ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2005, Vol.34, No.4

MOVEMENT OF FLOCKED SUBPOPULATIONS IN DISTRIBUTED GENETIC PROGRAMMING

Giedrius Paulikas, Dalius Rubliauskas

Department of Multimedia Engineering, Kaunas University of Technology
Studentų St. 50, LT–51368 Kaunas, Lithuania

Abstract. The rules of swarming intelligence can be applied to govern migration in the distributed genetic programming (DGP) algorithm, but they require modifications. Initial rules are taken from the flocking algorithm and adapted for DGP. As the rule for the alignment of direction is completely discarded and the remaining rules operate on implicit data of subpopulation locations, the resulting joint search technique must be reevaluated. This article presents the pragmatic coupling of flocking and DGP algorithms. The experiment of visualizing the movement of DGP subpopulations through the search space provides a graphic overview of the behavior of DGP subpopulations. The results confirm a typical influence of the modified flocking rules to the flockmates represented by the subpopulations.

1. Indroduction

Genetic programming (GP) is an evolutionary algorithm for automatic creation of computer programs. It allows getting the program to solve the given problem without specifying the algorithm itself. The roots of this automatic programming method lies in Darwinian natural selection and mimics the hypothesis of the evolution of living beings by the survival of the fittest.

Historically the first thoughts of using evolution for artificial intelligence date back to year 1948, when Turing [1] described the genetic, or evolutionary, search as one of the approaches to the machine intelligence. The foundation for GP were laid in 1975 by the research of cellular automates by Holland, that led to the formation of genetic algorithms (GA). Genetic programming emerged as the specification of GA that was suited to create the computer programs. While the first GP the papers of Smith report research in 1980 and Cramer in 1985 [2], J.R. Koza is considered as the inventor of genetic programming due to his extensive work in this field in the nineties. GA and GP falls to the same group of evolutionary algorithms (EA), that are based on applying selection and reproduction to the set of individuals according to their performance. The evolutionary algorithms also contain such related artificial intelligence techniques as evolutionary programming, evolution strategies and classifier systems.

As suggested by A. H. Turing, search and optimization is the core of artificial intelligence. The task of finding the solution of the problem to the computer means literally finding the best solution of all possible solutions. There are numerous search methods and while are more general than others, there is no universal method for all occasions ("No Free Lunch" theorem [3]). An interesting example of the search technique that theoretically is of good in all domains is Levin search [4]. If some search algorithm finds the solution in O(f(n)) steps, it is proved that Levin search can solve the problem in at most O(f(n)) steps. The practical application of Levin search is hindered by the constant factor in O(f(n)) notation which becomes crucial for real life problems with enormous search spaces.

The last years of GP research have brought some theoretical insight to the principals of the operation of genetic programming algorithm. The schemata theory existed since the early days of GA research and is as well applicable to GP. The substantial progress was made by exploring Markov chain models to describe processes of GP algorithm [5]. The GP theory research has revealed that GP is a more general search technique than GA, GA being the simpler case of genetic programming.

2. Distributed GP

The algorithms of evolutionary search depend on having the big population of possible solutions that covers as much of the problem search space as possible. The search space is defined at the start of search and in order to get the solution faster should be defined as small a possible. This is the task for the search algorithm programmer and selecting the right representation of the solution does it. In case of GP, where solutions are usually represented as program parse trees, the most important to successful search are the selection of tree nodes and the initial/maximal tree sizes. The first one denotes the allowed terminal and functional items of the parse tree and define the abstraction level of the search. The second slices some finite region of the infinite search space that is expected to have the sought solution. The adaptive representation (e.g., Automatically Defined Functions) gives the GP algorithm the ability to manipulate the levels of abstraction during the execution. Such a technique helps to narrow the search space according to the results of the search and usually speed up the search [6].

Another way to accelerate search is the parallelization of GP algorithm. The biggest portion of computations is conducted independently for each member of the population of solutions, so GP algorithm can be easily adapted to parallel execution. A big number of potential solutions provide the opportunity to select the required level of parallelized nodes: from fine-grained, where each solution can be assigned to separate node, to several large subpopulations. The empirical results show that coarse-grained division of the population to big almost independent subpopulations is the most beneficial: it minimizes the communication among computational nodes and matches the most popular architecture of (affordable) parallel systems. Moreover, such distribution finds a solution faster even if executed on a single non-parallel machine. This coarse-grained paralellization is called the "island model" (by similarity to the evolution of individuals of the separated islands), or simply distributed genetic programming (DGP). The only thing that is added to standard GP is the migration of solutions among subpopulations. The migration adds new parameters for the customization of the algorithm. As with any other configuration, changing the values of the migration parameters can lead either to improvement of deterioration of the search characteristics. Besides the number of subpopulations, the main additional parameters of DGP are:

1. migration frequency - how many generations are run between two subsequent migrations;
2. migration rate - how many individuals migrate each time;
3. migration topology - which subpopulations participate in the exchange.

There are some issues that must as well be considered in the DGP implementation, though their configuration is not as important as the three parameters of migration. The first one is selection of the individuals for migration. Basically, it may be left the same as the selection for the genetic operators (crossover, mutation and replication) since each subpopulation should use the information about the exploration of its search space domain and send good genetic samples to neighboring subpopulations. The other matter is the exchange type: copy, move or even some form of crossover of the emigrants. Either one of them is suitable as far as genetic material reaches the intended target.

This article focuses on setting of the migration parameters of the DGP algorithm. The goal is to set migration in the way that the performance of the DGP algorithm is at least better than the one of the standard GP (best case scenario is achieving the results comparable with the human optimized migration parameters) while keeping the need to set parameters by hand at the minimal level. In other words, the parameters should be set dynamically during run time.

3. Swarm intelligence

Swarm intelligence (SI) is another search and optimization technique that comes from the field of biology. SI system consists of the number of independent agents that interact with each other. There is no central control for the behavior of the agents. The emerging behavior of the whole system typically is predictable only in the short term. Each agent has the common goal (e.g. finding food in natural environment, or optimizing some values in the search task) as well as the responsibility to observe the conduct of neighboring agents. If the agents dispose the information about global status of the whole group (e.g. the optimal spot in the search space so far) depends on the implementation of SI. The best known SI methods are Particle Swarm Optimization (PSO) [7] and Ant Colony Optimization (ACO) [8].

One can notice some similarity between SI and DGP: both have a number of interacting entities that explore a problem search space. For DGP this entity, or agent, is a subpopulation that covers a portion of common search space and communicates with neighboring subpopulations. Application of the SI rules to the DGP algorithm gives the opportunity to influence the movement of the subpopulations through the search space. The rules are adapted from one of the earliest successful implementations of the SI - flocking. Flocking can be described as a case of PSO without the knowledge of the global results (that is, each agent communicates only with its neighbors). The flock members follow three main rules [9]:

1. Separation – avoiding crowding/collisions with neighbors;
2. Alignment – moving to the same direction as most of the neighbors;
3. Cohesion – moving to the average position of neighbors.

These rules can be summarized as the desire of each agent to keep a constant formation, or grid, of the whole flock. Since rule 1 and rules 2+3 drive the agent to the opposite directions, it stays in more or less stable position to its neighbors while moving through the search space.

4. Adapting flocking rules to DGP

Figure 1 shows an overview of the DGP algorithm that includes flocking rules. The important part is the dynamic migration that adds calculations of location and neighbors, while standard DGP only executes the described exchange of individuals.

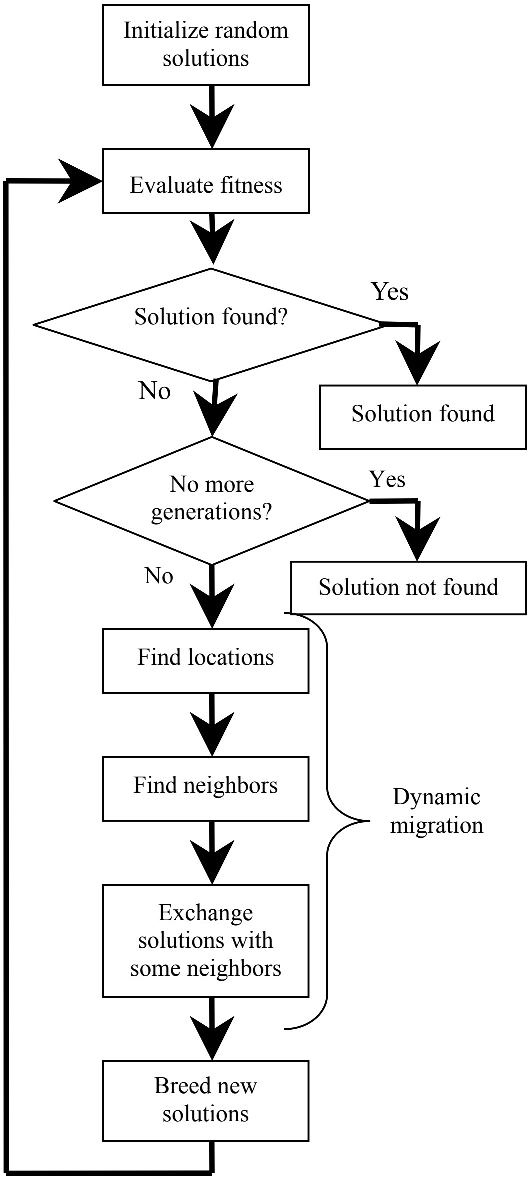

Figure 1. DGP algorithm with dynamic

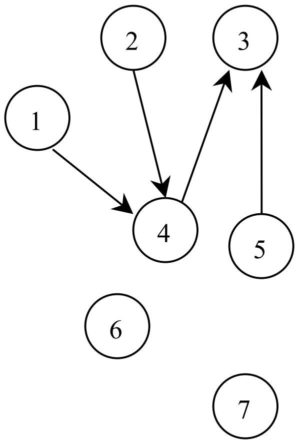

Figure 2. Example of the required migration

The main challenges of using DGP and flocking together arise from the different notions of the search space and movement of the search agents through it. These differences are summarized in Table 1.

As the domain (search space) and the way of conducting the search (movement) differ, it is impossible to apply the flocking rules directly. The arising difficulties are given below:

- **Search space**. While working with locations and distances in the Euclidean space is trivial, comparison of two trees that represent the solutions in DGP is not. The matter is even more complicated because of heterogeneity of tree nodes (various functions and terminals) and different arity of nodes (from 0 to infinity). Direct mapping from the tree to linear dimension is impossible (or requires intensive use of computational resources, what would annul the benefit of applying flocking to the DGP).
- **Movement**. The current location and velocity of the flockmate in Euclidean space fully describes the subsequent location, so altering the velocity is enough to drive the agent in the desired direction. But the changes made to the parse trees by the selection and genetic operators have the probabilistic nature and can't be predicted. So there's no way to anticipate the next form of the parse tree of a group of them (subpopulation).

Table 1. Main differences between DGP and flocking

	Search space (agent position)	**Movement**
DGP	Hierarchical (program parse tree)	Probabilistic selection and genetic operations
Flocking	Euclidean (location in multidimensional space)	Current location and velocity

These inadequacies of search methods require substantial modifications in order to use them together. As we seek just to control the migration of the DGP algorithm, the modified part must be the flocking rules, not the GP algorithm. The changes are as follows:

- **Location in the search space**. Though the real search space of DGP is a heap of program parse trees, they are too complicated to be used as locations of the flockmates. The remaining choice for the location is a phenotype, i.e. the fitness of the solution. While the single fitness value that is usually employed in the selection procedures of the DGP has very little data about the solution, the results of all fitness evaluations for the particular parse program can be used. The array of the evaluations of each fitness case supplies just the right format of data for the flocking algorithm. The following presumptions justify the use of fitness evaluations as the reflection of the positions in the search space:

Typically there are at least several test cases. This gives the greater variety of possible location values (as opposed to the single value of final fitness).

Different test cases cover some specific collections of input values. So, if the programs showed good results with those particular test data, they probably share some common genetic material.

- **Movement through the search space**. There isn't a notion of **direction** in the DGP search space. Even if the history of the parse tree was collected, it does not affect the genetic operators and the modifications they make to the tree. So the rule alignment to the neighbors must be discarded. As of the location – the way to "move" the parse tree must be done by changing its genetic information. In case of using test case evaluations as the location measures, moving means altering the genotype to have more genes that lead to the required results. This means that it is impossible to move a program to the arbitrary location, we can only move toward the program with the desired location. Such a procedure would cause problems to move a single program parse tree, as it would intersect with the operation of the genetic operators (crossover and mutation). But the flockmates are not single individuals, but subpopulations of the DGP, so the exchange of the genetic data means just passing some individuals among the subpopulations – the migration. The example of this migration is given in Figure 2, where subpopulation 4 tries to move closer to subpopulations 1 and 2, while subpopulation 3 – to 4 and 5. These actions are expected to assist in avoiding the collision with close subpopulations (6 and 5 for 4, 2 for 3).

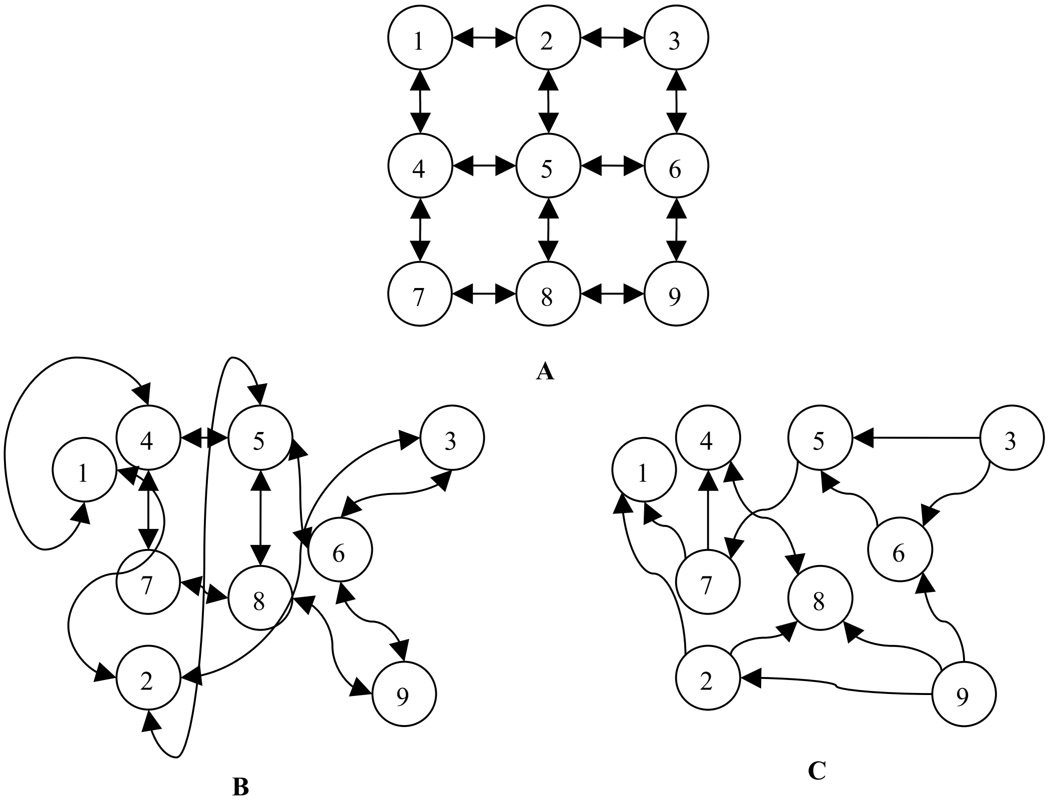

Figure 3. Migration schemes: A – conceptual grid, B – connections in static migration topology, C – connections in flocked migration topology

5. Movement visualization

In order to gain a better understanding of the influence of the flocking rules to the DGP, a simple experiment was carried out. The main goal was to track the movement of the DGP subpopulations through the search space and compare the track of the DGP algorithms with and without flocking.

The GP algorithm was used to solve the trivial problem of symbolic regression: while having only inputs and outputs, the function that yields the given results must be deduced. The function is $x^4+x^3+x^2+x$, test cases consist of 200 random values from range [-1;1). Half of each subpopulation is replaced at every generation. Most other parameters (population size, tree sizes, e.t.c.) were not important for this experiment and were kept at low level to decrease the number of required calculations.

The population was divided to 9 subpopulations. This number is close to the reported optimal values in various DGP research papers [10]. This is also convenient for our task, because the subpopulations are the entities that will be tracked and presented for visual evaluation – too many subpopulations would be hard to comprehend, while just a few wouldn't disclose the dispersion of the flocking subpopulations. The static migration scheme has the grid structure where each subpopulation interacts with adjacent subpopulations. There are 24 migration channels (12, if counting only unique pairs of communicating subpopulations, but information is exchanged both ways, so actually it is 12x2=24) and approximately 3 neighbors (24/9=2.7). 10% of the subpopulation migrates every second generation.

When flocking rules are employed, the neighbors are selected by the locations of the subpopulations and migration is conducted from the most distant neighbors (that should move the current subpopulation closer to the source of the immigrants). The location is measured as the array of the fitness case evaluations. Normally, the 200 fitness cases wouldn't be a problem, but this is far too much for the visualization. The most suited for representation for human comprehension is 2-dimensional space. Therefore the results of fitness cases are split to 2 roughly equivalent groups: one with input $x<0$, the other with $x>0$. Neighbors are selected dynamically (3 neighbors for each subpopulation), and decisions about the directions of migration are carried out according to the location of the neighbors (so there is a possibility to skip migration). The conditions of migration are checked every 2 generations and 10% of individuals are transferred if migration is required (parameters are similar to static migration). The imaginary migration topologies of static and flocked migration control schemes are shown in Figure 3. Flocked subpopulations try to communicate with adjacent subpopulations, while statically linked subpopulations do not take the current arrangements of subpopulations in the search space.

The track is made up of ten subsequent generations. The full pictures of both migration schemes are shown below (Figure 4). The subpopulations are encoded by the shape of figures: five subpopulations are circles with different radius, four – squares with different width. The generations are marked by the color of figures: the darker the color, the bigger the generation number (10th generation if black). Quick observation of these images shows the preference of flocked DGP to keep the search more dispersed. The dark figures of the static migration scheme gather to the upper left corner, while flocking rules support the exploration of more search space (presumably the effect of separation rule).

Figure 4. The combined traces of static (A) and flocking (B) migration schemes

The next sets of images show the snapshots of the arrangement of subpopulation at certain generations. They are presented to give the better overview of the locations of the members of the group of subpopulations at discrete time moments. The subpopulations are encoded as in previous figure (circles and squares with various sizes). The snapshots are taken every second generation.

Comparing the images in Figure 5, we can notice the same effect of greater dispersion of flocked subpopulations (subpopulations in the highest image in the right column are crowded, but this image is at the start of the run, so it's mostly random genetic data). The crowding in statically controlled migration scheme becomes more obvious as GP algorithm run reaches the later generations.

6. Conclusions

The article looks at the blending of distributed genetic programming with the modified version of swarm intelligence (flocking). The modifications are needed to unify the approach of these search techniques to the search space. Unmodified flocking rules have no ability to cope with the complex program parse trees that are governed by the probabilistic selection and genetic operators. In short, the proposed way is to use the results of fitness evaluations as the reflection of the real search space (parse trees) and discard the precise directions of movement by just moving flockmates toward each other.

Since the adoption of flocking requires a considerable amount of simplifications, it is desirable to check the flocking rules still work. The purpose of the flock (apart of the goal of finding food or some other search) is to keep the flexible but constant grid of the flock: no lost flockmates and no collisions. The presented tracks of DGP subpopulations display the desired characteristic of avoiding the crowded search. This should prevent some cases of premature convergence to the suboptimal solution.

The results of the experiments that were used for the visualization of search didn't reveal any substantial differences of the effectiveness of static and flocking schemes of migration control. Further research is required to confirm that flocking scheme will have the higher success rate for problems where futile search is common.

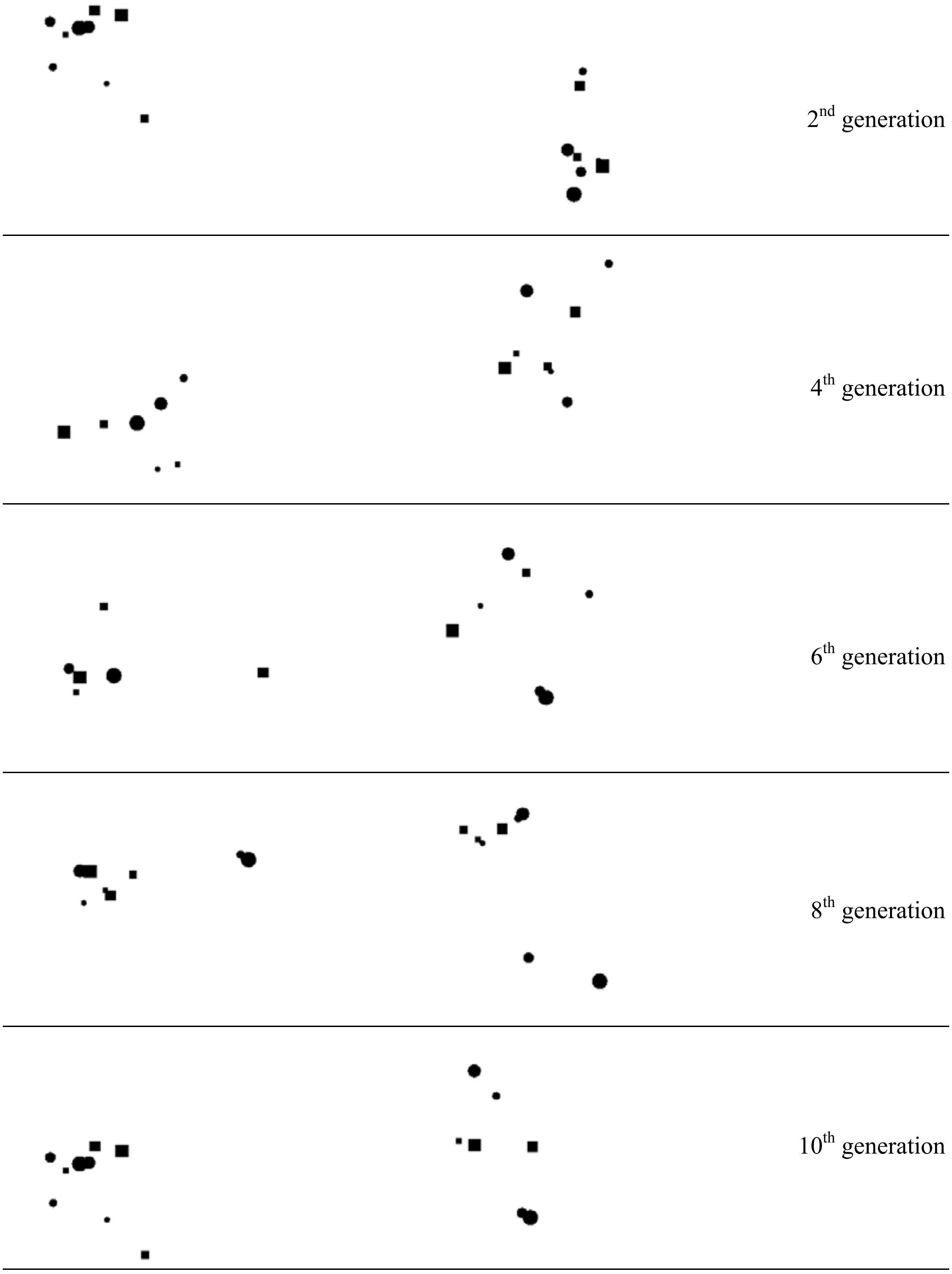

Figure 5. Generation snapshots of static (A) and flocking (B) migration schemes. The upper images are the oldest generations

References

[1] **J. R.Koza.** Genetic Algorithms and Genetic Programming. *http://www.genetic-programming.com/c2003lecture1modified.ppt*, 2003.

[2] **J. Schmidhuber.** Program Evolution / Genetic Programming. *http://www.idsia.ch/~juergen/gp.html*, 2005.

[3] GA or GP? That is not the question. *Proceedings of UK Workshop on Computational Intelligence*, 2003, 8.

[4] **J. Schmidhuber.** Levin search. *http://www.idsia.ch/~juergen/mljssalevin/node4.html*, 2003.

[5] **R. Poli, J. E. Rowe, N. F. McPhee.** Markov Chain Models for GP and Variable-length GAs with Homologous Crossover. *Proceedings of The Genetic and Evolutionary Computation Conference*, 2001, 8.

[6] **J.P. Rosca, D. H. Ballard.** Genetic Programming with Adaptive Representations. *The University of Rochester, New York*, 1994, 30.

[7] **J. Madar, J. Abonyi, F. Szeifert.** Interactive Particle Swarm Optimization. *Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, 2005, 314-319.

[8] **M. Dorigo, L.M. Gambardella.** Ant colonies for the traveling salesman problem. *BioSystems*, 1997, 10.

[9] **C.W. Reynolds.** Flocks, Herds, and Schools: a Distributed Behavioral Model. *Proceedings of Conference ACM SIGGRAPH* '87, *Anaheim, California*, 1987.

[10] **F. Fernandez, M. Tomassini, W.F. Punch III, J.M. Sanchez.** Experimental Study of Multipopulation Parallel Genetic Programming. *Lecture Notes in Computer Science, Vol.*1802, 2000, 283-293.