J. Software Engineering & Applications, 2009, 2: 354-360
doi:10.4236/jsea.2009.25047 Published Online December 2009 (http://www.SciRP.org/journal/jsea)

Improved Genetic Programming Algorithm Applied to Symbolic Regression and Software Reliability Modeling

Yongqiang ZHANG[1], Huifang CHENG[1], Ruilan YUAN[2]

[1]School of Information and Electric Engineering, Hebei University of Engineering, Handan, China; [2]College of Arts, Hebei University of Engineering, Handan, China.
Email: yqzhang@hebeu.edu.cn, hfcheng4@163.com, yuanruilan_2000@sina.com

Received June 24th, 2009; revised September 7th, 2009; accepted September 16th, 2009.

ABSTRACT

The present study aims at improving the ability of the canonical genetic programming algorithm to solve problems, and describes an improved genetic programming (IGP). The proposed method can be described as follows: the first investigates initializing population, the second investigates reproduction operator, the third investigates crossover operator, and the fourth investigates mutation operation. The IGP is examined in two domains and the results suggest that the IGP is more effective and more efficient than the canonical one applied in different domains.

Keywords: *Improved Genetic Programming, Symbolic Regression, Software Reliability Model*

1. Introduction

Genetic Programming (GP) is an automated method for creating a working computer program from a high-level problem statement of a problem [1,2]. It is a technique pioneered by John Koza [3] which enables computers to solve problems without being explicitly programmed and based on the idea of genetic algorithms presented by John Holland [4]. The goal is to use the concepts of Darwin evolution theory for computer program induction. The concepts are usually applied by genetic operators, such as selection, crossover, mutation and reproduction [5].

In Genetic Programming solutions to a problem are represented as syntactic trees (or symbolic expressions), which are evolved in a population of programs towards an effective solution to specific problems according to Darwinism. The flexibility and expressiveness of computer program representation, combined with the powerful capability of evolutionary search, makes GP a promising method to solve a great variety of problems [6].

1.1 Goals

To summarize, the learning/evolutionary process of the canonical GP algorithm has at least the following problems: the scale of the population is usually quit large and the convergence of the algorithm is very slow [7]; the evolved programs are usually too big and contain a large number of redundancy; programs within the initial population are generated randomly; setting the algorithm parameter depends on the experience. To overcome these problems, the overall goal of this paper is to investigate a new approach to improve the efficiency of GP algorithm. Specifically, we will investigate whether the new approach outperforms the canonical GP in terms of data fitting performance and training time in the evolutionary process.

1.2 Structure

The rest of the paper is organized as follows. Section 2 gives a brief overview of the canonical GP algorithm. Section 3 describes four developments to improve GP performance. Section 4 describes the experiment design and configurations. Section 5 is application. Section 6 draws the conclusions and gives future directions.

2. The Canonical GP Algorithm

Genetic Programming starts with an initial population of randomly created programs composed of functions and terminals appropriate to the problem domain [1,2]. Then all programs in the population are evaluated in terms of how well it performs in the particular problem environment. This evaluation is called fitness measure. According to the theory of survival of the fittest, genetic operations are used to create a new offspring population of programs from the current population [8]. But the pa-

rameters of genetic programming should be set in advance: those include the probability of reproduction, crossover, and mutation. Over many generations, the exact or the optimal solution will be found [9].

3. Algorithm Optimization

3.1 Problems of Canonical GP Algorithm

Problems of canonical GP algorithm are listed as follows. Firstly, the quality of the randomly created by initial programs can not be guaranteed against bad. Secondly, when the number of programs in the population is bigger, the convergence of the algorithm can be very slow, and one or two programs will be chosen randomly for reproduction, mutation or crossover during the evolutionary process. If the selected programs are too big, the evolution for the rest of the leaning process will be slow. More importantly, the good building blocks in these big programs will have a much greater chance to be destroyed than in the small ones, which could lead to poor solutions by the evolutionary process.

3.2 Improvement

Create a population of programs according to qualification. The concrete method is described as follows:

Step 1, set the scale of the population as M and then produce the initial population randomly with scale of MM bigger than M.

Step 2, retain M programs with better fitness in the population to replace the initial population.

Regarding this initial population creation process the selection of MM (big scale) has the direct impact on the convergence rate. If there is little difference between MM and M, the created initial population will be similar to the random product one. If there is a great deal of difference between the MM and M, the initial population creation process will need a long computing time. We set MM: M=4:3.

In Genetic Programming reproduction that copies better programs to the next generation, it gives expression to survival of the fitness [2]. Different reproduction method has different selection intensity. Generally speaking, the reproduction operator structure gives one kind of choice plan, which makes better programs in the current population easily enter the next generation population. In the canonical roulette wheel selection method, an individual in a population will be selected according to the proportion of its own fitness to the total sum of the fitness of all the programs in the population [10]. Namely programs with low fitness scores will have a low probability to be reproduced. Programs which perform particularly well will have a very high probability of being selected. But roulette wheel selection has its deficiency: on the one hand the roulette wheel selection has not selected all better programs. Some programs with high fitness scores have not the possibility to be duplicated to the next generation population, and even some inferior programs are possibly to be selected into the next generation. On the other hand, programs with good building blocks and low fitness scores have the possibility not to be duplicated. It easily arises the phenomenon of "premature". Premature means that evolution is converging in the local optima, but it is not converging in the overall optima.

In order to avoid the defects of roulette wheel selection method while retaining its advantages, a new method for improving reproduction operator is proposed as follows. Add the optimum programs to the next generation of the first category gene pool of the current population. Programs in the current population besides the first category gene pool are to be selected by the roulette wheel selection to the next generation population. Programs in the first category gene pool are to be copied to the next generation.

Crossover combines the genetic material of two parents by swapping a part of one parent with a part of the other [2]. With a tree-based representation, replacing a node means replacing the whole branch. This adds greater effectiveness to the crossover operator. The expressions resulting from crossover are greatly different from their initial parents. Therefore, it has the necessity to use the pre-selection mechanism to choose programs for crossover only when the new programs fitness scores are higher than the father programs' they can replace the older ones into the next generation, otherwise carries out the crossover operator again. Because of the structure similarity between the new programs and the replaced father programs', the genetic material of two parents is replaced by the same structure of the programs. Therefore, it can effectively keep the diversity of the population in evolution. The algorithm is more hopeful at finding the best individual in the whole population.

Mutation affects an individual in the population. Performing mutation operator is beneficial to form the diversity of the population and avoid premature [1]. The mutation probability is quite small, taking 0~0.05 generally. The mutation operator randomly selects a point in the tree and replaces the existing sub-tree at that point with a new randomly generated sub-tree. If the replaced node is a function, the node has the same number arguments is selected from function set. If the replaced node is a terminal node, then terminal is selected to form the leaf node [2].Thus fitness of the selected tree is changed. A worse individual is very likely to become better and a better individual is also very likely to become worse after performing mutation operator. Two ways have been adopted to perform mutation operator. In initial stage it replaces a whole node in the selected individual, that is to say, the operator removes a random sub-tree of a selected individual, and then puts a new sub-tree in the same

place. In later stage programs tends to be good, then it replaces just the node's information.

4. Experiments Design and Configurations

4.1 Program Representation and Generation, and Genetic Operators

In this Improved Genetic Programming (IGP), we use tree structures to represent genetic programs. The ramped half-and-half method [1] is used for generating the programs restricted by an initial maximum depth in the initial population. In the mutation operator and crossover operator programs are restricted by a maximum depth.

The maximum size allowed for a program is set as one of the principal parameters of a GP run to limit the depth of the program tree in crossover operator and mutation operator, and that may control the redundancy expression for individual to a certain degree, also may reduce computing time greatly.

Pre-selection mechanism, changing mutation intensity in mutation operator, is used in learning process. The elitist and roulette wheel selection mechanism is used in crossover operator and reproduction operator [11].

More details of program representation and generation methods and the genetic operators are described in Section 3.

4.2 Function and Terminal Sets

In the function set, the five standard arithmetic operators and seven math functions are used to form the non-terminal nodes:

FuncSet ={+,–,.*, ./, .^, sin, cos, tan, cot, exp, log,sqrt}.

The +, and—operators have their usual meanings——addition and subtraction, while.*, ./, .^ represents the variable with correspondence matrix elements carrying on multiplication, left division and involution. The programs are developed in MATLAB7.0.

Table 1. Basic parameters of Genetic Programming

Parameters	Values	Parameters	Values
population-size	800	internal-node-rate	0.9
initial population method	half-and-half method	initial-max-depth	5
crossover-rate	0.70	max-depth after mutation	7
mutation-rate	0.05	max-depth after mutation	7
reproduction-rate	0.05	max-generations	100

4.3 Parameters and Termination Criteria

In canonical GP the probabilities of selection, crossover and mutation are set in advance depending on the experience. The evolution process carries on under the predetermined probability from start to the end. In the IGP all control parameters of genetic programming algorithm are optimized and combined by use of the orthogonal experiment method. Orthogonal experiment is a method of studying the multi-factor multi-levels design experiment, and this method can curb the blindness during accessing parameter in GP and let us obtain the scientific experimental results by a few testing sequences. The basic parameter values used in this approach are shown in Table 1.

The evolutionary process runs for a fixed number (max-generation) of generations unless it finds a solution or a program is close enough to the desired solution.

5. Application

5.1 Applied to Symbolic Regression

The IGP has been tested by two considerable complexity examples, and under the same platform and precision compared to the original algorithm.

Function 1

$$y = x^4 + x^3 + x^2 + x \tag{1}$$

Function 2

$$y = \sin 3x + \sin 2x + \sin x \tag{2}$$

For reducing the factor influence as far as possible in the IGP evolution process, the experiments are repeated 50 times for all cases and the average results are shown in Figure 1~Figure 4. These results are compared with the canonical GP and the IGP using same instances.

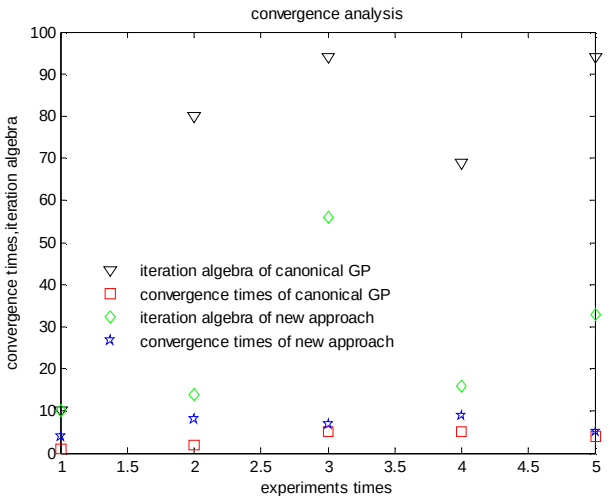

Figure 1. Compairson convergence between GP and the new approach (Function 1)

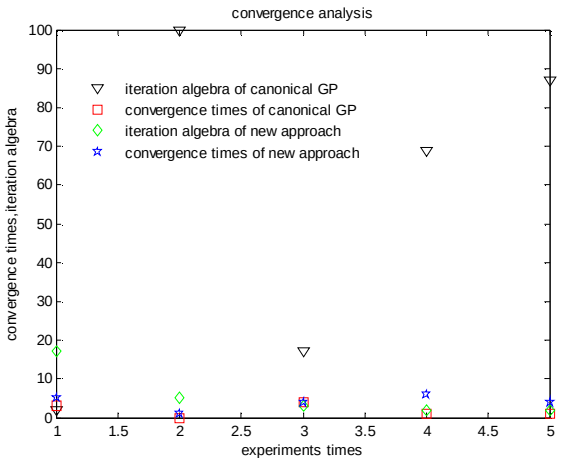

Figure 2. Comparison convergence between GP and the new approach (Function 2)

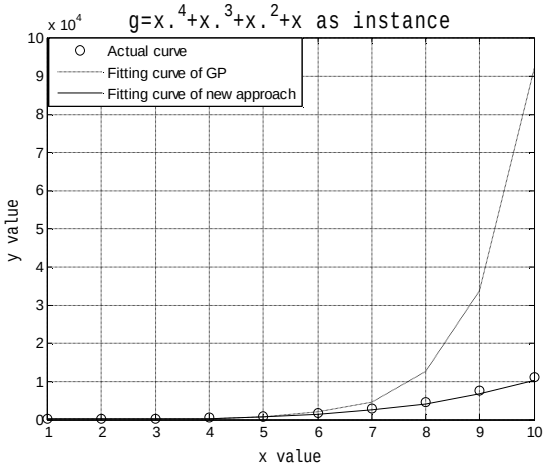

Figure 3. Simulation result of GP and the new approach (Function 1)

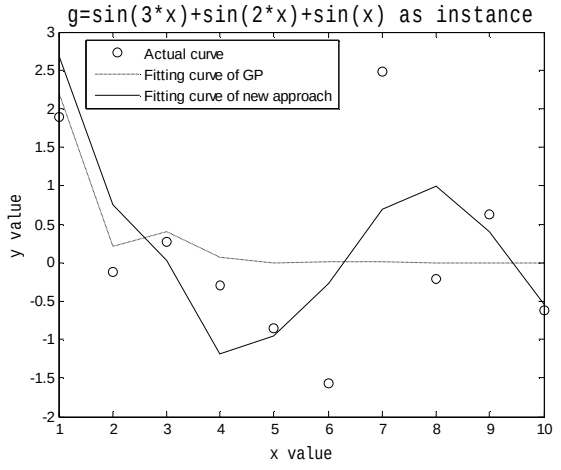

Figure 4. Simulation result of GP and the new approach (Function 2)

Fitting functions expression:

Function 1. GP fitting expression

$$y = (x - \sin x)^{\sqrt{x}} \log x + 4e^{x}$$

Function 1. IGP fitting expression

$$y = x + \tan(x\cot(x - \cot(x - e^{x\sqrt{x}}))) + x^{4}$$

Function 2. GP fitting expression

$$y = (x - \cot(x^{\cos x} + \sqrt{x})^{\tan(x^{\sin x \tan x})}) / e^{x}$$

Function 2. IGP fitting expression

$$y = x^{\tan x} / \log(xe^{x^{\sqrt{x}}}) / \cos x + \sin x$$

As seen from Figure 1 ~ Figure 4, the results demonstrate the ability of the IGP in this paper in the iterative algebra, the restraining time and the smooth fitting, and all have the distinct improvement. The results demonstrate the ability of the improved algorithm which is more efficient in evolving good programs for best solutions in symbolic regression.

5.2 Applied to Software Reliability Modeling

In the software qualitative target system, software reliability is the most important inherent characteristics. Software reliability models are the basis of quantitative analysis, and through the model the software reliability can be quantitatively assessed and predicted, and then estimate the delivery of the date, adjust the distribution of resources to determine whether the software has reached a predetermined reliability requirements and so on.

Among several models proposed in the literature, GP evolutionary model has its own characteristics:

- Retain all known information, directly involved in the quantitative calculations, and minimize the accumulation of errors;
- In addition to the original data, there are no factitious assumptions to maximize the faithful to a given data.

Software reliability modeling can be classified into two classes, based on whether the models center on failure times or failure counts. In this paper, we mainly focus on failure times. During the testing process when the failure data are collected, they can come in the format of failure times, i.e., (t1,t2,…,tn). ti is the time when *i*th failure is observed. We give two examples, one failure data series (in Table 2) comes from a software which is from armored force of engineering university, and the other (in Table 3) comes from NTDS(Naval Tactical Data System) of U.S. Navy Fleet Computer Programming Center).

In the following two tables x is sequence number and t_i is the cumulative failure time series.

- Calculation of reliability parameter

Table 2. Failure data series [12]

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
x_i	1	1	1	5	4	24	6	14	33	1	30	22	13	22	77	7
t_i	1	2	3	8	12	36	42	56	89	90	120	142	155	177	254	261

Table 3. Failure data series (NTDS) [13]

x	t_i	x	t_i	x	t_i
1	9	11	71	21	116
2	21	12	77	22	149
3	32	13	78	23	156
4	36	14	87	24	247
5	43	15	91	25	249
6	45	16	92	26	250
7	50	17	95	27	337
8	58	18	98	28	384
9	63	19	104	29	396
10	70	20	105	30	405

Under the same platform and precision compared to the original algorithm, after a 100-generation evolution of fitness computing to be better models for (x in Table 2-3 in the software failure time measured, and x> 1):

$$T_{GP} = f(x) = x \cdot \sqrt{x} \cdot \sqrt{x - \cos\left(\tan x \cdot \left(x - \ln x + \sin\left(x^{(\cot(x))}\right)\right)\right)}$$

$$T_{IGP} = f(x)_{IGP} = x\sqrt{x}\sqrt{x - \cos((x^{\sin x} + \cos x)/\cot(x + \tan x))}$$

Cumulative failure time Calculated by the GP model t17= 292.1355, t16 time the average time between failures MTBF = 31.1355. Cumulative failure time Calculated by the IGP model t17= 297.0135, t16 time the average time between failures MTBF = 31.0436. The cumulative failure time of the observed results t17=300, t16 time the average failure time MTBF = 39.

$$T_{NTDS} = f(x) = x + \sqrt{\left(x - \sin\left(\left(x - e^x\right)/\cot x\right) \cdot \tan x\right)^{\ln x}}$$

$$T_{NTDS-IGP} = f(x) = 1.2x\log^2 x$$

Cumulative failure time Calculated by the GP model $t_{27} = 315.22$, t26 time the average time between failures MTBF = 65.22. The cumulative failure time of the observed results t27=337, t26 time the average failure time MTBF = 47.

To summarize, the results from the experiment show that software reliability models are established by IGP having relatively good predictive power for one step.

u Failure curve

The initial failure rate is 0.59948, 0.6447 respectively calculated by the model (TGP), (TIGP). The current failure rate (t = 261) of the Software is 0.048614, 0.0038378 respectively. The software failure curve is shown below.

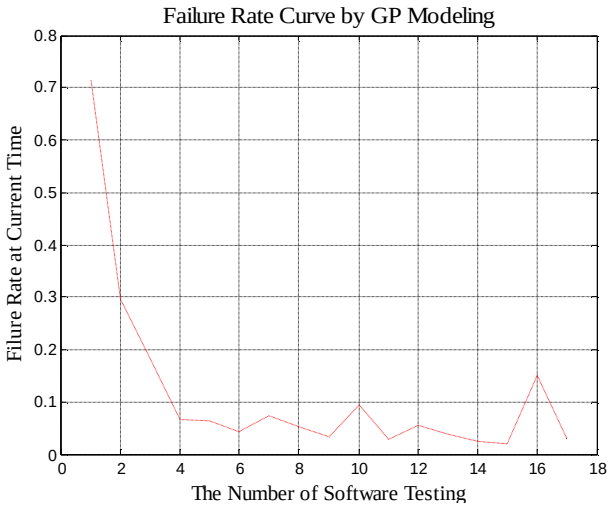

Figure 5. Failure curve of (T_{GP}) model

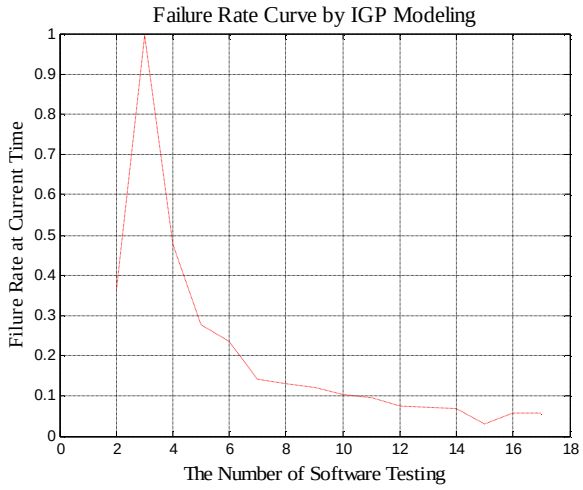

Figure 6. Failure curve of (T_{IGP}) model

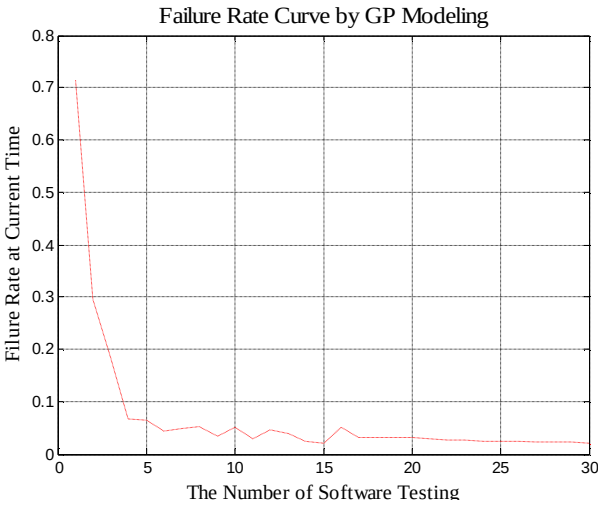

Figure 7. Failure curve of (T_{NTDS}) model

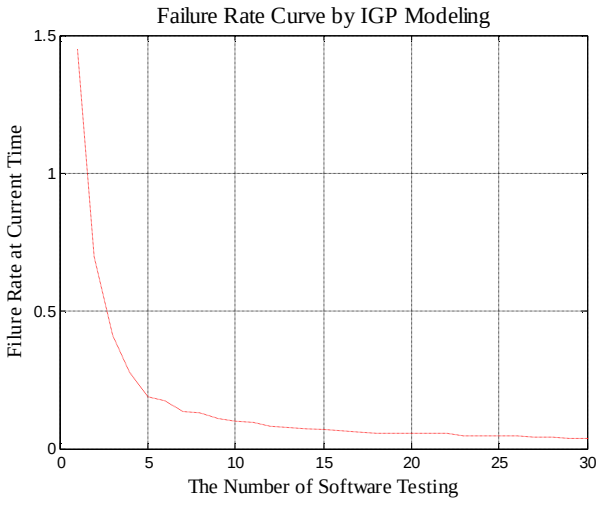

Figure 8. Failure curve of ($T_{NTDS-IGP}$) model

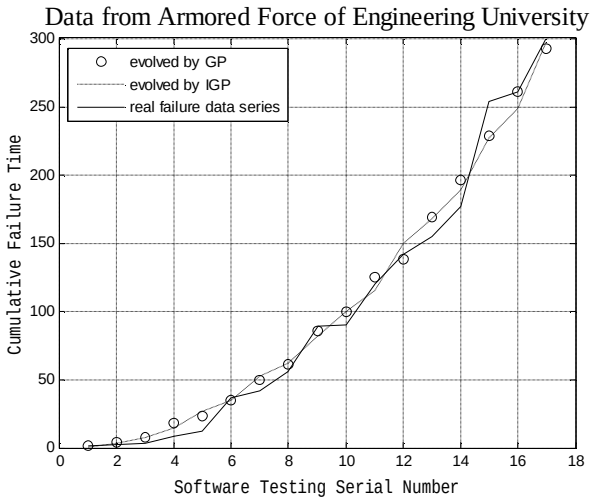

Figure 9. Simulation result of GP and IGP model

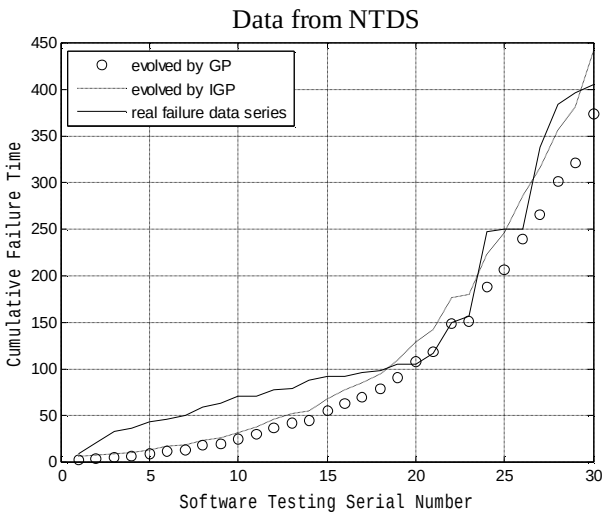

Figure 10. Simulation result of GP and IGP model

The initial failure rate is 0.5、0.24869 respectively calculated by the model (TNTDS), (TNTDS-IGP). The current failure rate (t = 405) of the Software is 0.0026775、0.0022268 respectively. The software failure curve is shown above.

u Model simulation

Simulation studies are presented to validate the models. The results show that the models set up by IGP are better than the models set up by the canonical algorithms.

According to the theoretical and experimental analysis, our IGP algorithm not only provides the same excellent performance as the canonical GP, but also can save considerable convergence time against canonical GP.

6. Conclusions

GP is a powerful paradigm that can be used to solve dif-

ferent problems in several domains. However, the evolutionary time is quite long. The goal of this paper is to investigate ways to improve the power of GP algorithm.

We describe fitness-based method to generate initial population, elitist and roulette wheel selection mechanism and pre-selection mechanism used in crossover operator, and set a limit to depth of the program tree in crossover operator and changing mutation intensity in mutation operator. The approach was examined in two symbolic regression experiments and two software reliability modeling all achieved much better results for several problems in different domains than the canonical one.

The improved genetic programming has accelerated the speed of the GP convergence and avoided the traps of local optima to a great extent. We will investigate whether the performance on the smooth fit can be improved more by using some math tools.

7. Acknowledgments

The authors wish to thank the Natural Science Foundation of Hebei province (Project No.F2008000752) under which the present work was possible.

REFERENCES

[1] H. Wright, "Introduction to Genetic Programming," USA: University of Montana, CS555/495:1-2, 2002.

[2] Wolfgang, N. Peter, E. K. Robert and D. F. Frank, "Genetic Programming: an introduction on the automatic evolution of computer programs and its applications," San Francisco, Calif: Morgan Kaufmann Publishers: Heidelberg: Dpunkt-verlag. Subject: Genetic Programming (Computer science): ISBN: 1-55860-510-X, 1998.

[3] J. R. Koza, "Genetic Programming Ⅱ: automatic discovery of reusable programs" , Cambridge M A :M IT Press, 1994

[4] J. H. Holland, "Adaptation in natural and artificial Systems," MIT Press, 1992.

[5] R. V. Silvia and P. Aurora, "A grammar—guided Genetic Programming framework configured for data mining and software testing," International Journal of Software Engineering and Knowledge Engineering, Vol. 16, No. 2, pp. 245–267, 2006.

[6] W. Banzhaf, P. Nordin, R. E. Keller and F. D. Francone, "Genetic Programming-an introduction on the automatic evolution of computer programs and its application," Morgan Kaufmann Publishers, Inc, 1998.

[7] J. L. Bradley, M. Sudhakar and P. Jens, "Program optimization for faster genetic programming," In Genetic Programming-GP'98, pp. 202–207, Madison, Wisconsin, July 1998.

[8] M. D. Kramer and D. Zhang, "GAPS: A genetic programming system," The Twenty-Fourth Annual International Computer Software and Applications Conference, pp. 614–619, 2000.

[9] A. P. Fraser, "Genetic Programming in C++[R]," Cybernetics Research Institute, TR0140, University of Sanford, 1994.

[10] A. Augusto, "symbolic regression via Genetic Programming," VI Brazilian Symposium on Neural Network, pp. 173–178, 2000.

[11] T. Walter, "Recombination, selection, and the genetic construction of computer programs," PhD thesis. Faculty of the Graduate School, University of Southern California, Canoga Patch, California, USA, April 1994.

[12] Y. Gong and Q. Zhou, "A software test report SRTP," Armored Force Engineering Institute, China, 1995.

[13] X. Huang, "Software reliability, safety and quality assurance," Beijing: Electronics Industry Press, Vol. 10, pp. 9–20, 15–17, 86–112, 2002.

