Experimental Study of Multipopulation Parallel Genetic Programming.

Fernández F.[1], Tomassini M. [2], Punch III, W. F., [3], Sánchez J.M. [1],

[1] Departamento de Informática. Escuela Politécnica. Universidad de Extremadura. Cáceres.
{fcofdez, sanperez} @unex.es
[2] Institut d'Informatique. Universite de Lausanne.
mtomassi@iissun4.unil.ch
[3]Computer Science and Engineering Department. Michigan State University.
punch@cse.msu.edu

Abstract. The parallel execution of several populations in Evolutionary Algorithms has usually given good results. Nevertheless, researchers have to date drawn conflicting conclusions when using some of the Parallel Genetic Programming models. One aspect of the conflict is population size, since published GP works do not agree about whether to use large or small populations. This paper presents an experimental study of a number of common GP test problems. Via our experiments, we discovered that an optimal range of values exists. This assists us in our choice of population size and in the selection of an appropriate Parallel Genetic Programming model. Finding efficient parameters helps us to speed up our search for solutions. At the same time, it allows us to locate features that are common to Parallel Genetic Programming and the classic Genetic Programming Technique.

1 Introduction

Many researchers have supported the use of several populations when working with Evolutionary Algorithm (EA) techniques. Different experimental and theoretical studies have reported the efficiency of parallel Genetic Algorithms and have studied the relationship between the classic model and the Island model [1][2]

But in the Genetic Programming (GP) domain, things are less clear. Some researchers talk about the usefulness of working with multiple large populations [3], while others question both the size of populations [4] and the efficiency of multiple populations in GP [5], at least for the few problems that have been intensively studied.

Nevertheless, there is no doubt about the nature of the Parallel Genetic Programming (PGP) algorithm. All authors agree that tuning parallel GP parameters, (e.g. the number of populations, the size of each population, the topological connections between populations) is important for gaining maximal performance.

Our goal is to study the parameters that affect parallel performance and study their interactions in common problems. By so doing we hope to develop a more robust model of how parameters might be set so as to maximize the performance of parallel GP on many different types of problems.

This article is structured as follows. The next section deals with the Island Model for parallel processing and the design of our experiments. Afterwards, we look at the tool we used that facilitated the development of these experiments. Next, we present the problems addressed, followed by a detailed description of results and regularities found in all the problems. Finally, we offer our conclusions, which lead us to the suggestion of the existence of an optimal parameter range of values in both GP and PGP.

2 Methodology

2.1 The Island Model

Many researchers have investigated how to parallelize classic evolutionary computation algorithms. Several ideas have been proposed and employed: parallelization using multiple interacting populations, parallelization of fitness evaluation etc.
In GP, parallelization using multiple interacting populations has been used in a number of interesting experiments [3][6][7]. This model introduces some new parameters into the algorithm and so we decided to use it to compare results.

While multipopulation models can be classified in a number of ways [8], we are most interested in coarse-grain parallelisation. In coarse grained models, also called island parallel models, the population of individuals is divided into several autonomous subpopulations, usually called demes. Demes are allowed to exchange individuals at a certain rate, called the migration rate, according to a usually predefined communication topology. The main reason for using this approach is its ability to avoid premature convergence by injecting new individuals, -these promote diversity- while at the same time exploring different portions of the search space within each deme.

Within each deme, a standard sequential evolutionary algorithm is executed between migration phases. Several migration policies have been described [8]. The most common one replaces the worst k individuals of a deme with the same number of individuals coming from other different populations, usually copies of the best individuals.

The topology of communication is also important. The most common topologies are ring structures, 2-d and 3-d meshes. Hypercubes and random graphs have also been employed. Here we introduce a dynamical topology, described in the following section, in which the exchange pattern changes during the run.

At present, there is no commonly accepted way of deciding the best set of parameters (number of demes, size of each demes, frequency of exchange, size of exchange, quality of exchange, topolgy, etc.). Moreover, changes on the part of an algorithm sometimes modify the behavior of the rest. These new parameters not only affect the “parallel” behavior of the system, but also affect the other aspects of all the algorithms; it is thus difficult to decide how parameters affect results.
Therefore, due to the high number of parameters and their interaction, we cannot simultaneously study all of them, and instead must select those that a priori seem to

be the most important. Thus, in our work, we decided to study how the number of subpopulations, the number of individuals, and the migration rate affect the results. Furthermore, we wanted to see how a dynamically changing topology, which will be explained below, may help to solve several typical problems.

2.2 The Software Tool

We chose to modify a standard GP tool [9] to allow us to work with several demes over an heterogeneous network. Our tool [10] uses the Parallel Virtual Machine [11] communication primitives to connect processes, which in this case are subpopulations.

This tool allows us not only to decide classic parameters like the number of individuals per population or mutations and crossover probabilities, but also to choose the number of populations involved in the experiment, the migration rate, the number of migrating individuals and the communication topology.

The tool works by means of the client/server model, in which each client is a subpopulation and the server is a process that takes charge of the input/output buffers and of establishing the communication topology (Fig 1). Thus the server can either work with a predefined topology or dynamically change it during the run.

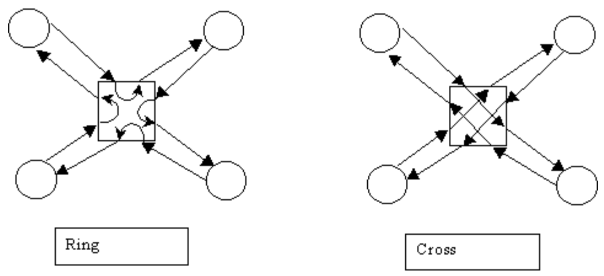

Fig. 1. Client/Server Architecture. The Server decides the topology.

The exchange method we explored most deeply was a random exchange topology. In fact, this means that no set topology was employed. When a population was ready for an exchange, the central server decided randomly from which population the new values would be drawn. This topology proved very effective on the test problems (see section 3.2) Executions and Effort

When using a parallel algorithm, one must carefully judge “improved performance” as these improvements could be due to two factors:

- The time saved by a simultaneous execution of code.
- The possible improvement due to the parallel nature of the algorithm.

While the first factor is useful for shortening the time required to find solutions in GP, which is often a slow process, the second is interesting because it shows us new features of the algorithms compared to the sequential version.

Although the first point certainly helped us to gather a lot of results in an affordable time, we designed the experiments to explore the second factor. We were interested in finding out how different parameters affect results.

Thus, in order to compare results, we decided to analyze data by means of the computational effort, calculated as the number of nodes that are evaluated in a GP tree. Let's suppose that we have executed the same experiment N times; we can compute the average size of individuals per generation in each of the executions (the number of nodes per individual solution tree), and then calculate the average of those N values. What we really have in each generation is the average number of nodes per individual over N execution in a particular problem. Once this average has been computed, the required effort in a particular generation will be: I*N*AVG_LENGTH, where I is the number of individuals in the population, and AVG_LENGTH is the average number of nodes previously calculated.

The computed effort is not necessarily useful for comparing results between very different problems, because different functions, nodes, etc, may require different time values. Nevertheless, it is a helpful measure when comparing different results from the same problem.

Since we are working with several populations instead of just one, it is also easy to compute the effort. In each execution we compute the average number of nodes, taking into account the average number of nodes per individual per generation in each of the populations. Then we can proceed as we would in a classic model, by averaging all the values of the N different executions per generation. Finally we multiply the number of population by the number of individuals and by the average number of nodes.

If we add up a number of generations' efforts, we'll have the computational effort required to obtain a result in a particular generation.

2.4 The Problems Addressed.

GA and GP have proven to be powerful tools when solving problems in many different fields. Results often have improved on others previously obtained by other kinds of machine learning methods.

In our work we are not particularly interested in solving specific problems but in extracting some basic features that could be useful in many problems; thus we decided to use some classic GP test problems. We chose "the even parity 5 problem" and "the symbolic regression problem".

2.4.1 The Even Parity 5 (Evenp 5) problem.

The goal here is to decide the parity of a set of 5 bits. The Boolean *even-k-parity function* of *k* Boolean arguments returns T if an even number of its Boolean arguments are T, and otherwise returns NIL. If *k=5,* 32 different possible combinations are available, so 32 fitness cases must be checked to evaluate the accuracy of a particular program in the population. The fitness can be computed as the number of mistakes over the 32 cases.

Every problem to be solved by means of GP needs a set of functions and terminals. In the case of the Evenp-5 problem, the set of functions we have employed is the following: F={NAND,NOR}, smaller than that described in the original version of the problem [12].

2.4.2 The Symbolic Regression Problem.

The goal here is to find an individual i.e. a program which matches a given equation. For each of the values in the input set, the program must be able to compute the output obtained by means of the equation. We employ the classic polynomial equation:

$$f(x) = x^4 + x^3 + x^2 + x \quad \textbf{(1)}$$

And the input set is composed of the values 1 to 1000.

For this problem, the set of functions is the following: F={*,//,+,-} where // is like / but returns 0 instead of ERROR when the divisor is equal to 0, thus allowing syntactic closure.

3 Results: Optimal Parameter Range of Values

Once the problems were defined, the different parameters of the experiments had to be chosen. We decided to deal only with the number of populations, the migration rate, the number of individuals in each deme.

Different test cases were tried. First we used only one population to solve the problem, with different number of individuals each time. Afterwards we tried to solve the problems with several populations and different numbers of individuals.. Each of the graphs we show is an average over 60 executions, so the results are statistically significant. Note that overall we performed approximately 4000 runs to gather these statistics.

3.1 Even Parity 5.

Figure 2 compares different sizes of populations when solving the Evenp-5 problem. We can see that there is an optimum number of individuals which produces the best results. This seems to agree with other researchers results [1][4]

The best possible result in this problem is the value 16, when a given effort is applied.

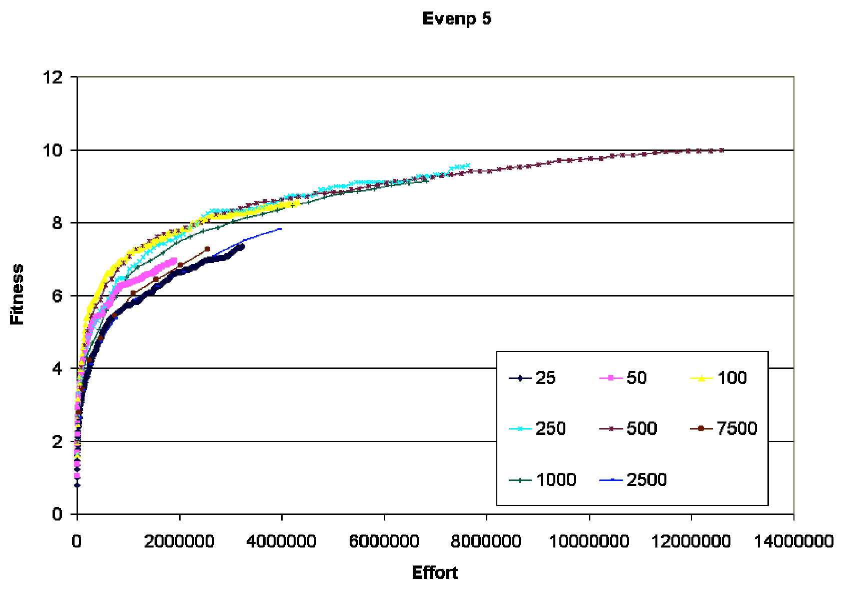

Fig. 2. Evenp-5 problem solved by means of Classic sequential GP.

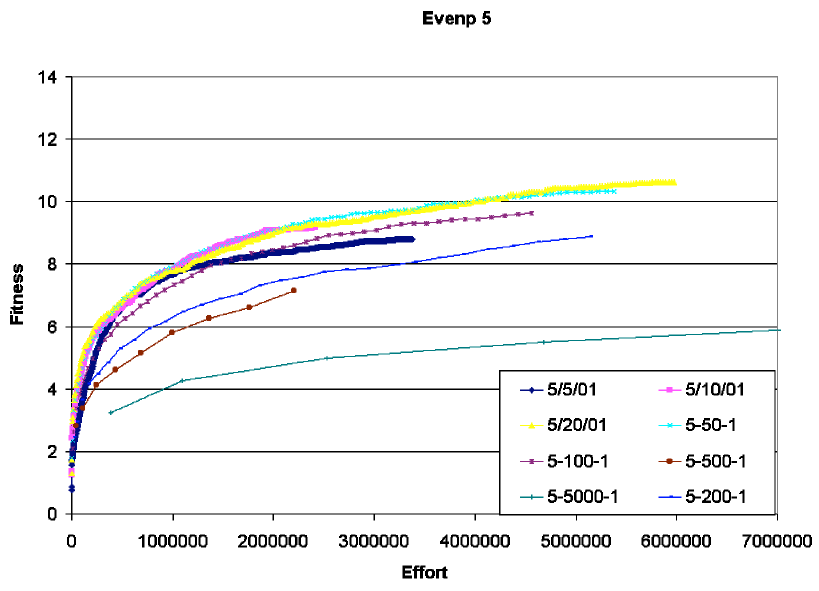

Fig. 3. The Evenp-5 problem solved by means of 5 population with a random communication Topology, where each population sends the best individual after 1 generation.

Figure 3 also compares different population sizes, but we are now working with the island model PGP. Again we find an optimum population size. In these experiments we used 5 populations communicating by means of random topology.

Finally, figure 4 depicts the same results as figure 3 but now using 2 populations. In both experiments, each population sends its best individual after each generation, according to random communication topology.

None of the three graphs finds solutions in the end, because we did not wait for them to be reached. However, some important information can be retrieved from the results.

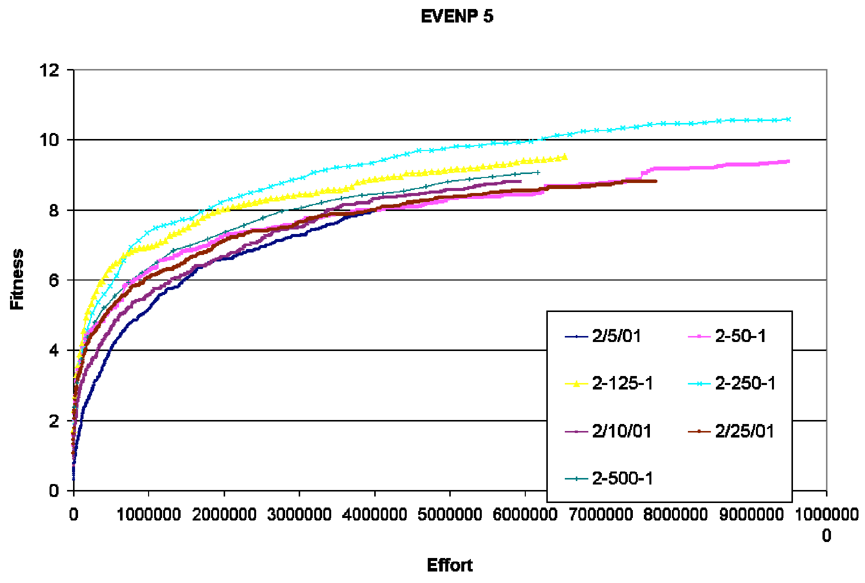

Fig. 4. The Evenp-5 problem solved by means of 2 population with a random communication Topology, in which each population sends the best individual after 1 generation.

We can observe an interesting point: the best choice in the classic problem corresponds with the best choice in PGP case: we just have to multiply the number of populations by the number of individuals. Although we cannot exactly say the optimum number of individuals in each of the cases, we can establish an optimal range limited by two numbers which indicate the best option each time. Let's imagine that the numbers are [A, B]. If we decide to use NP populations, in the PGP model, then this range tells us that the number of individuals per population should be chosen from between [A/NP, B/NP]. If we make NP equal to 1, then we are just talking about a number of individuals. For this problem the range seems to be [200,500].

But one should not generalize when talking about individuals. We think it's more accurate to talk about a pathway delimited by NUMBER_POPULATIONS * NUMBER_INDIVIDUALS.

3.2 Symbolic Regression

We present the results obtained from this problem when conducting the same experiments as with the evenp-5 problem.

In these diagrams, the best fitness value is 0. Therefore the best curves are at the bottom of each diagram.

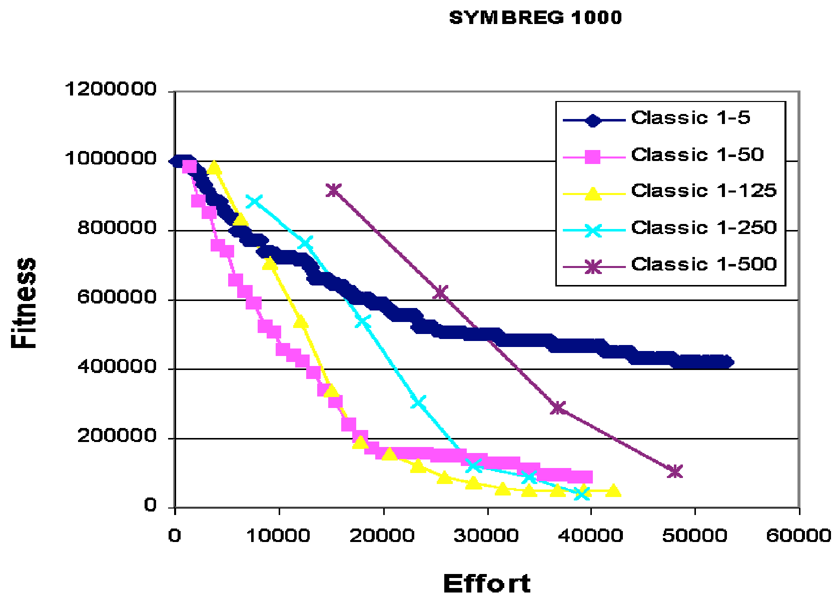

Fig. 5. The Symbolic Regression problem solved by means of Classic GP.

Figure 5 now compares different population sizes when solving the problem with 1000 checkpoints. Again, an optimum number of individuals can be observed, although it is different from that obtained in the evenp-5 problem (Figure 2).

Figure 6 is equivalent to figure 3 but the results are for the problem at hand and using 10 populations instead of 5. Figure 7 is also analogous to figure 4 employing 2 populations.

If we apply the same reasoning as in Evenp-5 problem, we observe that a different range appears again. Now, the limits could be [100,200]. So, when using one population we have a range of individuals [100/1, 200/1]. When using X, we have [100/X,200/X] which agree with the previous results: if we put 1 instead of X.

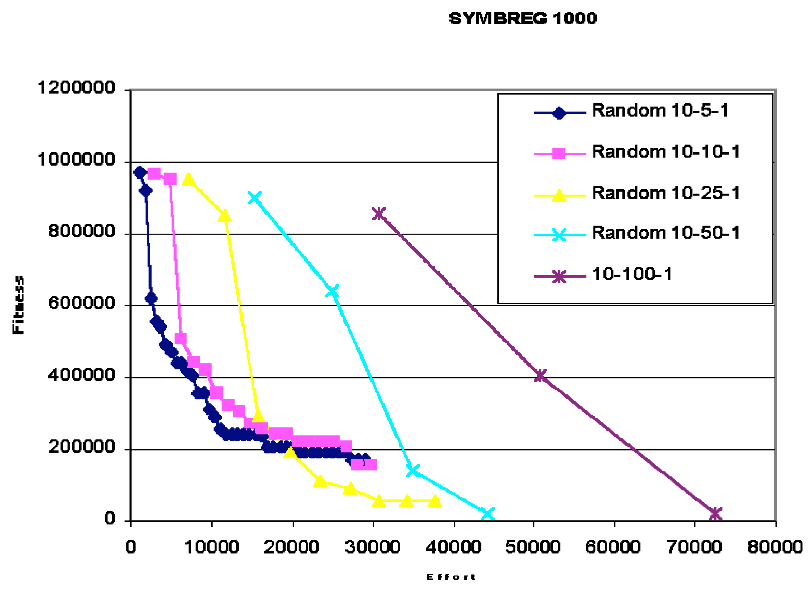

Fig. 6. The Symbolic Regression problem solved by means of 10 population with a random communication topology in which each population sends the best individual after 1 generation.

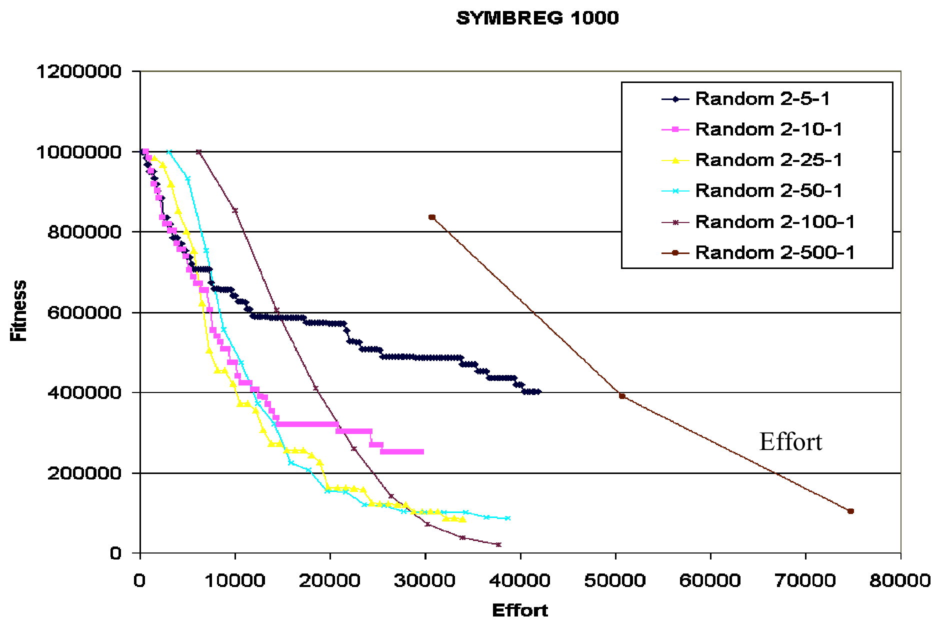

Fig. 7. The symbolic regression problem solved by means of 2 population with a random communication topology, in which each population sends the best individual after 1 generation.

4 Conclusions and Future Work.

We have presented a study of parallel GP as applied to two standard GP problems. It indicates that there is an optimal parameter range of values that allows us to reduce the effort involved in solving these problems. Moreover, the range is different for each problem. Thus, we should try to classify problems by means fo this range. The validity of these results is being checked via investigation into other problems. To date, the results for these new problems have supported our previous conclusions concerning the existence of the pathway. These experiment's results help us to understand the working of GP and PGP and the relationship between several parameters, particularly between the number of populations and the number of individuals. We think that only by means of a series of experiments like that which we presented above will we be able to understand perfectly the dynamics of evolutionary techniques.

On the other hand, we must bear in mind that more experiments, and more kinds of problems should be addressed. We are planning to compare different communication topologies in order to find out which is the best, or at least to classify problems according to the results of the experiments.

Moreover, theoretical models of the multiple populations' dynamics should be built in order to obtain more explanations for the empirically observed phenomena.

In future papers we will present new comparisons between other interesting parameters when using PGP.

5 References

1. Cantú Paz and David Goldberg: "Predicting Speedups of Ideal Bounding Cases of Parallel Genetic Algorithms". Proceedings of the Seventh International Conference on Genetic Algorithms. Morgan Kaufmann.
2. Darrell Whitley, Soraya Rana, and Robert B. Heckendorn. "Island Model Genetic Algorithms and Linearly Sparable Problems". Evolutionary Computing: Proceedings of the AISB Workshop, Lecture notes in computer science, vol. 305 D. Corne and J. L. Shapiro (Eds), Springer-Verlag, Berlin, 109-125, 1997.
3. David Andre and John Koza: "Parallel Genetic Programming: A Scalabel Implementation Using the Transputer Network Architecture". Advances in Genetic Programming 2. The Mit Press.
4. Matthias Fuchs: "Large Populations are not always the best choice in Genetic Programming". Proceedings of the Genetic and Evolutionary Computation Conference GECCO 1999.
5. William F. Punch*:* "How effective are multiple populations in Genetic Programming". Genetic Programming 1998: Proceedings of the Third Annual Conference, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogela, M. Garzon, D. Goldberg, H. Iba and R. L. Riolo (Eds),Morgan

Kaufmann,
San Francisco, CA, 308-313, 1998.

6. S. Arnone, , M. Dell'Orto, A. Tettamanzi, M. Tomassini,: "Highly Parallel Evolutionary Algorithms for Global Optimization, Symbolic Inference and Non-Linear Regression". Proceedings of the 6th Joint EPS-APS International Conference on Physics Computing. 1994.
7. M. Oussaidéne, B. Chopard, O. V. Pictet, , M. Tomassini: "Parallel Genetic Programming and its application to trading model induction". Parallel Computing, 23, pp. 1183-1198, 1997.
8. S-C Lin, W.F. Punch and E.D. Goodman, "Coarse-grain Genetic Algorithms, Categorization and New Approaches" Sixth IEEE Parallel and Distributed Processing Oct 94, pg. 28-37.
9. Weinbrenner, T.;"Genetic Programming Kernel Version 0.5.2 C++ Class Library". http://thor.emk.e-technik.th-darmstadt.de/~thomasw/gpkernel1.html
10. Francisco Fernández, Juan M. Sánchez, Marco Tomassini and Juan A. Gómez: " A parallel Genetic Programming Tool Based on PVM". Jack Dongarra, Emilio Luque, Tomás Margalef (Eds) Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer.
11. V.S. Sunderman "PVM: A Framework for Parallel Distributed Computing", Journal of Concurrency: Practice and Experience" pp. 315-339, Dec 1990.
12. J. Koza "Genetic Programming II". The MIT Press.