A Genetic and Evolutionary Programming Environment with Spatially Structured Populations and Built-In Parallelism

Miguel Rocha, Filipe Pereira, Sónia Afonso, and José Neves

Departamento de Informática
Universidade do Minho
Braga
PORTUGAL
mrocha@di.uminho.pt, jneves@di.uminho.pt

Abstract The recent development of the *Genetic and Evolutionary Computation* field lead to a kaleidoscope of approaches to problem solving, which are based on a common background. These shared principles are used in order to develop a programming environment that enhances modularity, in terms of software design and implementation. The system's core encapsulates the main features of the *Genetic and Evolutionary Algorithms*, by identifying the entities at stake and implementing them as hierarchies of software modules. This architecture is enriched with the parallelization of the algorithms, based on spatially structured populations, following coarse-grained (*Island Model*) and fine-grained (*Neighborhood Model*) strategies. A distributed physical implementation, under the *PVM* environment, running in a local network, is described.
Keywords: (Parallel) Genetic and Evolutionary Algorithms, Spatially Structured Populations.

1 Introduction

The unfolding of the *Genetic and Evolutionary Computation (GEC)* arena has been remarkable in the last few years. The success of the applications in scientific and engineering domains is, in fact, an undeniable fact. However, in the processes of software development and analysis, the success has not been of a similar nature, being still common that the elaboration of a *Genetic and Evolutionary Algorithm (GEA)* may imply the programming of an application from scratch. The motivation for this work stems from these facts, leading one to foresee programming environments that will enable programmers to adjust software modules to be reused.

In order to achieve the proposed aim, the major features of the object-oriented programming paradigm are used, namely its capabilities to divide and conquer, reutilization or modularity. Indeed, under the present framework, the entities that make the building blocks of the the different approaches to *GEC* are identified, in terms of their common background. Each conceptual entity

is, on the other hand, viewed in terms of an hierarchy of abstraction spaces, augmenting the programmer's degree of freedom.

The programming environment allows for several kinds of users, depending on the task's complexity and on the user's skills and knowledge. It provides both a tool for the rapid development of an application, taking advantage on existent knowledge, and for the possibility of redefining data structures or operators, at different abstraction levels, thus making room to the advent of more complex program's features.

The earlier work in the development of the proposed system contemplated panmictic *GEAs*; i.e., with a single evolving population [9]. The system has been used, both for academic purposes, being the basis for several projects executed by undergraduate students, and for practical applications, namely in the *Combinatorial Optimization (CO)* field, where it was applied to tasks such as the *Job Shop Scheduling Problem* [11], the *Traveling Salesman Problem*, the *0/1 Knapsacking Problem* or the *Graph Coloring* one [12]. It was also object of a process of fusion with an implementation of *Artificial Neural Networks*, developed under a similar methodology, giving rise to some interesting problem solving techniques, namely in the *Machine Learning* arena [10].

More recently, new features have been added to the system, namely considering different computational models for its parallelization, in terms of the population's spatial structure. So, new hierarchy levels were added to the environment, following two basic models: the *Neighborhood Model* and the *Island Model* [3]. In the former, a fine-grained strategy is followed, with a spatial structure being fed to the populations, by considering coordinates for each of its individuals in a n dimensional space. In the latter, a coarse-grained approach is considered, with different sub-populations evolving simultaneously and exchanging individuals at regular intervals. Both models have been implemented, in a sequential way, considering a single CPU.

Furthermore, a distributed implementation of the latter model is presented, built under the *Parallel Virtual Machine (PVM)* [1] environment. This system allows for the definition of several populations, evolving in different machines, and exchanging information through a local network infrastructure.

The paper is organized as follows: firstly, an overview of the basic framework developed for panmictic *GEAs* is presented; then, the two parallel *GEAs* models are defined and their implementation is described; next, the parallel implementation of the *Island Model* is uncovered; finally, some conclusions are drawn and prospective future work is presented.

2 Panmictic Genetic and Evolutionary Algorithms

The term *Genetic and Evolutionary Algorithms (GEA)* is used to name a family of computational procedures where a number of potential solutions to a problem makes the way to an evolving population. Each individual codes a solution into a string (*chromosome*) of symbols (*genes*), being assigned a numerical value (*fitness*), that stands for a solution's quality measure. New solutions are created

through the application of genetic operators (typically *crossover* or *mutation*). The whole process evolves via a process of stochastic selection biased to favor individuals with higher fitnesses. Under this scenario, panmictic *GEAs* are defined as those that allow for any individual to reproduce with any other; i.e., there is no isolation by a spatial structure.

The architecture of the *GEA's* model is built upon three conceptual levels, that encapsulate the features and behaviors of the main entities involved: the *individuals*, the *populations* and the *GEA* itself [9]. Each of these abstraction levels is materialized by an hierarchy of classes, whose root defines the set of common data structures and methods, as well as a set of default definitions. Specific structures and methods are defined when one descends from the root into the leafs of the hierarchy.

As an example, consider the individual's hierarchy of classes depicted in Figure 1. The root class, *Indiv*, is an abstract one; its role is to define a set of common procedures and interfaces, to be implemented in its sub-classes. This class has a template field, that can be assigned, *a posteriori*, with the respective type. This field is used to keep the genetic information of an individual, a sequence of genes of a given type. Thus, it is possible to consider different types of representation alphabets. In Figure 1 some of the built-in representations are shown, but this set can be augmented, when the need arises.

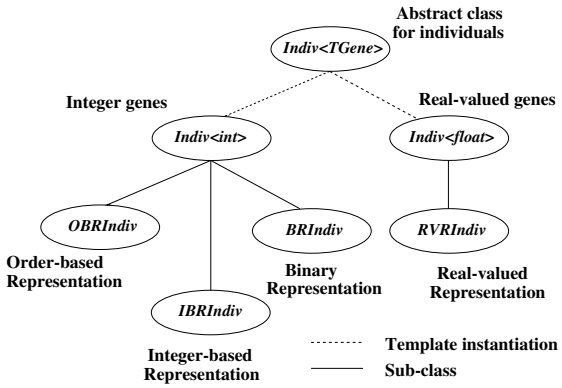

Figure 1. The *individuals* hierarchy's class

Similar strategies are used at the *population's* and the *GEA's* abstraction levels. In the former, the selection and the re-insertion procedures are defined, as well as the methods for creating the initial population. In the latter, the overall structure of the algorithm is created, being provided a default one, namely those depicted in Figure 2, to can be used by the programmer or redefined.

The last bit of the system's architecture is the *Evaluation Module*, whose function is to encapsulate the problem's dependent *decodification* and *evaluation*

```
BEGIN
  Initialize time (t = 0).
  Generate and evaluate the individuals in the initial population (P_0).
  WHILE NOT (termination criteria) DO
    Select from P_t a number of individuals for reproduction.
    Apply to those individuals the genetic operators to breed the offspring.
    Evaluate the offspring.
    Select the offspring to insert into the next population (P_{t+1}).
    Select the survivors from P_t to be reinserted into P_{t+1}.
    Increase current time (t = t + 1).
  END WHILE
END
```

Figure 2. Structure of a *GEA*

processes. This is achieved by an abstract class defining the syntax for the evaluation procedure. This class is instantiated whenever a different problem is to solved by a *GEA* (the corresponding fitness function is defined at this stage).

One interesting feature of the system under consideration relies on the genetic's operator handling flexibility. In fact, the user can specify, for a particular problem, the set of operators he/she may find more adequate, and also state their frequency of application. The concept of genetic operator was generalized to endorse both the traditional operators, such as *crossover* and *mutation*, and any kind of operator that may be considered of interest (since it can be stated as a function that takes n individuals and returns m different ones).

It is also provided a set of classes to configure the *GEA*, that can be used to set the values for the parameters in the *selection* and the *re-insertion* procedures, as well as the termination criteria, the population size or other parameters that control the process of evolution and the system's interfaces with the outside world.

One of the major system's requisites relies in its capability to define hybrid approaches; i.e., those that take the information of a problem's instance, and use it in the design of the genetic operators. The generalization of the genetic operator concept, and the hierarchy of classes at the individual's level, that allows the definition of specific operators and general purpose ones, not only makes possible to bring such operators into life.

3 The Neighborhood Model

In the *Neighborhood Model*, each individual in a population is given an unique location in a n-dimensional space, according to a predefined *topology*. The position of a given individual is given, in this case, by a set of n coordinates, being required that each possible location is inhabited by one individual [7]. Under this scheme, the neighbors of a specific individual can be identified, in terms of a *distance function* between each pair of individuals, and a threshold one that

defines the size of the neighborhood. This will be used in order to identify an individual's *deme*; i.e., its possibilities to find reproduction mates.

For instance, consider the case where the population's topology is that of a two-dimensional *torus*. The *distance* between two individuals can, therefore, be set, i according to Von-Newmann, as the set of separate consecutive movements in the progression towards a final location, from a predefined one; i.e., the *d-neighborhood* of an individual may be now understood as being the set of individuals that are located at a distance smaller or equal to d (Figure 3).

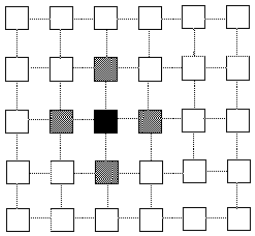

Figure 3. An individual's 1-neighborhood in a 2D-torus.

In terms of implementation, this model implies some major changes on the abstraction levels described in the previous section, namely on the *population's* and on the *GEAs* levels, making use of the original *individuals* classes.

To achieve a spatial structure in the population two new classes have been created, namely the *Space* and *Position* ones. The former enables the definition of different topologies, while the latter handles the individual's spatial distribution within the given topology. The topologies of a *ring* and of a *torus*, 2 or 3-dimensional, are handled by the system; other geometries are possible, if needed.

This approach differs from the one that makes use of panmictic *GEAs*, not only on the spatial distribution of the individuals, but also on the execution flow of the algorithms considered, which relies on a individual based engine; i.e., each individual can be thought to control its own evolution process. Therefore, the major structure of the algorithm is changed, being shown in Figure 4. In each generation, every individual in the population collects its neighborhood, selects from this pool its reproduction mates, applies the genetic operator and creates offspring. At this stage, an individual must decide on its own replacing offspring, and if such operation is or is not advantageous.

There are some considerations that must be taken into account when using this model. The tasks of collecting and selecting among neighbors, as well as the application of the genetic operators, repeated several times (per iteration) lead to an increased computational demand on the part of the *GEA*. So, one is lead to the necessity of simplifying the selection procedures in order to maintain an

```
BEGIN
  Initialize time (t = 0).
  Generate the individuals of the initial population (P_0).
  Assign a spatial position and evaluate each individual in P_0.
  WHILE NOT (termination criteria) DO
    FOR EACH individual DO
      Select the genetic operator to be applied.
      IF (number of incoming individuals in the operator is more than one)
        Collect neighbors.
        Select individual(s) from the neighborhood for reproduction.
      END IF
      Apply the genetic operator, breeding the offspring.
      Select individual from the offspring to replace current individual.
      Decide if new individual replaces its parent.
    END FOR
    Increase current time (t = t + 1).
  END WHILE
END
```

Figure 4. Structure of a *Neighborhood Model GEA*

acceptable computational behavior. A good alternative to this task may rely on the use of the *tournament based* selection procedures [2].

Since this model has specific properties that differ from the ones observed when using the panmictic *GEA* approach, there is an increase in complexity, brought in by the use of additional parameters. These include the topology used, the distance that defines the neighborhood, the method used to select the offspring or the decision procedure that rules the parent's replacement by its offspring or its maintenance in the population. Two options were considered, namely that of always replacing the old individual by the offspring, or proceed with the replacement's operation if it is the case that the offspring presents a better fitness value.

Due to the use of the object-oriented paradigm, the implementation took advantage of numerous features of the *GEA*'s implementation, namely the selection procedures, the evaluation module, the genetic representations and the genetic operators.

4 The Island Model

The *Island Model* considers several sub-populations, evolving independently and exchanging individuals (*migrants*). The islands are connected by *channels*, through which individuals may move from one island to another. The *topology* defines the connection architecture of the system; i.e., which sub-populations can exchange individuals. The topology is defined by the programmer, which has the option of selecting an existing one, such as those of a ring or a fully connected scheme, or to create a new one.

Each island is home to one panmictic *GEA*, although the setup of the different *GEAs* is independent, being possible to have heterogeneous populations evolving simultaneously (e.g., *GEAs* with different genetic operators). The overall structure of the *Island Model GEA* is depicted in Figure 5.

A class, called *server*, was implemented in order not only to control the islander's evolution, but also to manage the migration, to collect the local statistics, and finally to generate the global ones.

```
BEGIN
  Initialize time (t = 0).
  FOR (each island)
    Generate and evaluate the individuals on the initial population.
  END FOR
  WHILE NOT (termination criteria) DO
    FOR (each island)
      Run GEA for mi generations.
    END FOR
    Exchange individuals between islands using channels.
    Collect statistics from each island and generate global statistics.
    Increase current time (t = t + mi).
  END WHILE
END
```

Figure 5. Structure of an *Island Model GEA*

The exchange of individuals (migrations) occurs at fixed intervals, every mi generations, when a certain number of individuals goes through the existing channels in the topology. The *migration interval* and the exact number of individuals that migrates, through each channel are user defined parameter.

It is believed that exchanging individuals often and in large quantities accelerates the overall convergence of the *GEA* On the other hand, it contributes to a greater risk of being stuck in a local optimum. The opposite limits the speed of convergence and degrades the system's performance. So, a tradeoff must be achieved in order to find an equilibrium.

Two different ways of exchanging genetic material are allowed: *migration* and *pollination*. In *migration*, the individuals are moved from one island to another, while in *pollination* the information is simply copied and the individual is maintained in its original population. The user can decide what kind of exchange is more suitable to a specific problem. The programmer may also define how to select which individuals are the *migrants*, using one of the selection procedures inherited from the panmictic *GEAs* (e.g. *Roulette-Wheel*, *Tournament*, etc.).

5 A Distributed Implementation of the Island Model

Each of the two models here presented of *GEAs* with a spatial structure can be carried out either on a sequential or on a distributed architecture. In this work, and apart from the sequential implementation of both models, a distributed view of the *Island Model* was developed. It assumes different *islands*, that may reside in different computers, that cooperate in the search of the best solution to a given problem, by exchanging information (i.e., individuals) through the network. The environment includes a number of workstations and personal computers, with *Linux* or *Windows-95/98* operating systems, connected through an *Ethernet* local network with a bandwidth of 10 Mbit/s.

This implementation uses the *Parallel Virtual Machine (PVM)* [1] technology, which reduces the difficulty of implementing a distributed processing system, while having a reliable communication's infrastructure. It simulates a single virtual space in which several processes are executing, possibly in different machines (Figure 6), providing an ordered, asynchronous and reliable exchange of messages among them, in a anycast way. It is possible to have processes in different machines with different operating systems exchanging messages.

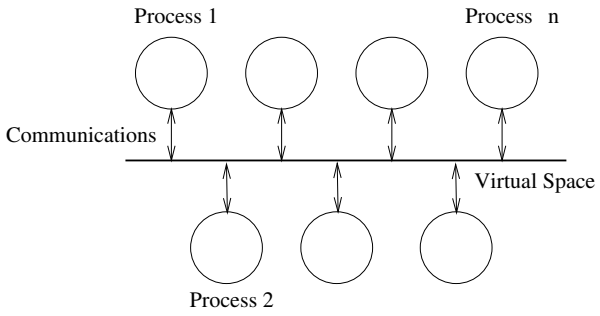

Figure 6. A scheme of the *PVM* environment.

The distributed implementation comprises two new classes, namely the *Server* and the *Island* ones. The former controls the whole process, by instructing the islands to proceed with their evolution a given amount of time (or generations), collecting local results, calculating global statistics and handling error situations. The latter implements the clients, by providing the interface between the *GEA*, that makes its core, and the virtual space, handling the communication with other islands and with the server (Figure 7).

The physical location of the islands and server processes is transparent, allowing the execution of the server in one computer, and of one or more islands in each computer situated in the local network. Thus, the use of the available resources can be maximized.

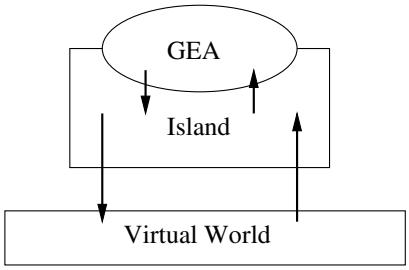

Figure 7. A representation of an *Island* object.

The configuration of the system follows what was said for the *Island Model* referred to above. The ultimate purpose is to achieve the same, or better, results when compared to those of the panmictic *GEA*, while greatly reducing the computational time.

6 Conclusions and Future Work

The proposed genetic and evolutionary programming environment, in its first release, shows itself as a powerful programming tool that makes easier the development of *GEAs* based applications. Furthermore, the system provided high connectivity, both with the problem's data structures making possible the development of hybrid approaches, and with other problem solving used in the intelligent system's design. An example of such endeavor is found in the combination achieved between *GEAs* and *ANNs*, both for *Machine Learning* tasks [10] and for *Time Series Forecasting* [5].

With the introduction of spatially structured populations, a new path towards better performances was revealed, allowing both for the better handling of the diversity in the populations and for the equation of models for the physical parallelization of *GEAs*, thus increasing their computational efficiency.

The implementation of the whole system followed the same methodology, giving a special attention to modularity, incremental development and reutilization, but never forgetting computational efficiency. The user with few knowledge in the area was remembered as well as the expert researcher, and the system fits the needs of both equally well. Therefore, the environment makes a very powerful tool for the programming of robust and efficient *GEAs*.

In the future one intends to extend the physical implementation in order to consider ways of dynamic load balancing, i.e., by assigning more work to the more powerful and more available computation nodes. Furthermore, one intends to study in detail the migration parameters [4] and the configuration of the *GEAs* in each *islands*, taking advantage on the flexibility of the implementation to achieve a better performance. The idea of developing a system that makes the migrations depend on each island's diversity, measured by the fitness standard

deviation, is also a topic under study [8], as well as the behavior of heterogenous *islands*, enhanced for exploration or for exploitation of the search space by using different genetic operators [6]. Finally, the fact that the framework is highly integrated makes easy to design hybrid parallel systems, that mix both models. The idea would is to consider an *Island Model* at a top level, where each island would be a *Neighborhood Model GEA*.

Acknowledgements

The work of José Neves was supported by the PRAXIS' project PRAXIS/P/EEI/13096/98.

References

1. A.Geist, A.Beguelin, J.Dongarra, W.Jiang, R.Manchek, and V.Sunderam. *PVM: Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.
2. J.E. Baker. Reducing Bias and Inefficiency in the Selection Algorithm. In J.Grenfenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*. Lawrence Erlbaum Associates, 1987.
3. E. Cantu-Paz. A survey of parallel genetic algorithms. IlliGAL Report 97003, University of Ilinois at Urbana-Champaign, Urbana, IL, may 1997.
4. E. Cantu-Paz. Migration policies, selection pressure, and parallel genetic algorithms. IlliGAL Report 99015, University of Ilinois at Urbana-Champaign, Urbana, IL, jun 1999.
5. P. Cortez, M. Rocha, J. Machado, and J. Neves. An evolutionary and connectionist approach for time series forecasting. In *Proceedings of Thirteenth International Conference on i Systems Engineering - ICSE 99*, Las Vegas, USA, aug 1999.
6. Francisco Herrera and Manuel Lozano. Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43–63, apr 2000.
7. H. Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337. Morgan-Kaufman, 1991.
8. Masaharu Munetomo, Yoshiaki Takai, and Yoshiharu Sato. An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithms. Technical Report HIER-IS-9301, Hokkaido University, 1993.
9. J. Neves, M. Rocha, H. Rodrigues, M. Biscaia, and J. Alves. Adaptive Strategies and the Design of Evolutionary Applications. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99)*, Orlando, Florida, USA, 1999.
10. M. Rocha, P. Cortez, and J. Neves. The relationship between learning and evolution in static and in dynamic environments. In C. Fyfe, editor, *Proceedings of the 2nd ICSC Symposium on Engineering of Intelligent Systems (EIS'2000)*, pages 377–383. ICSC Academic Press, 2000.
11. M. Rocha, C. Vilela, P. Cortez, and J. Neves. Viewing scheduling problems through genetic and evolutionary algorithms. In *Proceedings of the BioSP3 workshop*, 2000.
12. M. Rocha, C. Vilela, and J. Neves. A study of order based genetic and evolutionary algorithms in combinatorial optimization problems. In R. Loganantharaj, G. Palm, and M. Ali, editors, *Proceedings of the 13th IEA/AIE'2000*. Springer, 2000.