GENETIC PROGRAMMING AND PERL PROGRAMMING LANGUAGE

Ing. Michal JUROSZ, Doctoral Degree Programme (1)
Dept. of Control and Instrumentation, FEEC, BUT
E-mail: xjuros00@stud.feec.vutbr.cz

Supervised by: Dr. Zdeněk Malec

ABSTRACT

This paper introduces a tree–based genetic programming with Perl programming language approach. Various selection methods, initialization procedures and genetic operators (crossover, mutation, permutation and editation) was implemented. Some test results (artificial ant problem) will be also presented.

1 INTRODUCTION

Genetic programming (GP) is simple and powerful machine learning technique inspirired by biological evolution that use stochastic evolutionar algorithms to on the fly discovering optimal or near-optimal topology of structures and values for all elements of structures.

Structures are mostly variable-length executable computer programs and fitness (score, quality of individual) is determined by ability to perform an user-defined computational task. Functions act as the branch points in the computer program tree, linking other functions or terminals. Terminals act as end (leaf) nodes. A terminal might be a variable, a constant or a function with no arguments.

According to J. Koza [1], the first experiments with GP were reported by Stephen F. Smith (1980) and Nichael L. Cramer (1985). Thanks to various improvements in GP technology and to exponencial grown in computer power, GP has started produce human-competive results, e.g. [2], [3], [4], and useful solutions to problems in domains where there are no known algorithms (an automated invention machine). The genetic programming algorithm has a high computational cost to run and has difficulty scaling to larger and harder problem instances. However, if the problem is hard, genetic programming can be effectively distributed. Each individual or sub-population (deme) can be evaluated on a separate processor.

Different GP approach, such as linear genetic programming (LGP), performs GP through direct manipulation of bytecode or binary machine code. This make GP sixty times faster than classic GP implemented in declarative programming languages (Lisp, Prolog) [4].

2 PERL PROGRAMMING LANGUAGE

Perl is open source programming language. Perl has been ported to over a hundred different platforms and is widely used in web development, finance and bioinformatics, and indeed in most sectors where a premium is placed on rapid development and the availability of a large number of standard and 3rd-party modules. Although Perl has most of the ease-of-use features of an interpreted language, it does not strictly interpret and execute source code one line at a time. Rather, Perl (the program) first compiles an entire program into an internal form (a parse tree) which is then optimized before being run (source wikipedia.org).

Perl has a run-time evaluation, dynamic data types (e.g. arrays and hashes with garbage collector) and a large collection of Perl modules and documentation (accesible through cpan.org) which make it easy to implement tree-based GP.

2.1 PERL MODULES FOR TREE-BASED GENETIC PROGRAMMING

Algorithm::Evolutionary [5], a set of classes for doing object-oriented evolutionary computation in Perl, was extended for tree-based genetic programming. New sub-packages e.g. Individual::ProgTree, Op::ProgTreeCreator, Op::ProgTreeReproduction, Op::ProgTreeCrossover and Op::ProgTreeMutation (contains methods for base mutation, full mutation, permutation and editation) was implemented.

3 EXAMPLE: ARTIFICIAL ANT

Symbolic regresion, artificial ant and parity problem are basic examples and test problems used in GP. Artificial ant problem (“Santa Fe” Trail) contains numerous solutions with a lot of symetry and is deceptive for genetic programming.

Set of functions is { if-food-ahead-then-else, block2, block3 } and set of terminals is { move_forward, turn_right, turn_left, do_nothing }. The goal of problem is to find best program, consisting of mentioned functions and terminals, to control ant behavior to find all 89 pieces of food along an trail. Trail on a two-dimensional grid is irregular (see Fig. 1). The terminals are used for move. The ant can sense if there is food on the square directly in front of the ant and function “if-food-ahead-then-else” is condition controlled by this ant sense. Fitness is defined as a number of food picked up till the end of search. Maximal number of move operations is 400.

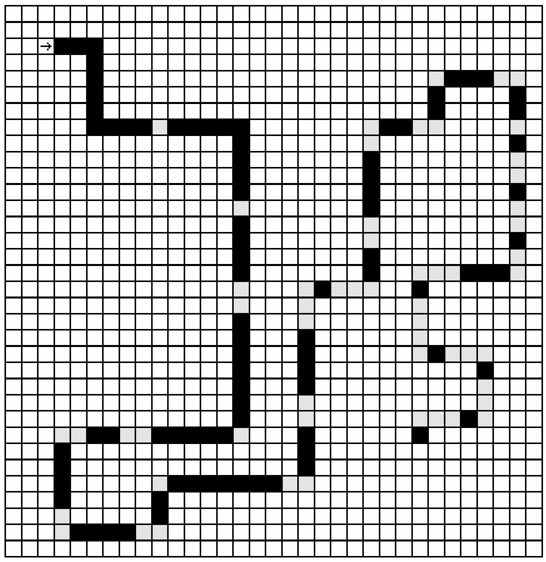

Fig. 1: *Map for artificial ant*

3.1 ARTIFICIAL ANT IMPLEMENTATION AND RESULTS

Genetic Programming generally, and GP with Perl Algorithm::Evolutionary is not exception, has a lot of parameters (for structure representation, initialization, fitness function, stop condition, selection method, genetic operations, and so on).

For experiment has been used this configuration (see [1] for detail explanation):

- maximum tree depth: 5
- initialization type: half-and-half
- fitness function: number of eaten pieces of food minus $10^{-4} \cdot$ (number of steps + number of turns)
- max sum of turns and steps: 400
- selection mechanism for crossover: two times random, one times proportional to fitness
- probability of mutation after crossover: 0.01
- operator rates: reproduction = 2, crossover = 7, mutation = 6, permutation = 4, full_mutation = 1, crossover node selection rate: (1,9,1)

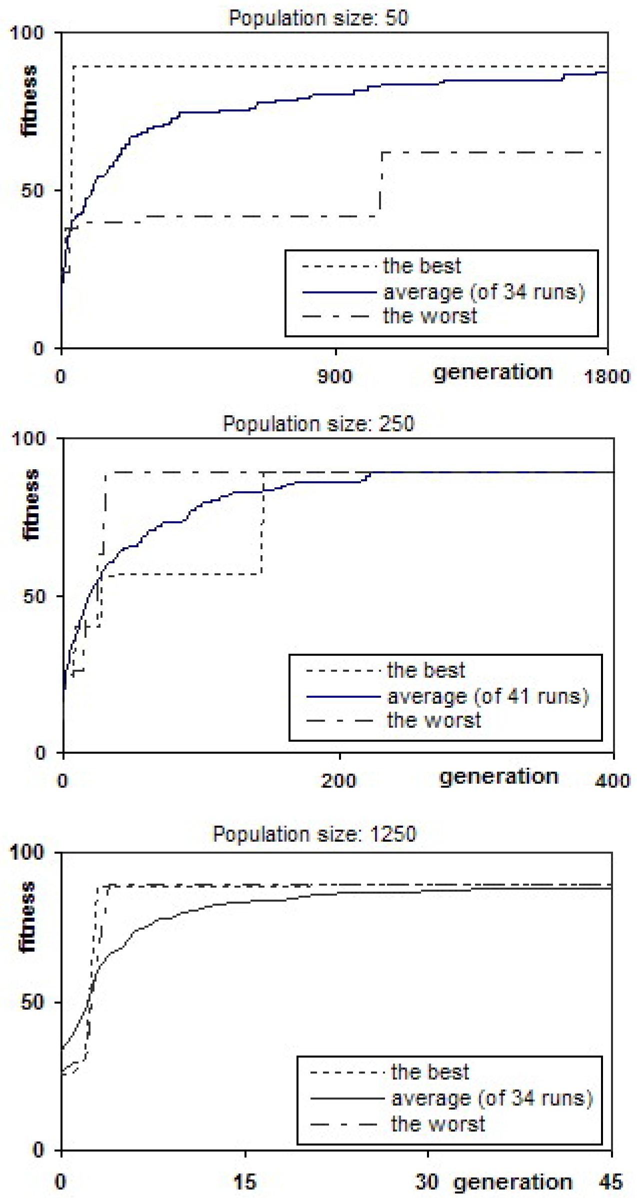

Fig. 2: *Statistics results for 3 different variable setups*

The experiment has been carried out with more than 30 separate runs on 3 different variable setups. First variable setup with small population size (50 individuals) has had the worst results. The worst run, from all 34 runs with 1800 generations in each, evolve artificial ant which can eat only 62 pieces of food (from 89 obtainable pieces). Second variable setup has fifth times greater population size. All 41 runs, each with 600 generations duration, has found out nearly the best solution. Computation with this configuration also founded the best ant code for used fitness function. Best ant picked up all 89 pieces of food and sum of used steps and turns to do that was 307 (see fig. 3). Runs with population size 1250 and maxima number of generations equal 200 has no significant better results.

4 CONSLUSION

Perl is dynamic programming language with runtime code evaluation, which has been useful for implementing flexible object oriented tree based genetic programming.

Experimental results suggest that fitness function, adequate population size and multiple runs are key to success. Genetic programming also need high-level rate for mutation operators, compared to genetic algorithms.

Editation is usefull non Darwinian operator, which speed computation and reduce program tree sizes.

```
# Fitness: 88.9693
if ( $bot->food_ahead() ) {
    ;
} else {
    $bot->right();
    if ( $bot->food_ahead() ) {
      $bot->forward();
    } else {
      $bot->left();
    }
}
$bot->forward();
if ( $bot->food_ahead() ) {
    ;
} else {
    $bot->left();
}
if ( $bot->food_ahead() ) {
    $bot->forward();
    $bot->forward();
} else {
    $bot->right();
}
```

Fig. 3: *The best solution*

REFERENCES

[1] Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT press, 1992

[2] Koza, J., Bennet, F., Andre, D., Keane, M.: Genetic Programming III, Morgan Kaufman, San Francisco, CA., 1999

[3] Koza, J., Keane, M., Yu, J., Bennet, F., Mydlowec, W.: Automatic Creation of Human-Competitive Programs and Controllers by Means of Genetic Programming, Genetic Programming and Evolvable Machines, Volume 1, Kluwer Academic Publishers, 2000

[4] Deschaine, L., Francone, F.: Comparison of Discipulus™ Linear Genetic Programming Software with Support Vector Machines, Classification Trees, Neural Networks and Human Experts, RML Technologies, Inc., www.aimlearning.com

[5] Algorithm::Evolutionary (Perl module), http://search.cpan.org/~jmerelo/Algorithm-Evolutionary/, 2005