ISSN 1392 – 124X INFORMACINĖS TECHNOLOGIJOS IR VALDYMAS, 2004, Nr. 2(31)

PARSE TREE POSITION MEASURING IN DISTRIBUTED GENETIC PROGRAMMING

Dalius Rubliauskas, Giedrius Paulikas, Bronislovas Kilda

Kaunas University of Technology, Department of Practical Informatics
Studentų 50, Kaunas

Abstract. Distributed genetic programming (GP) is a step forward in optimization of the GP algorithm, but it suffers from the difficulties of setting the proper distribution parameters. One of the most important parameter – classes, responsible for the migration among subpopulations, can be put under the control of flocking. The challenge in applying flocking to distributed GP lies in measuring the positions and distances between the program parse trees. This paper discusses the details of possible methods for measuring the tree position, paying the most attention to resulting distance values that are of the primary goal for a successful combination of distributed GP and flocking.

1. Indroduction

Genetic programming (GP) is an evolutionary search strategy. It is derived from the *genetic algorithms* (GA), adopting them to search for computer programs. GP is a versatile and powerful technique to solve tasks with known domain and expected results, but when the way to achieve these results is unknown. It's an algorithm for creating algorithms and is one of the methods used for automated programming.

Being an evolutionary strategy (all they are based on the Darwinian natural selection), GP suffers from the massive amount of the required computations, and, consequently, long run time. Since we prefer to get the solution to a problem as fast as possible (this is especially important to real life problems), GP algorithm can be accelerated by employing the technique, widely used in nature to speed up brain activities – the distribution of computations. Genetic programming, as well as all genetic algorithms, is well suited for parallelization: each individual (computer program in GP case) can be processed separately most of the time.

An observation of natural processes gives one more improvement of the GP algorithm. It was noticed that the algorithm performs even better when the individuals are distributed not globally, but to partially isolated chunks. Each part runs the GP algorithm almost independently, just with some information migration at fixed time intervals. Due to similarity to wild life evolution in isolated islands, this genetic algorithm parallelization method is called *island model* or *distributed GP*.

The problem with distributed genetic programming lies in the complicated selection of distribution parameters. These parameters denote how individuals must be divided into subpopulations and what level of information migration is needed among subpopulations. The parameters of the distributed GP add another level of required optimizations on parameters of the sequential genetic algorithm (e.g. genetic operator probabilities, population size, selection strategy) and the best values may vary from task to task. They usually are left for human operator competence, but since the whole run of the GP algorithm is needed to justify if parameters are well chosen (which is a lengthy process), some automated technique would be desirable.

Here comes another nature inspired algorithm, which can be applied to facilitate the selection of distribution parameters. It's called *flocking* and is used to simulate life-like behavior, observed in movement of animal parties (e.g. flocks of birds or schools of fish). The flock motion is controlled by the 3 main rules [3]

1. *Separation* - avoiding collisions with flock-mates.
2. *Alignment* - steering to the average direction of flock neighbors.
3. *Cohesion* - moving to the average position of flock neighbors.

Some researchers report a successful application of flocking rules as the search algorithm (particle swarm optimization) [4]. This search is similar to the exploration of the unknown feeding territory done by the party of animals. Each animal covers only a small area, but when it spots some food, it moves in that direction and eventually the whole flock gets affected by that behavior and tends to search the areas around the place where the food was found. As a result the flock moves in a hardly predictable manner, even

though the ultimate goal is to find food, and explores vast areas, giving the priority to the more promising regions.

All earlier researchers of the distributed GP and flocking analyzed these algorithms as separate entities, without attempts to combine them together. This paper suggests to join both search strategies, making the flocking a supplement for the distributed GP. As we will see further in the article, the fundamental task for this combination of the two algorithms is the assessment of the position of the program parse tree in the search space.

If we assume the whole population of individuals in the distributed GP as the flock of subpopulations, flocking rules can by adapted to control subpopulations. The distribution parameters can be separated into two main sections:

1. *Division parameters* direct how many and what size subpopulations are created.
2. *Migration parameters* govern the information exchange among subpopulations: exchange frequency, rate, selection of migrating individuals, etc.

As for the first parameter type, in the static distribution where subpopulations count and their sizes are set at the beginning of the genetic algorithm run, the flocking won't be much of use. To control these parameters, at least subpopulations of a volatile size are required. But since variable dynamic subpopulations aren't proved to be useful [2], such experiments are better left to be carried out in case flocking performs successfully in controlling other distribution parameters.

The second type of parameters is obviously fit to be adjusted by flocking rules, because they reflect the mutability of subpopulations. Internally, this mutability is controlled by GP rules that can be thought as the animal motivation to find food. But this controls only how the subpopulation moves through the search space by itself, without external impact from other subpopulations. When subpopulations communicate with each other by the means of migration, their coverage of the search space (or the location in the search space) is changed from external sources. Migration parameters control how this external effect is applied among neighboring subpopulations. By changing these parameters we can drive the subpopulation in the desired direction through the search space. That's where the flocking theory is used – to manipulate the influence subpopulations of the distributed GP make to each other.

Flocking usage grants one more merit for distribution control – migration topology must not be specified beforehand. Subpopulations, as well as their positions, change under the influence of genetic operations and migration input. Neighboring subpopulations are the ones with the smallest distance between them, so before the migration takes place, each subpopulation must find a required number of neighbors according to the current situation.

In the second part of this article we explore elements of five different ways (two indirect and three direct) to measure the position of the parse tree, analyzing the speculative merits and drawbacks of each of them. The last part contains the results of the experiment, which compares the distance measures derived from the application of the earlier discussed methods. The results show that indirect methods of position evaluation render more diverse distance measures, so, they should be used when smaller migration rates are required. The direct evaluation strategies give similar results and should be considered as interchangeable.

2. Position measuring strategies

As it is discussed in [1], the individual location in a hypothetical problem search space corresponds to positions required to be able to apply flocking rules. Here we talk about an individual, though what we really need is the subpopulation position. But since the subpopulation is nothing more than a collection of individuals, it seems natural first to measure positions of each individual and then calculate the average of the whole subpopulation. Still, the measurable positions must be calculated for individuals of the subpopulation, and that's what composes the difficulty of using flocking for the distributed GP. The real position of an individual, represented by the program parse tree, is the tree itself, which is rather complicated as the measure to be used in calculations. So we have two options: either to try using the raw (parse tree) individual position, or convert this position to some more manageable form. Conversion here basically means getting rid of the parse tree hierarchy. Actually, the hierarchical structure of the position isn't much of an obstacle for calculations (if not speaking about the amount for needed computations), but it's not clear how the hierarchy of the tree reflects the importance of nodes in designating place in the search space. So the linear position measure is desirable.

Here we'll discuss details of different possibilities for assessing GP individual position, starting from the simplest and finishing with more complicated and computation intensive.

The first two are *indirect* strategies; they are based on individual phenotype evaluation.

1. Fitness. That's the most straightforward strategy to measure the individual position; it doesn't require any additional computations besides the fitness evaluation, which is one of the most important aspects of the GP algorithm. Fitness maps each individual to a value (usually the floating point number) in a single dimension, so the position can be easily processed by flocking rules. The important drawback of using fitness as an individual position measure in the fact that the fitness value doesn't carry information about the individual itself, but rather about how successful that

program solves the given task. So two individuals with equal fitness values (and consequently equal positions) can have nothing in common when we look at their parse trees. Though, even if fitness isn't the right measure in strict sense, it's the value that governs the propagation of individuals to higher generations and that can be enough to organize the movement of the individual among subpopulations. There are several common fitness evaluations (raw, standardized, adjusted, normalized) and any of these can be used to evaluate the position of the individual. The one used for the parent selection in genetic operators probably should be preferred, since it emphasizes the differences among individuals that are best for the problem being solved.

2. Fitness cases. That's another indirect position measuring strategy, based on the performance of the individual when solving the given problem. It is applicable only when there is more than one fitness case, since in case of a single data set we immediately get the fitness value after the evaluation of the parse tree (see the "fitness" strategy above). When several fitness cases are present, we get the potential to record the results of the individual with each fitness case. This strategy can be viewed as special, more precise case of the "fitness" strategy, because it as well doesn't measure the genotype of the individual. So some information about the individual is lost when the transition from the parse tree to the performance assessment is carried out and different individuals can get similar results only because they solved the same fitness cases with comparable results. But, again, if the individuals performed similarly for each fitness case, then there are big chances that their parse trees carry lots of analogous genetic information.

All the following evaluation strategies are *direct,* based on the genotype of the individual.

3. Node items. This is the simplest attempt to break the entire hierarchical structure of the tree to simpler elements for easier calculations. Since the tree is a collection of nodes with links among them, we can try to neglect the links and count only how many different nodes the tree has. That way we abandon one important aspect of the location of the individual in the search space, since node functions depend on the place in the parse tree (we can't say that the functions of the node in the tree root or the leaf are the same), but achieve a simple multidimensional representation of the position of the individual. Even if it is clear that such a strategy captures only a small quantity of the full complexity of the position of the tree, further experiments may show that it's exactly the part that is required to efficiently migrate individuals among subpopulations.

4. Tree structure. The bare hierarchical structure of the tree captures even less genotype information than the "node items" strategy. Imagine an initial population where trees are generated using the full method and with the maximal tree depth, and all functional node items have the same arity (e.g. 2, which is common for arithmetic operations). Then we get a number of equal binary trees, so at the beginning all individuals have the same position evaluation. And that's not going to change further in the run of the algorithm, or will change only to some small extent, because the trees were generated with the maximal depth. Only mutation can change the structure of the tree (crossover just swaps some randomly selected branches), but the mutation probability is normally very small. So we end up with the population of individuals with constant equal positions. Even though this is a bit extreme example (often the "half" tree generation method is used and the initial depth of trees is less than maximal), it shows that this position evaluation strategy must be used with care.

Hierarchical links must be translated to some linear structure that could be interpreted as an array of position coordinates. The most straightforward way to do this is to traverse the parse tree from top to bottom and at each tree level write down the arities of each node. The bottom-up approach is possible, too, but it's more complicated and computation intensive, so we'll stick to the top-down way. The algorithm for the "tree structure" position evaluation would look like this:

```
level := root
position := []
i := 0
repeat
    for each node in level
        position[i] := node arity
        i := i + 1
until tree has deeper levels
```

This measuring strategy tends to generate large position coordinates, because the count of tree nodes grows exponentially with every new level and the linear position value is just a plain representation of the whole parse tree. This rises computation requirements for the position evaluation. The consideration of only the bare structure also looses a lot of genetic information of the individual. Thus, theoretically this strategy seems to be weaker than earlier mentioned competitors.

5. Exact strategy. The last choice is to use the individual parse tree without any mappings or transformations. This way we get the precise individual position and no further calculations for transformation are required. The difficult part is in obtaining the average position of the subpopulation and in calculating the distance between two individuals. Average position calculations weren't an obstacle in earlier strategies, there the position was always transformed to a linear representation. Here, to figure the average subpopulation position, we need to form a single parse tree with the structural and node item information of all individuals. As functional tree node items can have

different arities, the suggested technique is to form a tree of the maximal allowed depth with each node having the maximal arity of all functional items, used in this problem. The node of this newly created tree records what items subpopulation individuals have in corresponding nodes (each node has a hash map with item counts). When individual parse trees are applied to this template, their branches are aligned by the left side.

The second problem of "exact strategy" is the distance assessment. But, even if the position measures are hierarchical (tree templates filled with the information from each individual of the subpopulation), they all have the same structure and that fact is beneficial for further calculations. We don't need to flatten the acquired measure in order to get the distance, the hash maps can be compared node by node.

Distance calculations, applied to complex structured position measures, on the contrary to the linear ones, have some aspects that should be considered. The distance between two measures of linear coordinate arrays with members of an equal significance can be calculated as Euclidean distance:

$d(x,y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$, where x and y are positions with n coordinates.

But when the nodes have parent-child relations, these bonds can (or even must) be taken into account. This applies to "tree structure" and "exact" position evaluation strategies. Parent-child relations can be rated by introducing the weight to tree levels. A suggested scheme would be to divide the input of each level by some coefficient, e.g. 2. So, the root level will supply the full distance value to the Euclidean distance value, the first level – only half of the value, second one fourth and so on. If the tree is flattened as in "tree structure" case, this level weight correction should be applied during the flattening phase.

3. Distance measures

In order to obtain a better understanding of distance measures, generated by different position assessment strategies, there was developed the traditional GP system with all earlier mentioned position evaluation strategies [5]. During the experiment, each program parse tree was processed by all strategies. Nine trees were generated randomly for the symbolic regression of the mathematical expression $a^2 + b - c$. The parse trees were constructed of 3 variable nodes (a, b and c) and 4 functions with arity of two (+, -, *, /), the maximal allowed depth of the parse tree was set to 5 levels. The fitness of the individual was evaluated by 10 fitness cases, filling variables with values randomly drawn from the range [0, 1). Then the distance from the first tree to the other eight trees was calculated, thus getting the matrix with seven strategies for the position evaluation and eight distances for each strategy. The parse trees and resulting Euclidean distances are given in Table 1 and Figure 1.

Table 1. Randomly generated trees

No.	Pre-order tree representation
1	`(* (+ (* c a) a) (* b c))`
2	`B`
3	`(/ (- (- (- b b) (* b a)) (+ b a)) (+ c a))`
4	`(/ (+ c (+ (* c a) (- c a))) c)`
5	`(/ b a)`
6	`(+ a (- c (* b (/ c c))))`
7	`A`
8	`(/ b c)`
9	`(- a (* a a))`

4. Conclusions

Figure 1 indicates differences between indirect and direct measuring strategies: indirect methods tend to represent individuals as more disseminated in the search space, while direct methods report less distance diversity. Most direct strategies (except the weighted tree structure) have similar patterns of the distance distribution. Both weighted direct measures give distinguished results, probably due to an inadequate selection of weights for tree levels. So, what position evaluation strategy is selected may depend on the demands of the task: indirect measures should be chosen when more distance diversity is required. For flocking more diversity means less neighbors and, consequently, smaller migration rates. Until further experiments regarding the position measuring impact on the results of the distributed GP will clarify the advantages and disadvantages of each strategy, the choice is up to the human algorithm operator, who must take into account the distribution requirements of the problem being solved.

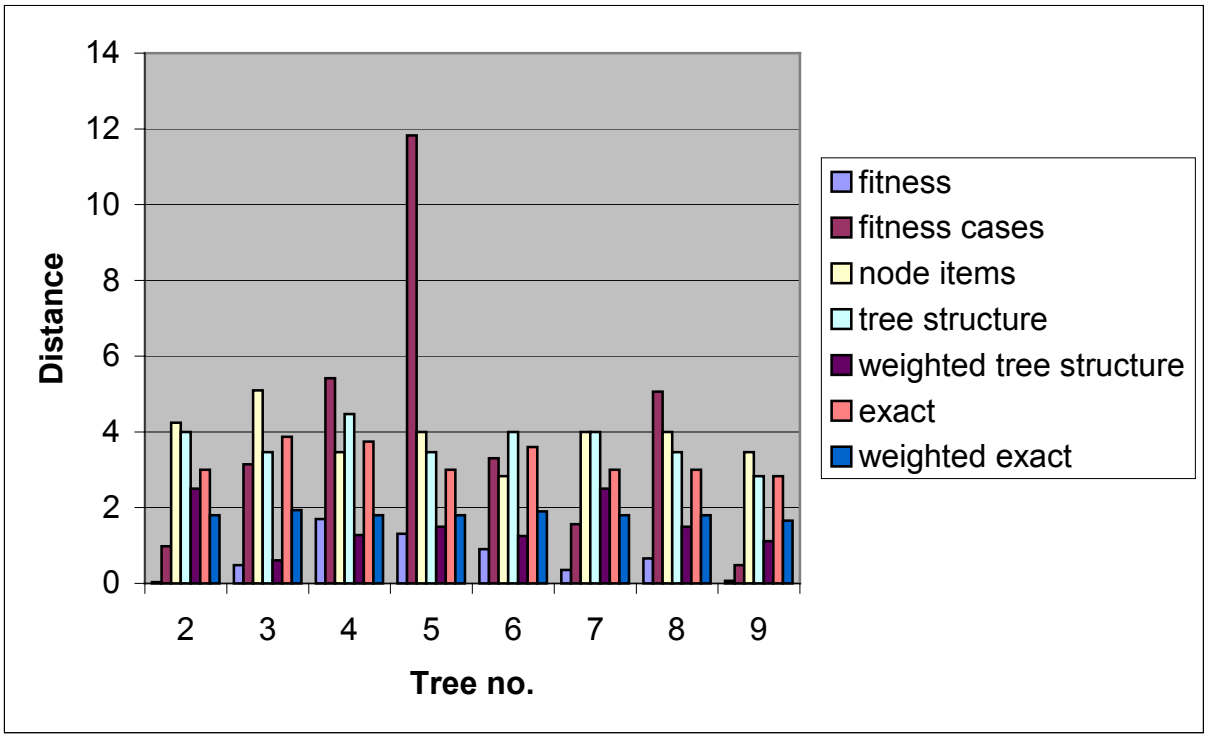

Figure 1. Distance from tree no. 1

References

[1] **D. Rubliauskas, G. Paulikas.** Flocking in Distributed Genetic Programming. *Information technology and control*, *Vol.*29, 2003, 14-18.

[2] **F. Fernandez, M. Tomassini, W.F. Punch III, J.M. Sanchez.** Experimental Study of Multipopulation Genetic Programming. 2000, 11 p.

[3] **C. Reynolds.** Boids, 1986, *http://www.red3d.com/cwr/boids.*

[4] **J. Kennedy, R. Eberhart.** Particle Swarm Optimization. *IEEE Int'l. Conf. on Neural Networks*, 1995.

[5] **A. Tatsukawa.** Ruby/GP documentation, 2004, *http://akimichi.homeunix.net/~emile/aki/program/gp/.*