A Jxta based asynchronous Peer-to-Peer Implementation of Genetic Programming

Gianluigi Folino, Agostino Forestiero and Giandomenico Spezzano
Institute for High Performance Computing and Networking (ICAR)-CNR Rende, Italy
Email:{folino, forestiero, spezzano}@icar.cnr.it

Abstract— **Solving complex real-world problems using evolutionary computation is a CPU time-consuming task that requires a large amount of computational resources. Peer-to-Peer (P2P) computing has recently revealed as a powerful way to harness these resources and efficiently deal with such problems. In this paper, we present P-CAGE: a P2P environment for Genetic Programming based on the JXTA protocols. P-CAGE is based on a hybrid multi-island model that combines the island model with the cellular model. Each island adopts a cellular model and the migration occurs between neighboring peers placed in a virtual ring topology. Three different termination criteria (*effort*, *time* and *max-gen*) have been implemented. Experiments were conducted on some popular benchmarks and scalability, accuracy and the effect of migration have been studied. Performance are at least comparable with classical distributed models, retaining the obvious advantages in terms of decentralization, fault tolerance and scalability of P2P systems. We also demonstrated the important effect of migration in accelerating the convergence.**

I. Introduction

Peer-to-peer (P2P) computing is attracting attention in research and industry, spurred also by the popularity of file sharing systems such as Napster, Gnutella, and Morpheus. Peers operate autonomously and asynchronously and perform well in a decentralized environment, as they reuse existing architecture, guarantee interoperability and can exploit the availability of networks resources. P2P networks are emerging as a new distributed computing paradigm for their potential to harness the computing power of the hosts composing the network and make their under-utilized resources available to others.

Genetic programming (GP) is an extension of genetic algorithms (GAs) that iteratively evolves a population of trees having variable size, by applying variation operators. Each individual encodes a candidate solution and is associated with a fitness value that measures the goodness-of-fit of that solution. The capability of GP in solving challenging problems, coming from different application domains, has been largely recognized, but for difficult problems GP requires large sizes of population and a sufficient number of generations. The necessity of high computational resources, both in terms of memory, to store large populations of trees, and in terms of time, to evaluate the fitness of the individuals in the population, may degrade GP performance drastically or make the algorithm inapplicable when it must cope with large difficult problems. The use of P2P resources may make large problems affordable for GP, by using large populations distributed around the network.

While P2P implementations of genetic algorithms are present in literature [30] [18], to the best of our knowledge, there are no P2P implementations of GP. Some efforts in this direction have already been made in the papers [10] [32] in which asynchronous and distributed GP models adapt to P2P implementations have been addressed. Other steps forward have been made in [13], in which some P2P problems are tackled to the aim of coping with wireless sensor networks.

We have devised a system, called P-CAGE (P2P CellulAr Genetic Environment), that is one of the first P2P implementation of GP. Our system has been developed using JXTA-J2SE libraries, the Java implementation of JXTA protocols [1] that guarantee interoperability, platform independence and ubiquity. JXTA technology is a set of open, generalized P2P protocols that allows any connected device (cell phone to PDA, PC to server) on the network to communicate and collaborate.

P-CAGE is based on a hybrid variation of the classic multi-island model that leads not only to a faster algorithm, but also to superior numerical performance. This hybrid model combines the island model with the cellular model. The island model is based on subpopulations, that are created by dividing the original population into disjunctive subsets of individuals, usually of the same size. Each subpopulation can be assigned to one processor and a standard (panmictic) GP algorithm is executed on it. Occasionally, a migration process between sub-population is carried out after a fixed number of generations. For example, the k best individuals from one subpopulation are copied to the other subpopulations exchanging the genetic information among populations. Our hybrid model modifies the island model by substituting the standard GP algorithm with a cellular GP algorithm. In the cellular model each individual has a spatial location, a small neighborhood and interacts only within its neighborhood. The main difference in a cellular GP, with respect to a panmictic algorithm, is its decentralized selection mechanism and the genetic operators (crossover, mutation) adopted. In P-CAGE, to take advantage of the cellular model of GP, the cellular islands are not independently evolved, but the outmost individuals are asynchronously exchanged so that all islands can be thought as portions of a single population. P-CAGE distributes the evolutionary processes (islands) that implement the detection models over the network nodes using a virtual ring configuration. P-CAGE implements the hybrid model as a collection of cooperative autonomous islands running on the various hosts within a heterogeneous network that works as a P2P system. P-CAGE supports many advantages of P2P systems. It is asynchronous, scalable, fault tolerant and maintains the accuracy of other distributed approaches of GP.

The paper is organized as follows: in section II and III, we supply a description respectively of Genetic Programming and of its parallel and distributed implementations; in section IV, we show the most related approaches of parallel GP and in section V we present the architecture of our peer-to-peer implementation of GP and some details about its implementation; section VI presents the experimental results including a comparison with other parallel approaches and an analysis of the scalability and of the effect of migration. Finally, section VII concludes the paper and illustrates some interesting directions for future works.

II. Genetic Programming

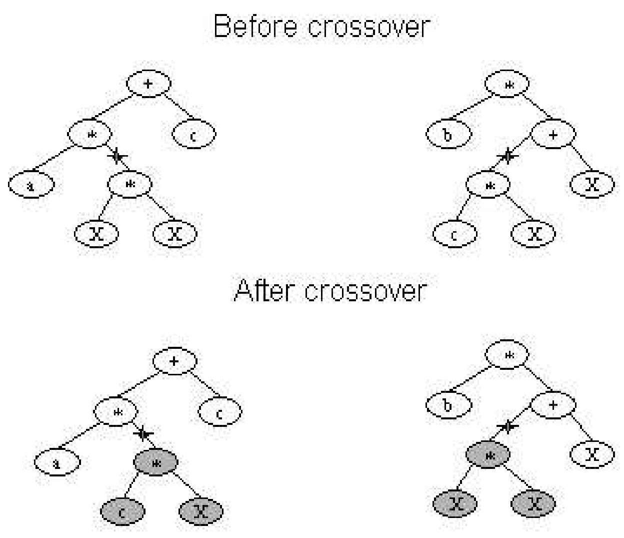


Figure 1. An example of crossover.

Genetic Programming (GP) is a variant of genetic algorithms that mimics the process of natural evolution in order to solve user-defined tasks. GP starts from a population of computer programs and make only best fit programs survive during the evolution. The programmer of the system must choose the functions and the terminals necessary to solve the problem and a fitness function that represents the program's ability to perform the task. The search space will be composed from all the possible programs generated recursively from the functions and the terminals chosen. A computer program (individual) is represented as a parse tree.

Genetic programming uses four main steps to solve problems:

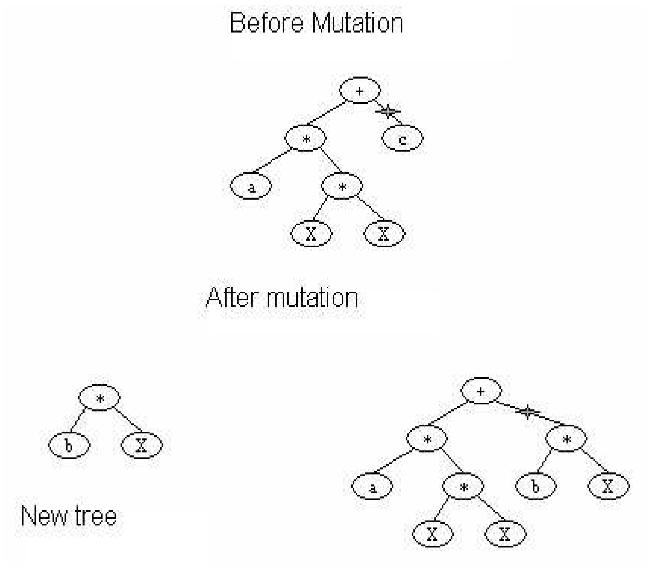


Figure 2. An example of mutation.

1) Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).
2) Execute each program in the population and assign it a fitness value according to how well it solves the problem.
3) Create a new population of computer programs by applying genetic operators (mutation, crossover, etc.) to some selected tree (best fit trees are selected most likely)
4) The best computer program that appeared in any generation, the best-so-far solution, is designated as the result of genetic programming.

Steps 2 and 3 are repeated until a maximum number of generations is reached or a termination criterium is met (problem is solved exactly or error is less than a threshold).

Most frequent operators of GP are mutation and crossover; in figure 1, it is reported an example of crossover applied to two trees representing algebraical expressions; in practise, two randomly chosen crossover points for each tree are taken and the respective subtrees are exchanged. In figure 2, it is reported an example of mutation for the same class of trees. A point is chosen at random, a new random tree is created and it is linked to the point.

III. Parallel and distributed implementations of GP

GP belongs to the class of evolutionary algorithms and, as such, shares the same distributed strategies of evolutionary techniques. A classification of the approaches for distributing *GP* includes three main models [3], [31]: the *global* model, the *coarse-grained* (*island*) model [19], and the *fine-grained* (*grid*) model [24].

In the global model, the fitness of each individual is evaluated in parallel on different processors. A *master*

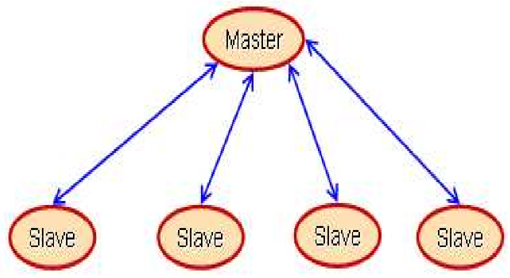


Figure 3. Global model.

process manages the population by assigning a subset of individuals to a number of *slave* processes. During the evaluation there is no communication among the processors. At the end of the evaluation the master collects the results and applies the variation operators to generate the new population. This model is easy to implement but a main problem, as observed in [31], can be a load imbalance, which decreases the utilization of the processors due to presence in the population of trees of different sizes.

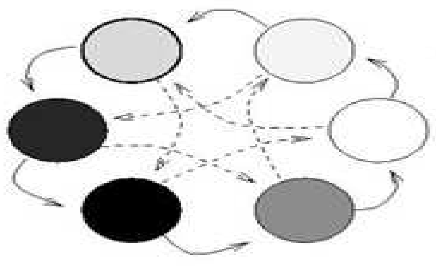


Figure 4. Island model.

The island model divides a population $\mathcal{P}$ of M individuals into N subpopulations $D_1, \ldots, D_N$, called *demes*, of M/N individuals. A standard GP algorithm works on each deme and is responsible for initializing, evaluating, and evolving its own subpopulation. Subpopulations are interconnected according to different *communication topologies* and can exchange information periodically by *migrating* individuals from one subpopulation to another. The number of individuals to migrate (*migration rate*), the number of generations after which migration should occur (*frequency*), the migration topology, and the number of subpopulations are all parameters of the method that must be set.

In the grid model (also called *cellular* [35]) each individual is associated with a spatial location on a low-dimensional grid. The population is considered as a system of active individuals that interact only with their direct neighbors. Different neighborhoods can be defined for the cells. The most common neighborhoods in the two-dimensional case are the 4-neighbor (*von Neumann neighborhood*) consisting of the North, South, East, West neighbors and 8-neighbor (*Moore neighborhood*) consisting of the same neighbors augmented with the diagonal neighbors. In the ideal case, one processor is

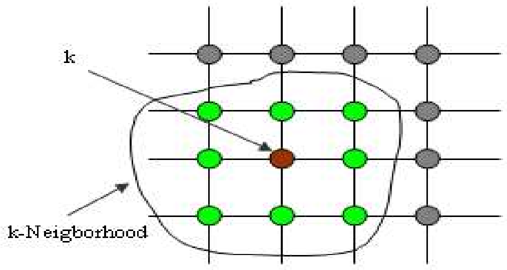


Figure 5. Cellular (grid) model.

assigned to each grid point, thus fitness evaluation is performed simultaneously for all the individuals. In practical implementations, however, this is not true because the number of processors generally does not coincide with the number of points on the grid. Selection, reproduction, and mating take place locally within the neighborhood. Information diffuses slowly across the grid, giving rise to the formation of semi-isolated niches of individuals having similar characteristics. The choice of the individual to mate with the central individual and the replacement of the latter with one of the offspring can be accomplished in several ways.

Ref. [31] noted that parallel genetic algorithms are faster than their sequential counterpart and benefit from the multi population approach in two different aspects. First, the problem of premature convergence is reduced thanks to the spatial isolation of the subpopulations that co-evolve independently but promote local search. Second, the same solution quality can be obtained in fewer generations, by using many populations instead of a single population with the same total number of individuals.

The same results, however, have not been obtained for the coarse-grained parallel implementations of GP. In fact, although Koza [2], [17] reported a super-linear speedup for the 5-parity problem, for other problems, Punch [25] found poorer results of convergence with respect to the canonical GP. Punch [25] was the first to analyze the behavior of distributed GP with respect to sequential GP.

IV. Related Work

The most famous coarse-grained parallel implementation of GP is due to Koza and Andre [2], [17]. They used a PC 486 computer as host and a network of 64 transputers as processing nodes. For the *Even-5 parity* problem, they obtained a super-linear speedup using a population of 32000 individuals, 500 on each node and a migration rate of 8% in the four directions of each subpopulation at each generation.

Stoffel and Spector [28] described a high-performance GP system (HiGP) based on a virtual stack machine (similar to [15]) that executed GP programs represented as fixed-length strings. HiGP manipulated and generated linear programs instead of tree-structured S-expressions. Each gene in a chromosome corresponded to an operator of the virtual machine. They executed 100 runs for a symbolic regression problem for a maximum of 30 generations and obtained good time performances. Nothing, however, was said about convergence results.

Juillé and Pollack [14], [15] described a parallel implementation of GP on a SIMD system. Each SIMD processor simulated a computer program by using a simple instruction set defined specifically for GP. S-expressions were evaluated efficiently, precompiling them in a postfix program. The authors, for few classical problems, reported the execution time for one run and the average execution time for one generation. For $cos2x$, using a population of 4096 individuals, one per processor, they found a solution after an average of 17.5 generations with an average execution time of 30.48 seconds.

Dracopoulos and Kent [4], [5] described two different implementations of parallel GP based on the *Bulk Synchronous Parallel* (BSP) model [34] of parallel computation. The first implementation adopted the master-slave paradigm. A master process performed the standard sequential GP algorithm and the slave processes assisted the master only during the fitness evaluation. The slaves received equal portions of the population from the master, evaluated the fitness of individuals, and returned them back to the master. The second implementation realized the island model. Each process was considered an island and, every 10 generations, the top 10% of individuals were migrated. Two different communication topologies were considered: the ring and the star. The results presented regards only the speedup obtained for the *Artificial Ant Los Altos Hills* problem by running these implementations for 50 generations. The authors experiments showed that the coarse-grained parallel implementation achieved better speedups than the global version. The convergence results of the two approaches were not reported.

Oussaidène et al. [21], [22] presented a parallel implementation of GP on a distributed-memory machine that used the master-slave model. The *Parallel Genetic Programming Scheme* (PGPS) had a master process whose task is to manage the GP algorithm, that is, it created the initial population, applied the variation operators, and performed the selection of the individuals for the reproduction phase. The slave processes controlled fitness evaluation, thus they received the parse trees from the master process to evaluate. The trees were packed as strings in a buffer and sent to the slaves. Each slave process unpacked the buffer content and rebuilt the parse tree in memory. In order to distribute the computational load among the processing nodes equally, two different load balancing algorithms (one static and another one dynamic) were used. The dynamic scheduling algorithm gave better speedup results as compared to the static one. PGPS has been applied to the evolution of trading strategies to infer robust trading models [21], [22].

Niwa and Iba [20] described a parallel implementation of GP, named *Distributed Genetic Programming* (DGP), based on the island model and realized on a MIMD parallel system AP-1000+ consisting of 32 processors. The global population was distributed among the processing nodes, each of which executed a canonical GP on its subpopulation. At every generation, the best individual of a subpopulation was sent asynchronously to its adjacent subpopulations and, for each subpopulation, the worst individual was replaced by this one if the fitness of the received individual was better than the best individual of the current subpopulation. The authors used three different communication topologies: ring type, one-way torus, and two-way torus. Experimental results on three problems (*discovery of trigonometric identities, predicting a chaotic time series and Boolean concept formation*) revealed the ring topology as the best.

Salhi et al. [26] reported a parallel implementation of GP based on a random island model, designed specifically for symbolic regression problems. In such a model, individuals migrate at random. This is possible because two new operators, $import$ and $export$ are introduced. They have the role of supporting communication among the islands and have associated a probability like the other variation operators. For two symbolic regression problems the authors obtained a superlinear speedup, and for $cos2x$ they obtained a linear speedup. Convergence results were not reported.

Punch [25] discussed the conflicting results on the use of multiple populations in GP, in contrast with the indisputable benefits obtained in genetic algorithms with the same approach. He argued that there are problem-specific factors that affect the multiple-population approach. He presented experiments for the *Ant Santa Fe* and the *royal tree* problem.

Tongchim and Chongstitvatana [33] presented a coarse-grained parallel implementation of GP with asynchronous migration and applied it on a mobile robot navigation problem. They obtained a superlinear speedup by using a population size of 6000 individuals, while migrating the top 5% of individuals of each subpopulation with a frequency depending on the number of processors used.

Fernández et al. [7] presented an experimental study to verify the influence of two parameters, number of subpopulations and size of each population, on the performances of parallel genetic programming. A standard GP tool was suitably modified to allow the coarse-grained parallelization of GP. The tool, described in [6], used communication primitives of the PVM (*Parallel Virtual Machine*) and adopted a client/server model where the server has the task of managing input/output buffers and of choosing the communication topology (that can be dynamically changed), while the clients constitute the subpopulations. The results reported for the Even-5 parity and a regression problem were evaluated with respect to the number of nodes evaluated in a GP tree, called computational effort. For these two problems they found optimal ranges for parameter values. Such values, however, are problem dependent.

An improvement of the tool described in [6] was presented in [9] and consisted of a parallel GP kernel that used MPI (*Message Passing Interface*) message passing system, and a graphical-user interface. The communication between the processes/subpopulations and the master process was synchronous. A new communication topology, the random one, among the subpopulations, was added to the ring and mesh topologies. In the random topology, the master process received a block of individuals and sent them to a randomly chosen subpopulation. This software tool was used in [8] to study the influence of the communication topology and the frequency of migration on the performances of parallel GP. Three test problems were considered: Even-5 parity, Ant Santa Fe and a real world problem. The authors found that the random and ring topology were better than the mesh for the ant problem. For the Even-5 parity, if the population size was large, the grid was the best, while, if the population size was small, the ring and the random were better. With regard to migration, a number of individuals of about 10% the population size, every 5-10 generations, appeared to be the best values for all the problems considered.

In [23], the authors try to cope with the problem of addressing the setting of additional parameters introduced in the distributed models of Genetic Programming; they used a kind of adaptive flocking for automatizing the migration rate parameter. Early experiments seem promising.

V. P-CAGE: A P2P IMPLEMENTATION OF GP

In this section we describe the architecture of P-CAGE and the algorithm executed by each peer to implement cellular GP.

A. P-CAGE Architecture

As showed in figure 6, P-CAGE exploits the potentialities of p2p network using the main services of JXTA protocols [1]. These supplies interoperability, platform independence and ubiquity.

JXTA technology is a set of open, generalized P2P protocols that allows any connected device (cell phone to PDA, PC to server) on the network to communicate and collaborate. Each device is represented as a peer that can be a minimal edge peer (typically a phone or a PDA), a full-featured edge peer, a rendezvous peer (responsible for processing and delegate queries to other rendevous peers) and a relay peer (for reaching peers that are behind firewalls or NAT systems). In JXTA, any resource can be modeled as an advertisement, a XML document that can be exchanged among the peers. Our system uses the JXTA discovery protocols to discover peers that want to participate to the computation, the membership protocols to create a common group for peers composing the virtual GP network and pipe protocols to create communications channels between neighboring peers. Our application is built on the top of JXTA protocols and comprises also a graphical user interface.

Peers are arranged on a self-configuring ring topology architecture, see figure 7, so they need to know only information on the left (previous) and right (next) neighbor. Each peer constitutes a cellular model of Genetic Programming.

B. Implementation issues

Each peer adopts the GP cellular approach described in [12] and performs the algorithm showed in figure 8.

```
$find\_peers(); build\_ring(); build\_pipes();$
$create\_populations();$
$gen = 0;$
$while(!termination\_criterion())\{$
        $evolve\_population(gen);$
        $evaluate\_fitness();$
        $if(gen\%gen\_migration == 0)$
                $send\_borders(); asinc\_receive\_borders();$
        $gen++;$
$\}$
$report\_on\_run();$
```

Figure 8. The pseudo code executed by each peer (termination criteria and fault tolerance strategy not reported).

In practice, first it discovers new peers, then it builds the virtual ring topology, conveniently choosing the left and right neighbor and connecting the neighboring peers by bidirectional channels. After the topology is established, each peer executes a certain number of iterations exchanging, at prefixed generations, the borders of the sub-population, in an asynchronous fashion, until a termination criterion is achieved.

In the following, the different phases of the algorithm are described more in detail.

Finding peers and building the ring:

The ideas to build the virtual ring are inspired to the strategies described in [27]. The algorithm uses two parameters *min_peers* and *max_peers* to define respectively a minimum and a maximum number of peers forming the ring. At the beginning, each peer creates and joins a group called ”GeneticProgramming” using the JXTA *membership service* and starts the search of peers belonging to this group (JXTA *discovery service*), until a timeout is expired or the maximum number of peers is reached. The time of discovery of each peer is stored. At this point, information is distributed to all the peers discovered and the peers are ordered by minimum time of discovery. The order allows to identify the *left* and the *right* neighbor of each peer, respectively the previous and following in the ordered sequence. The first peer is named *leader* of the group. If the minimum number of peers is not reached, the leader tries to discover other leaders and two or more groups of peers are joined, following the schema illustrated in figure 9 until the *min_peers* number is reached.

Building pipe and finger table: Each peer creates a bidirectional channel with its left neighbor and one with its right neighbor using the pipe service of JXTA. To make the system robust a finger table is built storing k left and k right successors, where k typically assumes a value from 2 to 4. If a pipe falls down, the peer creates a new one using the information contained in the finger table, i.e. the table is checked until a peer is found and then linked with a pipe.

Evolving population: In the evolving phase, a population of trees with dimension $a \times b$ evolves on each peer,

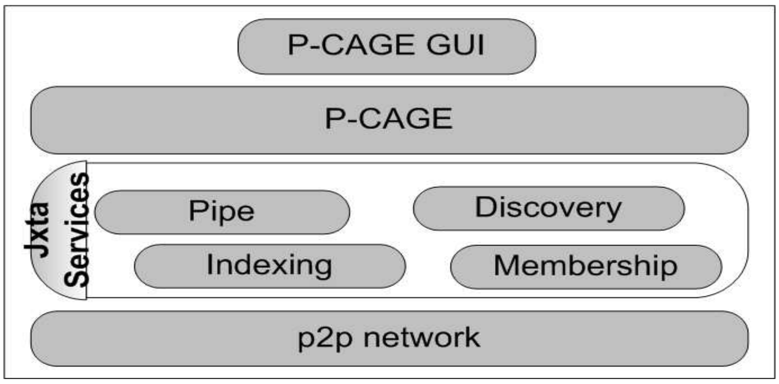


Figure 6. P-CAGE application overview.

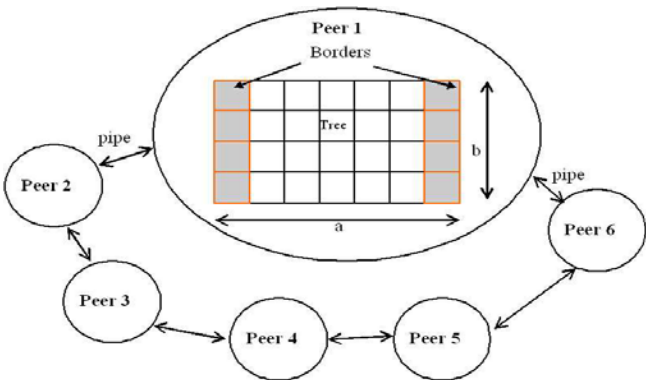


Figure 7. Architecture of P-CAGE.

following the hybrid multi-island model adopted by P-CAGE (in practice, each peer represents an island adopting the cellular model as CAGE). Every k (gen_migration parameter) generations, the borders of the populations (the outside b trees on the left and on the right side) are exchanged among neighboring peers using the pipes, in an asynchronous fashion (i.e. if borders do not arrive to a peer before starting the computation, the peer goes on with the old trees).

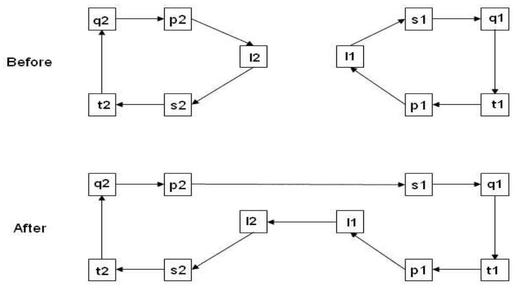


Figure 9. Merging two groups by means of leader l1 and l2.

Termination criteria: The algorithm terminates its execution, when a termination criterion is met, chosen among the following : *max-gen*, *time* and *effort*.

Using the first criterion, P-CAGE ends when each peer has reached the maximum number of generations. This criterion is easy to implement, but faster peers must wait slower ones. A timeout is introduced to avoid infinite waiting for peers that have gone down.

Using a time-based termination criterion, a maximum time is fixed and, when this time expires, all the peers must terminate. No synchronization is required, but it is difficult to estimate the right time, also considering the different loads and speeds of the computers involved in the computation. In fact, if we choose a time too short, the algorithm could not converge.

Last choice considers the effort, derived from the definition given in [11]. Let G be the number of generations, N the number of individuals of the population lying on each peer and AVG_LEN_p the average number of nodes per individual concerning peer p. Then the required effort in a particular generation n is:

$$\sum_{p=1}^{n} G \times N \times AVG\_LENGTH_p \qquad (1)$$

Effort takes into account the overall effective computation carried out by the system. However, it requires an overhead necessary to compute the effort and to communicate the information; in fact, for each prefixed period of time, the total effort must be computed summing the efforts produced by each peer.

VI. Experimental results

In order to verify the goodness of our implementation, we compared, in terms of accuracy, our asynchronous hybrid multi-island model to a canonical sequential implementation of GP and to a parallel cellular implementation of GP.

We used sgpc1.1 [29] as the sequential implementation of genetic programming and CAGE [12] as the parallel cellular implementation of GP. The peer-to-peer experiments were conducted on the LAN of the CNR-ICAR Institute. All the experiments were run for 100 generations and averaged over 20 tries.

A. Benchmark

We have used the following problems as benchmark:

Symbolic Regression ([16] par. 7.3). The *symbolic regression* problem consists in searching for a non-trivial mathematical expression that, given a set of values x_i for the independent variable(s), always assumes the corresponding value y_i for the dependent variable(s) of a given mathematical function. In the first experiment, the target function was the polynomial $x^4 + x^3 + x^2 + x$. A sample of 20 data points (x_i, y_i) was generated by randomly choosing the values of the independent variable x in the interval [-1,1].

Discovery of trigonometric identities. ([16] par. 10.1) In the second experiment, our aim was to discover a trigonometric identity for $cos2x$. 20 values x_i of the independent variable x were chosen randomly in the interval [0,2π] and the corresponding value $y_i = cos2x_i$ computed. The 20 pairs (x_i, y_i) constituted the fitness cases. The fitness was then computed as the sum of the absolute value of the difference between y_i and the value generated by the program on x_i.

Symbolic Integration. ([16] par. 10.5) The symbolic integration problem consists in searching for a symbolic mathematical expression that is the integral of a given curve. In this experiment the curve was $cosx + 2x + 1$ so the genetic program had to obtain $sinx + x^2 + x$, given 50 pairs (x_i, y_i) in the interval [0,2π].

Even-4 parity. ([16] par. 20.4) The *Even-4 parity* problem consists in deciding the parity of a set of 4 bits. A Boolean function receives 4 Boolean variables and it returns TRUE only if an even number of variables is true. Thus the goal function to discover is $f(x_1, x_2, x_3, x_4)$ = $x_1x_2x_3x_4 \vee \overline{x_1}x_2\overline{x_3}x_4 \vee \overline{x_1}x_2x_3\overline{x_4} \vee x_1\overline{x_2}\overline{x_3}x_4 \vee x_1\overline{x_2}x_3\overline{x_4} \vee x_1x_2\overline{x_3}\overline{x_4} \vee \overline{x_1}\ \overline{x_2}\ \overline{x_3}\ \overline{x_4}$. The fitness cases explored were the 2^4 combinations of the variables. The fitness was the sum of the Hamming distances between the goal function and the solution found. **Even-5 parity.** The problem is the analogue of even-4 considering 5 bit instead of 4.

Ant Santa Fe. ([16] par. 7.2) The *artificial ant* problem consists in finding the best list of moves that an ant can execute on a 32×32 matrix in order to eat all the pieces of food put on the grid. In this experiment we used the *Santa Fe trail* that contains 89 food particles. The fitness function was obtained by diminishing the number of food particles by one every time the ant arrived in a cell containing food. The ant can see the food only if it is in the cell ahead in its same direction ($IfFoodAhead$ move); otherwise it can move randomly ($left$ or $right$) for two ($Progn2$) or three ($Progn3$) moves.

This last problem is known to be difficult for distributed GP.

The parameters of the method are shown in table I; functions and terminal symbols for each problem are the same described in Koza's book [16]. For all the experiments, we used an overall population size of 3200, except for *Symbolic Regression*. For this problem, the size of the population was set to 800 individuals. In the parallel implementation, the population size was divided equally among the nodes and the same was made in the peer-to-peer implementation among the peers. In this two last cases the $Moore$ neighborhood was adopted. For P-CAGE, we used 5 peers, migration occurred every 5 generations and the termination criterion was max-gen.

TABLE I.
GP PARAMETERS

Maximum number of generations	100
Probability of crossover	0.8
Probability of choosing internal points for crossover	0.1
Probability of mutation	0.1
Probability of reproduction	0.1
Generative Method for initial random population	Ramped
Maximum depth for a new tree	6
Max depth for a tree after crossover	17
Max depth of a tree for mutation	4
Parsimony factor	0.0

B. Accuracy

In the figures 10 (a) and (b), 11 (a) and (b) and 12 (a) and (b) the results of our experiments are reported respectively for symbolic regression, discoveries of trigonometric identities, symbolic integration, ant Santa Fe and even 4 and 5 parity problems.

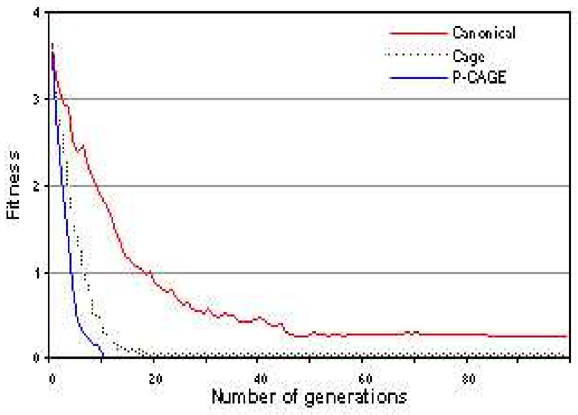
Observing these figures, we can notice that P-CAGE outperforms canonical GP for all the problems and it is better or at least comparable with the parallel cellular implementation. The accuracy of P-CAGE accuracy is considerably better for the even 4 parity problem, it is lightly better for the discovery of trigonometric identities and the symbolic regression problem, and is lightly worst only for the Ant Santa Fe problem.

C. Scalability

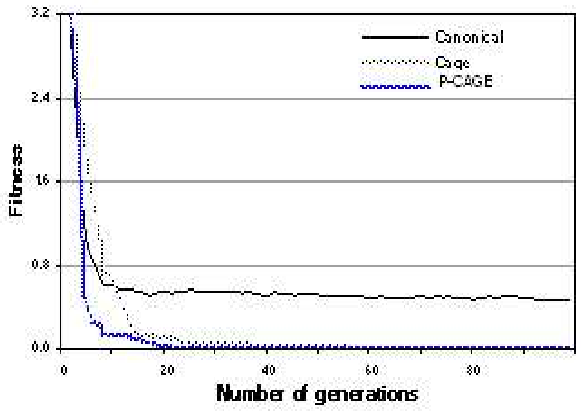
To study the scalability of our system, we ran our system, using the same parameters previously described, varying the number of peers (3, 5, 10) and maintaining the same overall population. As you can see in figures 13 (a) and (b), 14 (a) and (b), 15 (a) and (b), differences are not statistically significant or are minimal. The only exception is the even-parity problem (4 and 5) in which the experiments on three peers converges slightly earlier probably because in the 10 peers' case, population on each peer is not adequate.

D. Migration Effect

We want to very the effect of the migration on the convergence of the algorithm. In order to do it, we ran

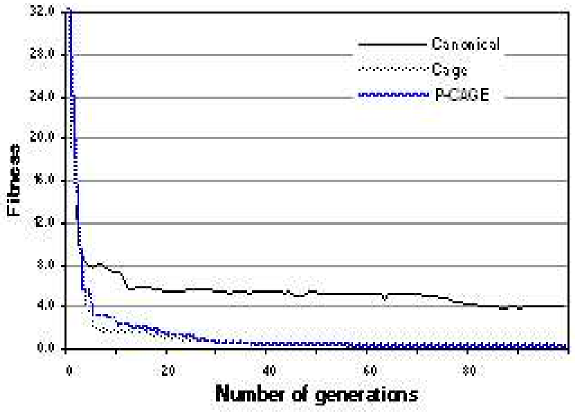


(*a*)

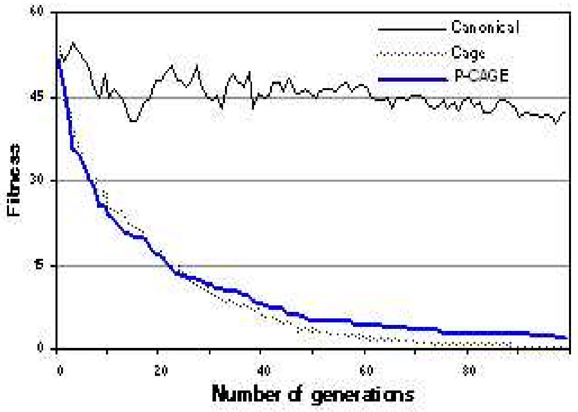


(*b*)

Figure 10. Accuracy comparison for a) symbolic regression and b) discovery of trigonometric identities: Canonical GP, CAGE and P-CAGE.

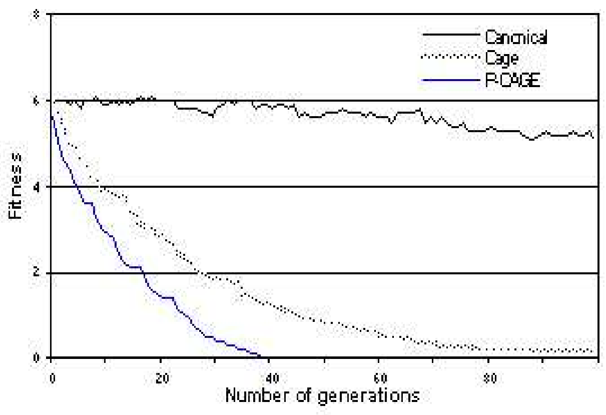


(*a*)

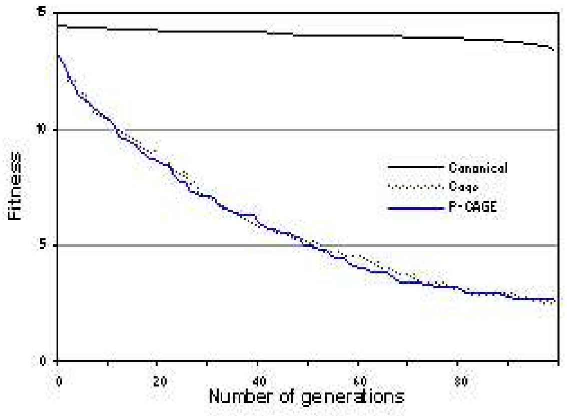


(*b*)

Figure 11. Accuracy comparison for a) symbolic integration and b) Ant Santa Fe: Canonical GP, CAGE and P-CAGE.



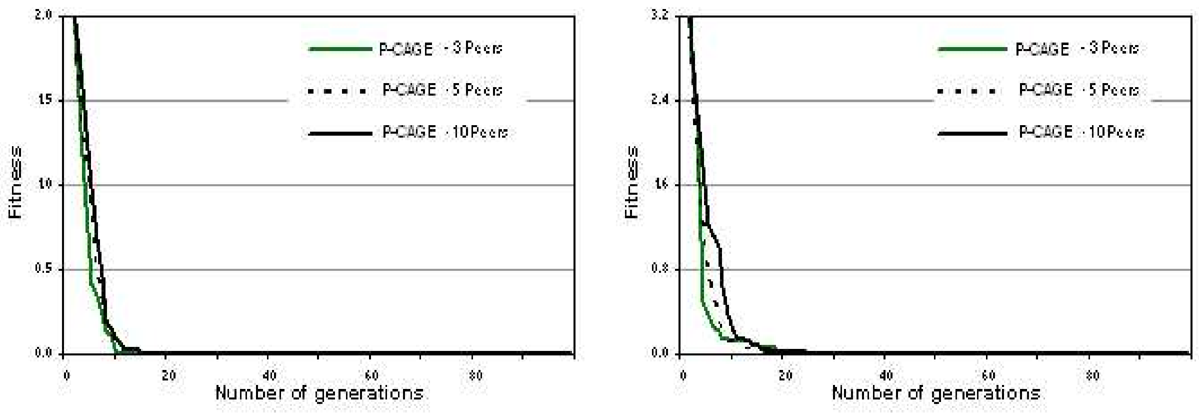
(*a*)



(*b*)

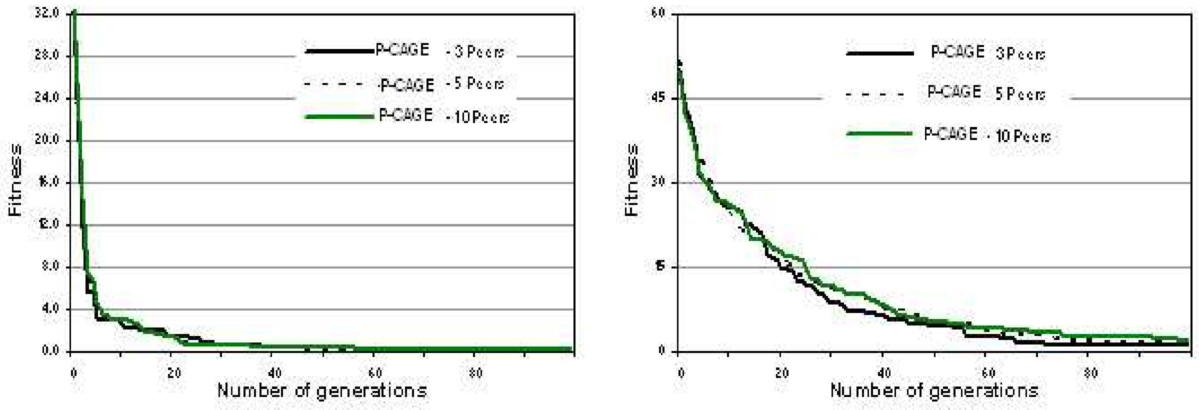
Figure 12. Accuracy comparison for a) Even 4 parity and b) Even 5 parity: Canonical GP, CAGE and P-CAGE.

our system, using the same parameters of the previous subsections, on 10 peers and varying the rate of migration. We conducted experiments without using migration (i.e. each peer evolves independently from the others), migrating the borders each generation and every 5 generations. The results of these simulations for the six benchmarks are reported in figures 16 (a) and (b), 17 (a) and (b), 18 (a) and (b). If we does not use migration, the accuracy is appreciably worst than in the other two cases for all the benchmarks, especially for the most hard, as Ant, Even and Even 5. Comparing the curves for each generation and for each 5 generation, we can note that the accuracy is almost equal, especially after 40-60 generations. In order to avoid to deteriorate the performance of the network,



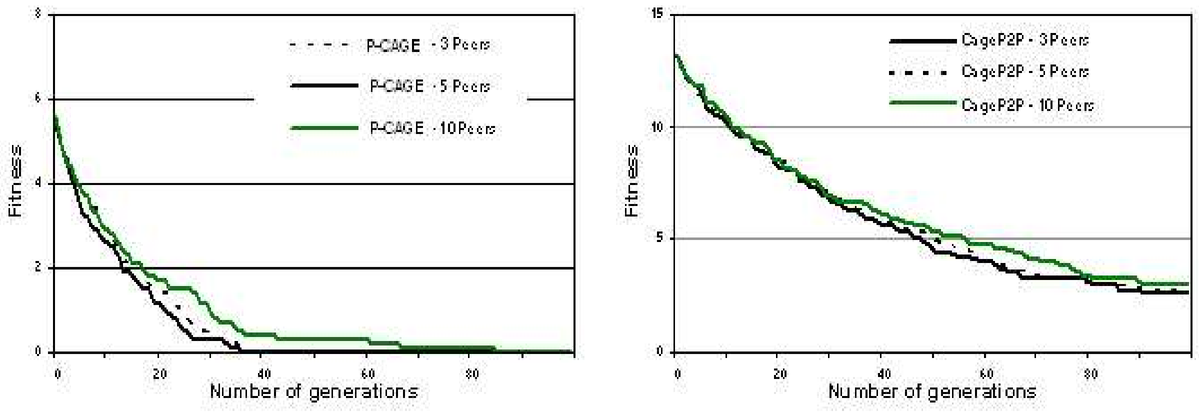
(*a*) (*b*)

Figure 13. Scalability for (a) symbolic regression and (b) discovery of trigonometric identities: 3, 5 and 10 peers.



(*a*) (*b*)

Figure 14. Scalability for (a) symbolic integration and (b) Ant Santa Fe: 3, 5 and 10 peers.



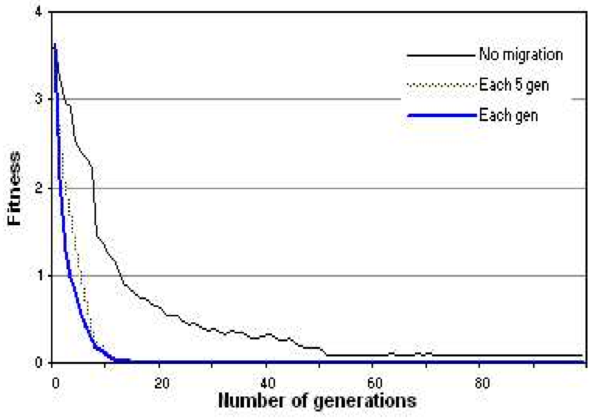
(*a*) (*b*)

Figure 15. Scalability for (a) Even 4 parity and (b) Even 5 parity: 3, 5 and 10 peers.

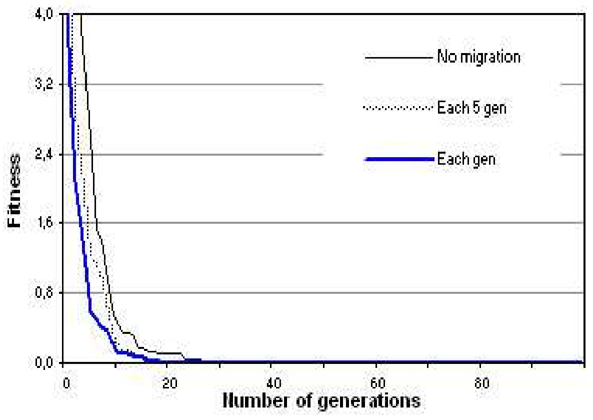
is preferable to use a lower migration rate, therefore a migration rate of 5 seems to be appropriate.

VII. Conclusions and future works

In this paper, P-CAGE, an environment for the execution of genetic programs in P2P networks, has been presented. Each peer adopts a cellular model and the migration occurs between the neighboring peers, distributed in a virtual ring topology. Experiments on a real network demonstrated that P-CAGE show a superior performance in comparison with the canonical GP implementation, as expected, and at least comparable with the most known cellular implementations of GP. Anyway, it exploits the advantages of P2P networks as it is fault tolerant, inex-

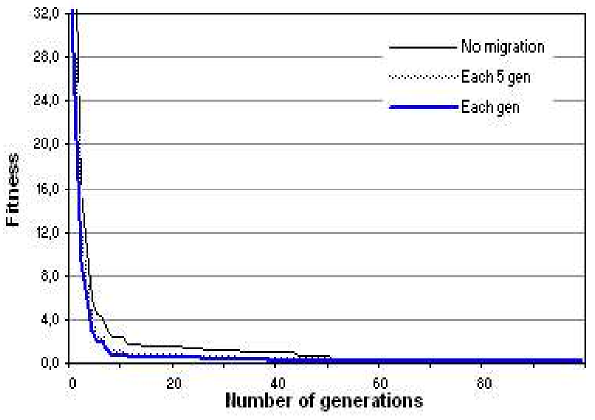


(a)

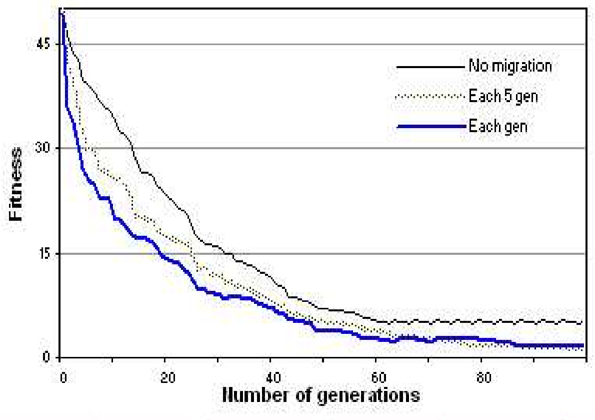


(b)

Figure 16. The effect of migration for (a) symbolic regression and (b) discovery of trigonometric identities: no migration, migration each 5 generation and migration each generation.

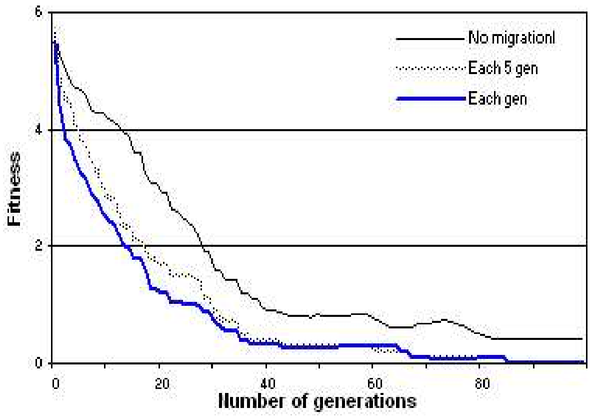


(a)

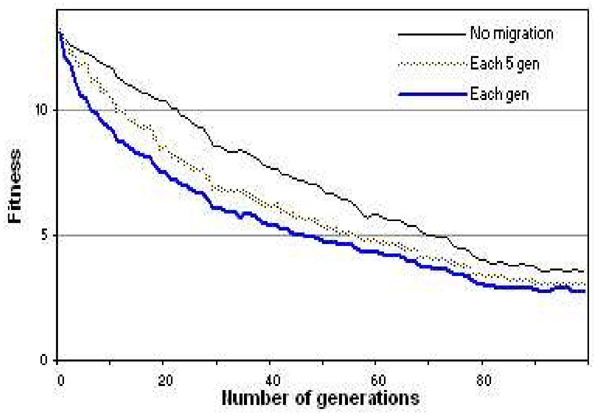


(b)

Figure 17. The effect of migration for (a) symbolic integration and (b) Ant Santa Fe: no migration, migration each 5 generation and migration each generation.



(a)



(b)

Figure 18. The effect of migration for (a) even 4 parity and (b) even 5 parity: no migration, migration each 5 generation and migration each generation.

pensive and scalable in terms of resources. Experimental analysis of the scalability showed that the number of peers does not affect the accuracy even if, in some cases, it was excessive for the difficulty of the problem. If we choose an adequate migration rate, the convergence of the algorithm is better than not using migration at all. Future works will investigate the use of different topologies as small world and scale-free networks. These topologies expect to improve the accuracy and to accelerate the convergence.

Gianluigi Folino was born in Catanzaro, Italy, on May 27, 1972. He received his Laurea degree in Engineering from the University of Calabria, Italy, in 1997. In 1999 he joined CNR-ICAR, Institute of High Performance Computing and Networking of the Italian National Research Council, supported by an INFN fellowship. Currently, he is a researcher at CNR-ICAR in the area of distributed and parallel computing. He has published more than 50 papers in conference proceedings and journals and he is member of the program committee for many international conference and reviewer for many international journals as IEEE TEC, Parallel Computing, Parallel and Distributed Computing. His research interests include cellular automata, genetic programming, swarm intelligence, peer-to-peer and grid computing.

Agostino Forestiero received his Laurea degree in Computer Engineering from University of Calabria, Cosenza, Italy, in 2002. He collaborates with the Institute of High Performance Computing and Networks of the Italian National Research Council (ICAR-CNR) in Cosenza, Italy, and since 2003, he is a PhD Student in Computer Engineering at the DEIS Department, University of Calabria. His research interests include Grid Computing, Peer-to-Peer Networks and Multi-Agent Systems.

Giandomenico Spezzano is a research director at the institute of high performance computing and networking (icar) of the italian national research council (cnr), where he manages the intelligent grid and peer-to-peer systems group. He is also a contract professor in the department of electronics, computer science and systems (deis) at the university of calabria, italy, since 1994. Previously, he worked at crai (consortium for research and applications of information technology), Italy, where he has led various research projects in the distributed and parallel computing area. He received the laurea degree in industrial technologies engineering from the university of calabria, italy, in 1980. He has published two books and more than 120 papers in conference proceedings and journals such as ieee tec, ieee cse, fgcs, pdcp, parallel computing, concurrency:practice and experience. his current research interests cover models and tools for massively parallel architecture, grid computing,peer-to-peer computing, parallel and distributed data mining, parallel genetic programming, cellular automata and swarm intelligence. He is serving as a program committee member for many international conference and he is member of ACM and IEEE Computer Society.

References

[1] The jxta project. http://www.jxta.org.

[2] D. Andre and J. R. Koza. Exploiting the fruits of parallelism: An implementation of parallel genetic programming that achieves super-linear performance. *Information Science Journal*, 1997.

[3] E. Cantú-Paz. A summary of research on parallel genetic algorithms. Technical Report 95007, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.

[4] D. C. Dracopoulos and S. Kent. Bulk synchronous parallelisation of genetic programming. In Jerzy Waśniewski, editor, *Applied parallel computing : industrial strength computation and optimization ; Proceedings of the third International Workshop, PARA '96*, pages 216–226, Berlin, Germany, 1996. Springer Verlag.

[5] D. C. Dracopoulos and S. Kent. Speeding up genetic programming: A parallel BSP implementation. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 125–136, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[6] F. Fernández, J. M. Sanchez, M. Tomassini, and J. A. Gomez. A parallel genetic programming tool based on pvm. In J. Dongarra, E. Luque, and T. Margalef, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proceedings of the 6th European PVM/MPI Users' Group Meeting, Barcelona, Spain, September 1999*, number 1697 in Lecture Notes in Computer Science, pages 241–248. Springer-Verlag, September 1999.

[7] F. Fernández, M. Tomassini, W. F. Punch III, and J. M. Sanchez. Experimental study of multipopulation parallel genetic programming. In *Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 283–293, Edinburgh, 15-16 April 2000. Springer-Verlag.

[8] F. Fernández, M. Tomassini, and L. Vanneschi. Studying the influence of communication topology and migration on distributed genetic programming. In *Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 51–63, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.

[9] F. Fernández, M. Tomassini, L. Vanneschi, and L. Bucher. A distributed computing environment for genetic programming using mpi. In J. J. Dongarra, Peter Kacsuk, and Norbert Podhorszki, editors, *Recent advances in parallel virtual machine and message passing interface: 7th European PVM/MPI Users' Group Meeting, Balatonfured, Hungary, September 10–13*, volume 1908 of *Lecture Notes in Computer Science*, pages 322–329, New York, NY, USA, 2000. Springer-Verlag Inc.

[10] Francisco Fernandez, G. Galeano, and J. A. Gomez. Comparing synchronous and asynchronous parallel and distributed GP models. In *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 326–335, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag.

[11] Francisco Fernandez, Marco Tomassini, and Leonardo Vanneschi. An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines*, 4(1):21–51, 2003.

[12] Gianluigi Folino, Clara Pizzuti, and Giandomenico Spezzano. A scalable cellular implementation of parallel genetic programming. *IEEE Transactions on Evolutionary Computation*, 7(1):37–53, February 2003.

[13] Derek M. Johnson, Ankur Teredesai, and Robert T. Saltarelli. Genetic programming in wireless sensor networks. In *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in*

Computer Science, pages 96–107, Lausanne, Switzerland, 30 March - 1 April 2005. Springer.

[14] H. Juille and J. B. Pollack. Parallel genetic programming and fine-grained SIMD architecture. In E. V. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 31–37, MIT, Cambridge, MA, USA, 10–12 November 1995. AAAI.

[15] H. Juille and J. B. Pollack. Massively parallel genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 17, pages 339–358. MIT Press, Cambridge, MA, USA, 1996.

[16] J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[17] J. R. Koza and D. Andre. Parallel genetic programming on a network of transputers. Technical Report CS-TR-95-1542, Stanford University, Department of Computer Science, January 1995.

[18] José Carlos Clemente Litrán, Xavier Défago, and Kenji Satou. Asynchronous peer-to-peer communication for failure resilient distributed genetic algorithms. In *Proc. 15th IASTED Int'l Conf. on Parallel and Distributed Computing and Systems (PDCS)*, volume II, pages 769–773, Marina del Rey, CA, USA, November 2003.

[19] W. N. Martin, J. Lienig, and J. P. Cohoon. Island (migration) models: evolutionary algorithms based on punctuated equilibria. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C6.3:1–16. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

[20] T. Niwa and H. Iba. Distributed genetic programming: Empirical study and analysis. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 339–344, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[21] M. Oussaidène, B. Chopard, O. V. Pictet, and Marco Tomassini. Parallel genetic programming: An application to trading models evolution. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Proceedings of the First Annual Conference on Genetic Programming*, pages 357–380, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[22] M. Oussaidène, B. Chopard, O.V. Pictet, and M. Tomassini. Parallel genetic programming and its application to trading model induction. *Parallel Computing*, 23:1183–1198, 1997.

[23] Giedrius Paulikas and Dalius Rubliauskas. Flocking of subpopulations in distributed genetic programming. In *ISDA*, pages 320–325, 2005.

[24] C. C. Pettey. Diffusion (cellular) models. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C6.4:1–6. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

[25] W. F. Punch. How effective are multiple populations in genetic programming. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Proceedings of the Third Annual Conference on Genetic Programming*, pages 308–313, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[26] A. Salhi, H. Glaser, and D. De Roure. Parallel implementation of a genetic-programming based tool for symbolic regression. *Information Processing Letters*, 66(6):299–307, June 1998.

[27] Ahmed Sobeih, William Yurcik, and Jennifer C. Hou. Vring: A case for building application-layer multicast rings (rather than trees). In *MASCOTS*, pages 437–446, 2004.

[28] K. Stoffel and L. Spector. High-performance, parallel, stack-based genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Proceedings of the First Annual Conference on Genetic Programming*, pages 224–229, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[29] W. A. Tackett and Aviram Carmi. SGPC (simple genetic programming in C). *C Users Journal*, 12(4):121, April 1994,*ftp://ftp.io.com/pub/genetic-programming*.

[30] K. C. Tan, M. L. Wang, and W. Peng. A p2p genetic algorithm environment for the internet. *Commun. ACM*, 48(4):113–116, 2005.

[31] M. Tomassini. Parallel and distributed evolutionary algorithms: A review. In P. Neittaanmki K. Miettinen, M. Mkel and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, J. Wiley and Sons, Chichester, 1999.

[32] Marco Tomassini, Leonardo Vanneschi, Francisco Fernandez, and German Galeano. Experimental investigation of three distributed genetic programming models. In *Parallel Problem Solving from Nature - PPSN VII*, number 2439 in Lecture Notes in Computer Science, LNCS, pages 641–650, Granada, Spain, 7-11 September 2002. Springer-Verlag.

[33] S. Tongchim and P. Chongstitvatana. Comparison between synchronous and asynchronous implementation of parallel genetic programming. In *Proceedings of the 5th International Symposium on Artificial Life and Robotics (AROB)*, Oita, Japan, January 2000.

[34] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the Association for Compunting Machinery*, 33(8):103–111, 1990.

[35] Darrell Whitley. Cellular genetic algorithms. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 658–658, San Mateo, CA, USA, July 1993. Morgan Kaufmann.