000, Caracas 1080-A, Venezuela.
{gescuela, yudith, 01-33936}@ldc.usb.ve

Abstract

Distributed Genetic Algorithm (DGA) is one of the most promising choices among the optimization methods. In this paper we describe DGAFrame, a flexible framework for evolutionary computation, written in Java. DGAFrame executes GAs across a range of machines communicating through RMI network technology, allowing the implementation of portable, flexible GAs that use the island model approach. Each island can be configured independently from others providing the implementation of heterogeneous DGAs. To evaluate the performance of DGAFrame, we implemented the Protein Structure Prediction problem and compare the DGA execution to its sequential counterpart through quality of solution. We also measure the computation to communication ratio and results show that the proposals consistently outperform equivalent sequential GAs.

1. Introduction

Genetic Algorithms (GAs) [13] are efficient search methods that are being applied successfully to find acceptable solutions to search, design and optimization problems in business, engineering, and science. They are based on a basic evolutionary principle: better individuals survive and reproduce more often than other individuals. Given a problem to solve, a GA considers a set of potential solutions as a population of individuals. An individual is represented using a genotype that encodes the variables of the problem. Individual aptitude to solve the problem, called fitness, is measured in accordance with an objective function that guides the algorithm. Generally, a first population is generated randomly and it evolves in an iterative way through generations, in which selection, crossover and mutation operators are applied to individuals in order to get genetic variations that allows obtaining better solutions, in terms of objective function.

Mostly, a GA implementation is problem-oriented in terms of design decisions about adequate genetic encoding and operators. Even thought, GAs produce “good enough” solutions to a variety of problems, it is observed that there are two main difficulties with them. One is premature convergence problem, making the evolution stuck to a local minimum due to a lost of genetic diversity. The other difficulty concerns to performance. When GAs are applied to larger and harder problems with complex searching spaces, there is an increase in the time required to find adequate solutions. To overcome these difficulties, many authors have proposed distinct approaches, as modified genetic operators [18] and methods based in the diversity preservation, through a spatial separation of the population [3, 17].

Distributed Genetic Algorithm (DGA) is one of the most promising choices among the spatial separation methods. In the literature, this model is sometimes also referred as the Island Model, where each subpopulation represents an island or deme. This implies having subpopulations distributed among many processors, which requires a new operator to be added to the problem configuration: migration. Migration consists of selecting individuals to be exchanged amongst subpopulations. Technically the migration of individuals is controlled by three parameters: the topology that defines connections between subpopulations, migration rate that controls the percentage of individuals selected for migration from each subpopulation, and migration intervals that affect how often migration occurs [5, 9].

Four basic island variants are seen to exist, each with appropriate migration operators [21]: i) all islands execute identical parameters, resulting in homogeneous DGAs, ii) all islands execute different parameters, obtaining heterogeneous DGAs, iii) each island evaluates different objective functions, where islands have a (possibly) reduced problem domain each executing a GA, and iv) each island represents a different region of the genotype or phenotype domains by isolating each processor to solve specific, nonoverlapping regions of phenotype or genotype space.
In this paper we describe DGAFrame (Distributed Genetic Algorithm Framework), a flexible framework for evolutionary computation, written in Java. It is designed to aid programmers in the development of applications that use evolutionary algorithms to solve optimization problems by allowing reusing design and code based on the object oriented paradigm. To evaluate the performance of DGAFrame three classical optimization problems were implemented: Boolean Satisfactibility Problem (SAT), Protein Structure Prediction (PSP), also called Protein Folding Problem, and Traveling Salesman Problem (TSP). In this paper we only report experiments and results obtained implementing PSP. We assess the quality of our design and implementation through a measurement of the computation to communication ratio and quality of solution.

2. DGAFrame Description

DGAFrame executes GAs across a range of machines communicating through Java RMI network technology, providing the implementation of portable, flexible GAs that use the island model approach. Each island can be configured independently from others. Making distinctions between the subpopulations of a DGA by applying GAs with different configurations, we obtain heterogeneous DGAs. All configuration parameters should be specified with XML files, giving the advantage not to recompile the code each time these parameters are modified.

2.1. Communication Model

The communication model of DGAFrame follows the master-worker distributed systems paradigm that supports the island model. Through a GUI, the user defines the DGA and configures the Coordinator, which controls the distribution of subpopulations and receives results, and Servers, which evolve their own initial population (step 1 on Figure 1). Once configuration XML files are generated, the Coordinator distributes them amongst Servers and start the execution (step 2). Migration is executed asynchronously during the execution (step 3). When execution is finished, the Coordinator collects results (step 4).

2.2. Packages Overview

In this section we explain the main packages offered by DGAFrame.

Package Genetic the main class of this package is GATemplate, an abstract class which contains methods to configure the GA with a particular problem, and to initiate an experiment (runGA() method). It has two abstract methods: evalIndividual() and configureGA(). Users must extend this class and implement these methods for a new problem implementation. configureGA() allows the user to instantiate the operators and other configuration needed before the run. It is specially useful when the problem requires specialized genetic operators.

Package Operators has classes related with genetic operators: selection, crossover and mutation. Each operator is implemented through an interface to facilitate the specialized operator construction. The current implementation of DGAFrame offers the operators: Tournament selection, AllPoints or OnePoint mutation, and Two points and Uniform crossover.

Package Individual contains an Individual interface, with common GA methods. Some of these methods are: obtainChromosomeLength(), obtainFitness() and obtainAllelValue(). The abstract class IndividualTemplate defines the different gen types offered by DGAFrame. It is extended in three concrete classes: IntegerGen, BinaryGen and CharacterGen. User can add new classes for representing other types of chromosomes due to the library encoding flexibility.

Package RemoteInterface contains necessary interfaces to describe methods that are remotely available. ServerInterface class defines remote method signatures to be implemented by Servers. This class has a method migrateSubPopulation(), which is used by the class GATemplate as the GA migration operator. Experts users can overwrite this method to personalize the migration mechanism. Interface ProblemDefinition allows users to specify the problem to solve. It defines important methods as: getChromosomeLength(), readInstance(), getGenType().

Package Process contains two subpackages: Master and Server. The first one contains the class Coordinator which is responsible for Servers configuration and the synchronous initiation of the execution of every
GA on each island. Package Server contains the class Server with GA code which will be executed by each island.

2.3. Configuration Files

DGAFrame uses XML files to specify the configuration of a GA, thus the information that they contain is aimed to: GA common parameters that define the specific optimization problem, Coordinator configuration and Servers identification and configuration. This XML specification about evolutionary algorithms looks for simplicity and easy to use, including a complete set of necessary arguments in this matter. Each label in XML files is self-descriptive, resulting in a more easy way for user in configuration process. This is an example for the XML file that contains problem definition details defining the PSP problem:

```xml
<problem-definition>
<optimization-direction>max</optimization-direction>
<chromosome-length>20</chromosome-length>
<instance-file>/home/test.txt</instance-file>
<initial-fitness>0.0</initial-fitness>
<gen-type><alphabet-character="LFR"></gen-type>
</problem-definition>
```

3. Related Work

For a long time, there have been a special interest in developing C and C++ libraries special for distributed and parallel processing [3, 15, 1, 11, 10, 4]. However most of those libraries have many disadvantages: i) they employ some rather arcane programming techniques, leading to considered developing time and effort, and making for a steep learning curve; ii) the use of compiled libraries in a heterogeneous computer network may be hindered by issues with code portability; iii) they usually have many “hard-wired” parameters which bring the need to recompile the program if users want to change them. The main advantages of DGAFrame compared to those existing GA libraries are the high scalability reached by the distribution of evaluation and reproduction, the platform-independence allowing heterogeneous networks to collaborate, and user-defined, reusable simulation environment.

In the following, we concentrate the comparison of DGAFrame with frameworks for DGAs implemented in Java. JIDEAL (Java Distributed Evolutionary Algorithms Library [7]) is a Java objects library for developing Evolutionary Computation experiments. It lets the user distribute some task through a heterogeneous computer network, but it does not provide the island model. Compared with DGAFrame, JIDEAL is just a library, while DGAFrame is a framework which provides a friendly GUI that hides many programming details and allows for less error-prone developing process. DGAFrame is specially to implement DGAs with the island model. On the other hand, DGAFrame uses XML configuration files, giving the advantage to do not recompile the code each time these parameters are modified.

ECJ [16] is an evolutionary computation framework that provides a basic GUI for loading and running jobs from parameter files and editing parameters as DGAFrame does. However DGAFrame GUI has other features such as automatic XML files generation and start the execution. There are restrictions placed on the islands in ECJ: they must have the same kind of subpopulations and species; each subpopulation must send the same number of migrants as any other subpopulation; and migrants from a subpopulation in an island must only migrate to the same subpopulation in other islands. DGAFrame places none of the above restrictions on subpopulation migration: user can specify different migration parameters to each island.

JEO (Java Evolving Objects [12]) is a framework for building evolutionary computation experiments. It is designed for any Evolutionary Computation paradigm, not only for GA, as is DGAFrame. However the experiments specification in JEO should be done by using an special language called EASEA (EAsy Specification of Evolutionary Algorithm) [6], while DGAFrame employs XML file and a GUI to make easier the deployment of GA experiments.

We also want to compare DGAFrame communication approach with other works. For example, the dynamic DGA proposed in [22], divides the population into subpopulations, as well as in DGAFrame, but there still exists a central monitor, which observe the performance of each sub-population and adjust their size according every certain generation. The resizing and migration is performed by the central monitor in a synchronous way and only the best individuals could be candidates for migration, while the migration in DGAFrame can be asynchronous and the emigrants could be a combination of best and random individuals, with the purpose of introducing diversity into the demes to try to alleviate the problem of converging prematurely to a solution of low quality.

In [19] is proposed EVOLVE/G, a middleware to enable implementation of evolutionary computing on grid platforms. EVOLVE/G offers an API to implement the components of the computation (Worker, Agent and Super Agent). That means, it is not transparent for developers the distributed implementation as in DGAFrame is. We like the idea of extending DGAFrame to be used on grid platforms but keeping the transparency for users and developer.

There are other tool implementations which use Jini network technology to build a distributed GA [2, 23], instead of using RMI directly as DGAFrame does. Jetic [2] has each genetic operation and even chromosomes set up as Jini services. This design choice resulted in a large amount of communication overhead compared with using RMI as the backbone of the distributed system. [23] introduces JavaSpace object repositories to provide for a more efficient and elegant solution, since they facilitate loosely coupled communication between entities in a Jini federation, however this
solution still represents a higher overhead communication compared with using RMI. Those Jini-based implementations are tools that facilitate the distribution of the computation but do not provide GUI and other facilities to help developers in building DGAs in an easy way.

Many libraries and frameworks for DGAs implement migration only for the best individuals. However, the population diversity could be reached keeping the competition among individuals fairer. The Hierarchical Fair Competition (HFC) model is originated from an effort to avoid the premature convergence in traditional evolutionary algorithms [14]. The fair competition is obtained in HFC model by dividing the individuals in independent castes or classes according with their skills. In HFC model, multiple demes are organized in a hierarchy, in which each deme can only accommodate individuals within a specified range of fitness. Individuals are moved from low-fitness to higher-fitness subpopulations if and only if they exceed the fitness-based admission threshold of the receiving subpopulations. We plan to incorporate the HFC in DGAFrame.

4. Experiments and Results

PSP is a complex open problem pursued by computational structural biology. It has the aim to predict tertiary structure (specific three-dimensional shape or fold) of proteins from the primary structure (amino acid sequence). Considering it difficult, many authors use a simplified protein model reducing algorithms search space. We considered the HP model[8], which divide the 20 naturally occurring amino acids that make up protein chains in two groups: hydrophobic (H), and polar or hydrophilic (P).

The input data for the algorithm consists in a HP protein instance, formed by a string \( i \in \{H, P\}^n \), where \( n \) is the number of amino acids or protein length. Additionally, in our PSP experiments, tertiary structures are represented by internal relative coordinates on a square 2D lattice, where each move is interpreted in terms of neighborhood contacts (with lattice units). The native or functional state that achieves the lowest energy, which can also be seen as the non-local H-H contacts maximization. Equation \( F(s) = n_H(s) - \rho \times n_L(s) \) represents the objective function, where \( n_H \) is the number of non-local H-H contacts, \( \rho \) is a penalty term, and \( n_L \) is the number of intersections present in conformation.

We selected four instances (see Table 1) with known optimal, used in PSP related articles [20]. Parameter settings for the evolutionary algorithm are listed in Table 2. The frequency and the rate of migration, as well as the rates of mutation and crossover are the result of a systematic parameters tuning realized for this problem. All experiments were executed in PC’s with dual-core processor 3.4 GHz Pentium D, 1GB of memory, connected with 10 Mbps Ethernet, with Debian Etch 4.0.

First, we performed several experiments to know the impact in terms of better solutions for 3 islands when we introduced the migration operator. That is, we run the DGAFrame PSP implementation without migration for an independent evolution in each island. Afterwards, we run it in a distributed way. The two best fitness obtained and their frequency for 50 runs, are shown for the instances with and without migration (see Table 3). Migration seems to be helpful to the evolutionary search in this context, not only by its clearly superior performance, as by its better scaling properties in terms of instance length.

### Table 1. HP Protein Instances used in PSP problem experiments.

<table>
<thead>
<tr>
<th>Id</th>
<th>HP Protein</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>i20</td>
<td>HPHPPHHPPHHPPHPPPHPPPH</td>
<td></td>
</tr>
<tr>
<td>i36a</td>
<td>PPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPH</td>
<td>36</td>
</tr>
<tr>
<td>i36b</td>
<td>PPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPHHPPH</td>
<td>36</td>
</tr>
<tr>
<td>i50</td>
<td>HPHPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPPHPPH</td>
<td>50</td>
</tr>
</tbody>
</table>

### Table 2. Parameter values used for PSP tests.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Island Model Parameters:</td>
<td></td>
</tr>
<tr>
<td>Number of subpopulation</td>
<td>2,3,4,8</td>
</tr>
<tr>
<td>Size of subpopulation</td>
<td>Uniform</td>
</tr>
<tr>
<td>Topology</td>
<td>Ring</td>
</tr>
<tr>
<td>Migration Mechanism</td>
<td>50% random, 50% best, replace worst</td>
</tr>
<tr>
<td>Migration Frequency</td>
<td>30</td>
</tr>
<tr>
<td>Migration Rate</td>
<td>20</td>
</tr>
<tr>
<td>Local GA’s Parameters:</td>
<td></td>
</tr>
<tr>
<td>Number of Individuals</td>
<td>100</td>
</tr>
<tr>
<td>Replace Model</td>
<td>Generational and Elitism</td>
</tr>
<tr>
<td>Selection approach</td>
<td>Tournament-2</td>
</tr>
<tr>
<td>Mutation (Type: Rate)</td>
<td>All points: 0.6/Fold length</td>
</tr>
<tr>
<td>Crossover (Type: Rate)</td>
<td>Uniform: 1.0</td>
</tr>
<tr>
<td>Termination condition</td>
<td>1000 generations (for each subpopulation)</td>
</tr>
<tr>
<td>Run numbers</td>
<td>50 per experiment</td>
</tr>
</tbody>
</table>

In order to prove DGAFrame performance, we made several runs raising the number of demes (machines) and measuring times. In distributed systems, granularity is a qualitative measure of the ratio of computation to communi-
Table 3. Comparing best solutions with and without migration for PSP problem.

calculation times (Cp/Cm), and it is established that the higher this ratio, the more efficiently its resources are used. Using 2, 4 and 8 demes, the computation and communication time for each Server was calculated for i50. In this test, if at least one Server reaches the optimal solution this is consider as a termination condition instead reaching 1000 generations. The obtained values for Cp/Cm were 10.3 (2 demes), 10.875 (4 demes) and 11.854 (8 demes). We can note that DGAFrame uses efficiently the system resources, as Cp/Cm increases while number of demes increases too. In these experiments communication time represents only between 6% and 8% of total execution time. On the other hand, obtained best fitness averages were 16.6 (2 demes), 19.4 (4 demes) and 19.9 (8 demes). Results suggest that quality of solutions is improved while the number of processors is increased, due to collaboration amongst islands keep diversity on subpopulations.

5. Conclusions and Future Work

We have introduced DGAFrame, a flexible framework for distributed evolutionary computation based on object oriented technologies: Java, RMI, and XML. Our experiments have shown that the quality of solutions are improved by using the DGAFrame distributed approach. Additionally, the computation to communication ratio demonstrates that DGAFrame uses resources efficiently.

We are currently working on the possibility of evaluate different objective functions on each island (to solve multiobjective optimization problems) into DGAFrame. Additionally, we are extending DGAFrame to be used transparently on grid platforms.

References