Typewritten symbols recognition using Genetic Programming

Prof. Igor L. Bratchikov, Alexei A. Popov

Faculty of Applied Mathematics and Control Processes,
Saint-Petersburg State University

Universitetskii prospekt 35, Petergof, Saint-Petersburg, 198504, Russia

E-mail: {`braigor@yandex.ru, popov.lex@mail.ru`}

Abstract

Genetic programming is a new technique in Artificial Intelligence based on the evolutionary algorithms and inspired by biological evolution. As a matter of fact, genetic programming is a special case of genetic algorithms, where each individual is a computer program. Therefore, this technique could be used to optimize a population of computer programs to solve the problem.

This report demonstrates how genetic programming can be used to solve the problem of optical character recognition, specifically typewritten symbols. At present, there are many approaches to solve this problem, but all of them have their own limitations.

The approach given in this report could be successful at learning, maintaining and upgrading rules for typewritten symbols recognition, particularly in disputable situations. Specific fitness functions, terminals and functions that satisfy the requirements of a main problem were considered.

This research presents the successful application of genetic programming to a difficult and topical task.

1 Introduction

Genetic programming (GP) is a new technique in Artificial Intelligence based on the evolutionary algorithms and inspired by biological evolution. Basically it is a special case of genetic algorithms, where each individual is a computer program (usually represented in memory as a tree structures). This technique could be used to optimize a population of computer programs to solve the problem.

This paper investigates the use of genetic programming for typewritten symbols recognition. The term 'recognition' means the mechanical or electronic translation of scanned images of printed or typewritten symbols into machine-encoded text. In practice, it is extremely hard to generate, maintain and upgrade the system that would be successful in solving the problem of the character recognition especially as such system would give the human-competitive results. Generally there is a rule set for each symbol that is presumably true only for that symbol. The main point is that any changes in a current rule set have to be tested on very large sets to insure that all examples of the symbol are accepted and all others (wrong ones) are rejected. Therefore it would be great to design and develop the system that could easily, fluently and correctly recognize any typewritten characters.

The main purpose of the research is to estimate the application of GP for the problem of typewritten symbols recognition.

The principle goals are the following:

- To determine the superiorities (advantages) and disadvantages of GP in comparison with the other approaches;
- To design and develop the terminals and functions, fitness measure, certain parameters for controlling the run, the termination criteria and method for designating the results of the run.

GP has been successfully applied to the simple character recognition problem [2, 3, 4, 5, 6]. John Koza evolved programs that could recognize an 'I' and a 'L' letters using Boolean templates and control code for moving the templates. This approach involved low-resolution characters, e.g. 4x6 or 5x5. Later on Andre extended this approach by using programs that were good for recognition of digits using co-evolved two-dimensional feature detectors [1, 2]. However, this approach involved low-resolution symbols as well. So both approaches were good at recognition the limited symbol sets only.

Therefore the general task is to define whether GP is a successful technique in solving the problem of character recognition or not. And in case of positive result, the principal goal is to apply such technique by designing and developing the GP-based system that could solve the problem of typewritten symbol recognition. Moreover, such a system can be regarded as the first stage of the development of a system for recognition of handwritten symbols.

Let us consider the main principles of GP, its specificity, features and benefits.

2 Typewritten symbols recognition

2.1 What is GP?

Let us continue by saying a few words about GP. Basically it is an automated method for creating a computer program from a high-level problem statement of a specific task. Genetic programming starts from a statement of ”what needs to be done” and automatically creates a computer program to solve the problem. GP is a machine learning technique used to optimize a population of computer programs according to a fitness landscape designed to evaluate the program's ability to solve a given computational task [4].

GP evolves computer programs traditionally represented in memory as tree structures. Trees can be easily evaluated in a recursive manner. Every tree node has an operator function and every terminal node has an operand that makes mathematical expressions easy to evolve and evaluate. Thus traditionally GP favors the use of programming languages that naturally embody tree structures (for example, Lisp; other functional programming languages are also suitable).

The fact that genetic programming can evolve entities that are competitive with human-produced results suggests that genetic programming can be used as an automated invention machine to create new and useful patentable inventions.

2.2 How does it work?

One of the central challenges of computer science is to force a computer to do what needs to be done, without specifying how to do it. Genetic programming starts with a set of thousands of randomly created computer programs. This population of programs is progressively evolved over a series of generations. The evolutionary search uses the Darwinian principle of natural selection (survival of the fittest) and analogs of various naturally occurring operations, including crossover (sexual recombination), mutation, gene duplication, gene deletion. Sometimes genetic programming also employs developmental processes by which an embryo grows into fully developed organism.

Therefore GP is a domain-independent method that genetically breeds a population of computer programs to solve a problem. Specifically, such technique iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations. In addition, GP can automatically create a general solution to a problem in the form of a graphical structure whose nodes or edges represent components and where the parameter values of the components are specified by mathematical expressions containing free variables.

However to achieve the goal the user (human) has to specify the following preparatory steps:

1 the set of *terminals* (the independent variables of the problem) for each branch of the program;

2 the set of *functions* for each branch of the program;

3 the *fitness measure* (for implicitly or explicitly measuring the fitness of individuals in the population);

4 *certain parameters for controlling the run*;

5 *termination criterion* and *method for designating the result of the run.*

The figure below shows the aforesaid preparatory steps for the basic version of genetic programming. Those steps are the user-supplied input to the GP-system. The computer program represents the output of the genetic programming system.

Genetic programming iteratively transforms a population of computer programs into a new generation of the population by applying analogs of naturally occurring genetic operations. These operations are applied to some individuals selected from the population. They are stochastically selected to participate in the genetic operations based on their fitness (the third preparatory step). The iterative transformation of the population is executed inside the main generational loop of the run of genetic programming.

The first two preparatory steps specify the ingredients that are available to create the computer programs. A run of genetic programming is

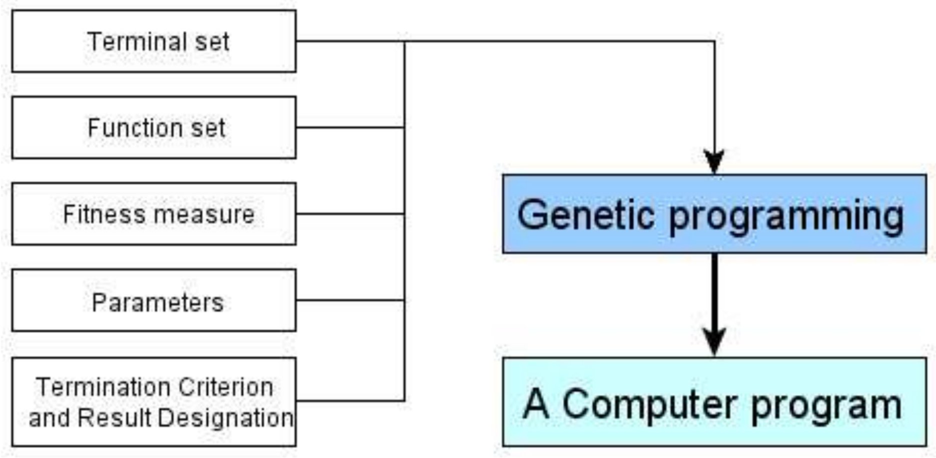

Figure 1: How does GP work

a competitive search among a diverse population of programs composed of the available functions and terminals.

The important thing to remember is that GP is a highly iterative process. It involves nested loops of procedures. The goal is to evolve successively better individuals with every generation. Theoretically we might want to run GP indefinitely, but due to limited computational power, we need to define some kind of termination criteria.

Generally the termination criteria has two parts, a successful fitness or a maximum number of generations. We provide GP with some number and say, if you evolve an individual with fitness better than or equal to that number, accept it and stop the run. This means that we have found a solution that is good enough. Alternatively, we want to stop GP from running too long if it is not progressing. We find that the probability of making further progress drops with number of generations passed, so we define some maximum number of generations. If we haven't succeeded by so many generations, it might be time to stop and try again.

The whole procedure can be described as follows:

1. Randomly create an initial population (generation 0) of individual computer programs composed of the available functions and terminals.

2. Iteratively perform the following sub-steps (called a generation) on the population until the termination criterion is satisfied:

   a Execute each program in the population and ascertain its fitness (explicitly or implicitly) using the problem's fitness measure.

b Select one or two individual program(s) from the population with a probability based on fitness (with reselection allowed) to participate in the genetic operations in (c).

c Create new individual program(s) for the population by applying the following genetic operations with specified probabilities:

I *Reproduction*: Copy the selected individual program to the new population.

II *Crossover*: Create new offspring program(s) for the new population by recombining randomly chosen parts from two selected programs.

III *Mutation*: Create one new offspring program for the new population by randomly mutating a randomly chosen part of one selected program.

IV *Architecture-altering operations*: Choose an architecture-altering operation from the available repertoire of such operations and create one new offspring program for the new population by applying the chosen architecture-altering operation to one selected program.

3. After the termination criterion is satisfied, the single best program in the population produced during the run (the best-so-far individual) is harvested and designated as the result of the run. If the run is successful, the result may be a solution (or approximate solution) to the problem.

2.3 Adding GP to the problem

Please see the flowchart of GP in Figure 2.

Now let's get back to our research, and especially to the character recognition problem. The data source was database of typewritten symbols. It consists of testing and training subsets.

The symbols are centered and represented by a matrix of black and white pixels (20 pixels wide and 30 pixels in height). The first step in the recognition of any symbol is to extract the boundary pixels from the bitmap. This can be done using a quick one-pass raster scan method [1] Each element in a list contains row and column information for a boundary pixel. The second step is to close holes that are small enough to be accounted for by noise. The outer boundary of the symbol is then split into four segments (left, right, bottom and top). Therefore, the maximum and minimum row

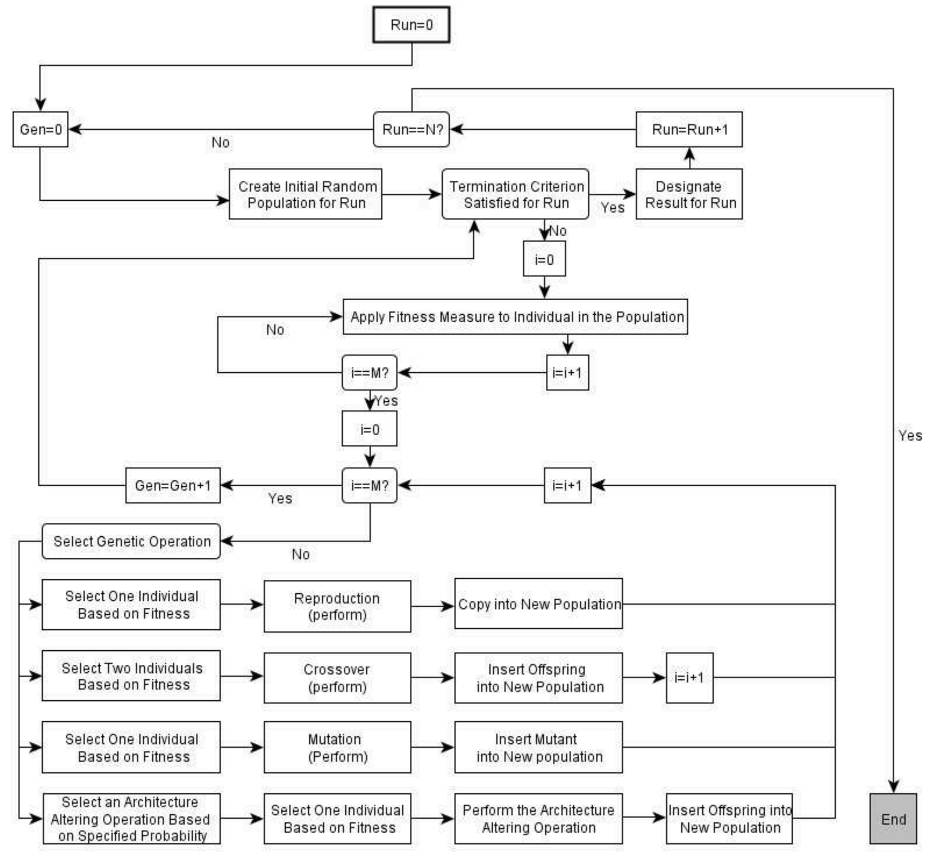
Run=0
Gen=0
No
Run==N?
Run=Run+1
Create Initial Random Population for Run
Termination Criterion Satisfied for Run
Yes
Designate Result for Run
No
i=0
Apply Fitness Measure to Individual in the Population
No
i==M?
i=i+1
Yes
i=0
Gen=Gen+1
Yes
i==M?
i=i+1
Yes
Select Genetic Operation
No
Select One Individual Based on Fitness
Reproduction (perform)
Copy into New Population
Select Two Individuals Based on Fitness
Crossover (perform)
Insert Offspring into New Population
i=i+1
Select One Individual Based on Fitness
Mutation (Perform)
Insert Mutant into New population
Select an Architecture Altering Operation Based on Specified Probability
Select One Individual Based on Fitness
Perform the Architecture Altering Operation
Insert Offspring into New Population
End

Figure 2: Typical GP scheme

and column are calculated for each hole, for each segment, and for the symbol as a whole. In addition, the number of pixels in each hole stored, and the segments are ranked according to their number of pixels.

It is assumed that solutions are always comparable and that, given a pair of them, we are always able to point the better one, unless they have the same value of the evaluation function.

For some hypothesis the evaluation function returns its accuracy of classification on the training set. Incomparability involves a partial order in the solution space and the possibility of existence of many best solutions at the same time. We can prevent the algorithm from losing good solutions by replacing the scalar evaluation function with a pairwise comparison of solutions.

Let's define formally the outranking relation between two solutions (hypotheses), given the sets of examples correctly classified by these hypotheses. Outranking means that first hypothesis is at least as good as a second one [5]. This condition has to hold separately and simultaneously for examples representing some decision classes. Therefore someone might ask an obvious question, how to select the best solutions?

Tournament selection scheme cannot work properly in solving this problem due to the fact, that the incomparability decreases the selection pressure, so some tournaments might remain undecided. Therefore we have to select some non-outranked solutions (hypotheses).

The solutions (programs-candidates) performing image analysis and recognition are evaluated on a set of training cases (images), called fitness cases. Let us now consider some estimated values. Thus, the population size was 2000 (that's a quite enough indeed). The probability of mutation equals to 0.05 (it is a common or standard value). The value of maximal depth of a randomly generated tree (initialization): 3 or 4 (it depends in common case). Maximal number of generations: 100 (stopping condition; in some cases this value could be increased or decreased). Training set size equals to 200 (100 images per each class). We used the tournament selection which means that the selection works by selecting a number of individuals from the population at random, a tournament, and then selecting only the best of those individuals. Tournament sizes tend to be small relative to the population size. The ratio of tournament size to population size can be used as a measure of selective pressure. Note that a tournament size of 1 would be equivalent to selecting individuals at random and a tournament size equal to the population size would be equivalent to selecting the best individual at any given point. The selective pressure of tournament selection can be adjusted by means of the tournament size parameter, which makes it a more flexible selective procedure than fitness-proportional selection [4]. Therefore, the tournament selection works by creating a tight selective pressure on a small local group of individuals. Neither of these two selection procedures is better than the other for every problem. There are also a whole slew of other selection procedures that may or may not be based on either of these.

Because we only care about whether one individual is better than another, to save processing, we only need to consider standardized fitness for this selection procedure. Recall that better individuals have lower standard fitnesses.

3 Conclusion

In conclusion we would like to mention that GP has some evident superiority in comparison with the other approaches such statistics, neural networks and the other techniques, though it is not an ideal approach to solve the problem. In theory it could be successfully used simultaneously with the other methods in some disputable issues.

The main result obtained in the experiment is that the aforesaid search technique solves the problem in common cases. In addition the accuracy of classification on both testing and training sets was increased, although the results did deviate sometimes. Furthermore the results were false positive or negative at times. The system recognized the given symbols as a rule except some complicated cases. For example, the recognition of the digit '0' and letter 'O'; or the recognition of the digit '1' and letter 'l'. Another interesting fact is that the good solutions were commonly defended from discarding. Therefore we expect that the better results will be obtained after performing some modifications.

Although the research is made, its subject could be extended. There are some problems and aspects that were not consider in the current paper. First of all it will be great to make a deskewing and font normalization of the characters before their recognition. The second main task in perspective is to design and develop the recognition system (programming complex or toolbox). Once it is done it will be a big improvement to recognize not only the typewritten symbols but also handwritten characters. It will also be interesting to try to use in practice the approach given in this paper simultaneously with the other techniques such as the neural networks and the other techniques.

Bibliography

[1] Andre D., *A fast one pass raster-scan method for boundary extraction in binary images.* Canon Research Center Technical Report, Palo Alto, California, 1993.

[2] Andre D., *Learning and upgrading rules for an OCR system using Genetic Programming.* Stanford University and Canon Research Center of America, Stanford, 1994.

[3] Breunig M. M., *Location independent pattern recognition using Genetic Programming.* In Koza, J. R., editor, Genetic Algorithms and Genetic Programming at Stanford, Stanford Bookstore, Stanford University, 1995.

[4] Koza J. R., *Genetic Programming: on the programming of computers by means of natural selection.* MIT Press, Cambridge 1994.

[5] Krawiec K., *Genetic Programming using partial order of solutions for pattern recognition tasks.* Proceedings of II National Conference 'Computer Recognition Systems' KOSYR'2001, pp. 427–433.

[6] Poli R., *Genetic Programming for image analysis.* School of Computer Science, The University of Birmingham, 1996