A Niching Gene Expression Programming Algorithm Based on Parallel Model

Yishen Lin, Hong Peng, and Jia Wei

School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510641, China
Linnys@gmail.com, mahpeng@scut.edu.cn, wei.jia@mail.scut.edu.cn

Abstract. GEP is a biologically motivated machine learning technique used to solve complex multitude problems. Similar to other evolution algorithms, GEP is slow when dealing with a large number of population. Considering that the parallel GEP has great efficiency and the niching method can keep diversity in the process of exploring evolution, a niching GEP algorithm based on parallel model is presented and discussed in this paper. In this algorithm, dividing the population to the niche nodes in sub-populations can solves the same problem in less computation time than it would take on a single process. Experimental results on sequence induction, function finding and sunspot prediction demonstrate its advantages and show that the proposed method takes less computation time but with higher accuracy.

1 Introduction

Natural biological systems are well adapted to the environment; they can be used to solve many complex multitude problems. Inspired by the process of biological evolution in natural systems, evolutionary methods of algorithm designs are applied to stochastic searches for optimal results.

Gene Expression Programming (GEP) was first introduced by Candida Ferreira [1]. It combines the characteristics of Genetic Algorithms (GA) and Genetic Programming (GP), and overcomes some drawbacks of them. It has performed well for solving a large variety of problems, including symbolic regression, optimization, time series analysis, classification, logic synthesis and cellular automata, etc [1, 2, 3 and 4]. The GEP algorithm is a robust but slow process with a large number of individuals and complex multitude problems. Parallel execution is a better method to reduce computation time and to improve the efficiency in evolution algorithm. There are many studies in parallel GA [5, 6] and parallel GP [7, 8], but there are few studies in parallel GEP [9].

In this paper, a new algorithm called PNGEP (Parallel Niching GEP) which combines parallel model and niching method is presented. Experimental results on sequence induction, function finding and sunspot prediction show that this new algorithm gets better performance and higher efficiency than the basic GEP.

M. Xu et al. (Eds.): APPT 2007, LNCS 4847, pp. 261–270, 2007.

2 Related Works

Basic GEP can get good results in regression and prediction problem [1, 2, 3 and 4]. Niching method is a biologically technology, using this technology in evolution can get higher efficiency [10, 11 and 12]. However, similar to other evolution algorithms, GEP is also slow when dealing with a large number of individuals and complex multitude problems. To solve this problem, some researches have imported parallel model in evolution algorithm, and the hybrid algorithm has better performance [5, 6, 7, 8 and 9].

2.1 Niching Method

Niching method is widely used in GAs like Niching Genetic Algorithm (NGA). NGA are preserved the diversity inside the population by altering the operators to prevent premature convergence to an optimum result, like fitness sharing [10], crowding [11] and deterministic crowding [12] model.

For example, sharing fitness encourages individuals to populate proportionally over the whole search space by introducing a penalty on the fitness of each individuals based on its relative distance to its neighbors. This causes population diversity pressure that allows a population to maintain individuals at local optima, and reduce premature convergence [14]. This strategy will also force the final distribution of individuals to be dispersed throughout the niche. Each individual is under pressure to maximize distance between itself and its neighbors. This diversity pressure within the niches retards the exploration of the fitness peak areas in each niche, as fewer individuals are able to populate and explore the fitness peak areas.

2.2 Parallel Model

There are two parallel models in evolution algorithm: the coarse-grain model and the fine-grain model. In the coarse-grain model, the parallel program, which consists of a few computing-intensive processes, has few communication demands, such as the Message Passing Interface (MPI) model. The fine-grain one is made up of a large number of processes with low computational requirements but high demands on the communication in order to coordinate all the processes. The former utilizes fewer processors with less communication than the latter.

In the GEP algorithm, the individuals must be exchanged from each population, and the population in different processed must be cooperated with others. It is obvious that the fine-grain model is appropriate for applications if considering the balance of computational speed and precision.

This fine-grain model in GEP algorithm is also called the cooperation model. The processes sometimes exchange information by allowing some individuals to migrate from one process to another according for optimization. A share individuals' pool will be set. This approach re-injects diversity into converging processes. Then, different processes will be tended to explore different parts of the search space. This parallel model is in figure 1.

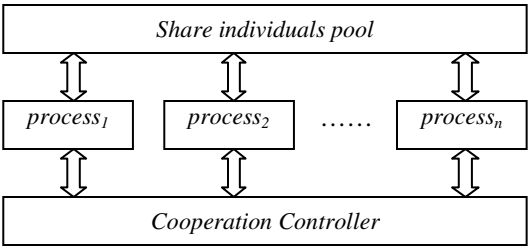

Fig. 1. Population is divided into several processes; the best individuals of each process will be exchanged through the share individuals' pool during the calculations. The cooperation controller controls the evolution of generation in each process.

3 Niching GEP Based on Parallel Model

In this paper, a hybrid algorithm called PNGEP is presented. This algorithm uses the fine-grain parallel model, which combines the niche theory and genetic mechanism.

3.1 Niching Method

The fundamental step of niching method is like the basic GEP. There is some different when the fitness of each individual is evaluated, a clustering of individuals operation will be done first. Before doing genetic operation, the individuals will be divided into k niches using the k-means clustering algorithm according to their fitness and *NMSE* value. The genetic operation will be done only in the same niche.

The main idea in this method is to define k centroids, one for each cluster. Each point is belonging to a given data set and associates it to the nearest centroid. Then recalculate k new centroids as new centers of the clusters resulting from the previous step, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. Finally k niches are set in a population with different types, such as good, average, poor, etc. Individuals only compete in the same niche and breed like in any traditional algorithm.

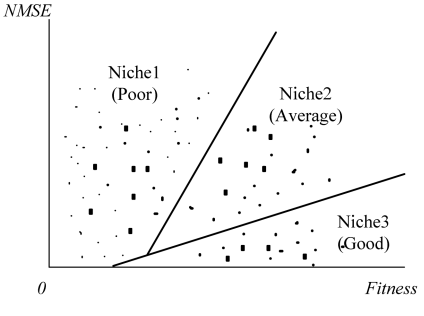

Fig. 2. First k centroids are defined; the individuals are selected and taken to the nearest centroid. Then re-calculate k new centroids of the clusters results and assign the individuals to the nearest new centroid. A loop has been generated. As a result of this loop the niche sets are initialization, the niches are marked like figure 2.

After doing the genetic operation in each niche, the *k* niches will compound to a new population and the elitism method will be used. This is one generation's operation, a loop will be generated. This clustering niching operation with *k*-means algorithm is shown in figure 2.

3.2 Parallel Model in Niching GEP

The main idea in this parallel algorithm is to define *N* sub-populations (processes), each sub-population with *k*-niche is mapped into a processor and its individuals are sometimes exchanged between the sub-populations during the calculations. The topology of this parallel model is shown in figure 3.

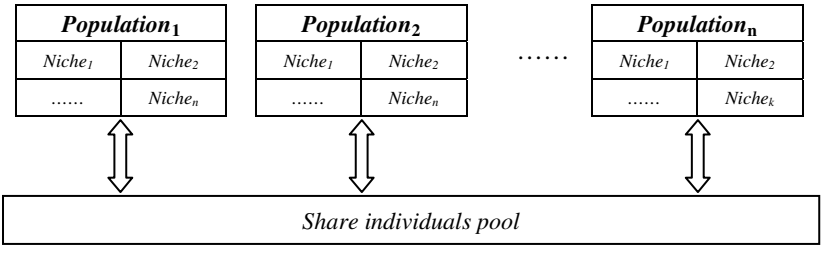

Fig. 3. Populations are divided into *N* sub-population and a sub-population is mapped into *k* niches. The best individuals of each sub-population will be exchanged through the share individuals' pool during the calculations.

In this parallel model, the best individual will be put into the share individual's pool and exchange to each sub-population. Then each sub-population will be re-injected the best genes. This behaves will be converged to a global/local optimum result.

3.3 Niching GEP Based on Parallel Model

PNGEP has seven genetic operators: mutation, transposition (insertion sequence transposition, root transposition and gene transposition), recombination (one-point, two-point and gene recombination). Among these operators, mutation is the most important and powerful one. PNGEP algorithm is depicted as follows:

```
Algorithm: PNGEP (Ts, Fs, f, P, Ps, k, N, G)

Input: Ts: the terminal set; Fs: the function set; f:
 the fitness function to evaluate the individuals; P: the
 sub-population for evaluation; Ps: the parameter for the
 genetic operation, such as the mutation rate, the multi-
 ple-point crossover rate, etc; k: number of the niches;
 N: number of the sub-populations; G: number of the gen-
 erations.

Output: The model with the highest fitness.
  1. For each sub-population:
       Initialize the sub-population P_i(i=1 to N) ran-
domly;
```

```
  2. For each generation g (g=1 to G)
      Evolution in each sub-population P_i (i=1 to N):
       (1)Inject: inject share-pool-individuals into P_i
random by pool exchange rate;
       (2)Evaluate: for each individual p, compute f(p);
       (3)Divide the individuals into k niches:
       (4)For each niche, generate the new population:

            (a) Mutation: generate new individual by mu-
            tation old individual.
            (b) Transposition: generate new individual by
            transposition old individual.
            (c) Recombination: generate new individual by
            recombination the two old individuals.

       (5) Using the elitism method;
       (6) Put the best m individual into share pool.

3. Return the best model with highest fitness.
```

4 Experiment and Results

In this paper, we compare PNGEP with the basic GEP in three problems [9, 15]. The first one is a problem of sequence induction, where a_n consists of the nonnegative integers. The n[th] term N of the chosen sequence is given by the formula:

$$N = 5a_n^4 + 4a_n^3 + 3a_n^2 + 2a_n + 1 \tag{1}$$

The second is a problem of "V" shaped function requiring floating-point constants. In this case, the following "V" shaped function is chosen:

$$y = 4.251a^2 + \ln(a^2) + 7.243e^a \tag{2}$$

where a is the independent variable and e is the irrational number 2.71828183.

Table 1. Wolfer sunspots series (read by rows)

101	82	66	35	31	7	20	92	154	125
85	68	38	23	10	24	83	132	131	118
90	67	60	47	41	21	16	6	4	7
14	34	45	43	48	42	28	10	8	2
0	1	5	12	14	35	46	41	30	24
16	7	4	2	8	17	36	50	62	67
71	48	28	8	13	57	122	138	103	86
63	37	24	11	15	40	62	98	124	96
66	64	54	39	21	7	4	23	55	94
96	77	59	44	47	30	16	7	37	74

The third one is the predicting sunspots problem. In this case, 100 observations of the Wolfer sunspots series are used (Table 1) with an embedding dimension of 10 and a delay time of one.

4.1 Setting the System

The relative error (equation 3), the absolute error (equation 4) and the normalized mean square error (*NMSE*, equation 5) are used to test the evaluation model.

$$fitness = \sum_{j=1}^{n} (M - |(y_j - y'_j)/y_j| * 100) \tag{3}$$

$$fitness = \sum_{j=1}^{n} (M - |y_j - y'_j|) \tag{4}$$

$$NMSE = \frac{\sum_{j=1}^{n} (y_j - y'_j)^2}{\sum_{j=1}^{n} (y_j - \bar{y}_j)^2} \tag{5}$$

In the equations, M is the range of selection; y_j is the fact value; $\bar{y}_j$ is the average of all y_j; y'_j is the value return by GEP. The less *NMSE* shows the good result.

For the sequence induction problem, the first 10 positive integers a_n are used as fitness cases. The fitness function is based on the relative error with a selection range of 20%, the maximum fitness is 200.

For the "V" shaped function problem, a set of 20 random fitness cases chosen from the interval [-1, 1] is used. The fitness function is also based on the relative error but in this case a selection range of 100% is used, the maximum fitness is 2000.

For the sunspot prediction problem, an embedding dimension of 10 and a delay time of one are used with 90 fitness cases. In this case, the fitness function is based on the absolute error with the selection range is 1000% and the maximum fitness is 90,000.

Because of the constants have less effect on the expected evolution; there is no constant using in the PNGEP algorithm. Our experiments show that the evolutionary results without constants of the three problems are good.

The PNGEP algorithm is written in C# using the threading class. *N* threads are created when the algorithm is initialized. Then *N* sub-populations are initialization and each sub-population is mapped into a thread process. When a-generation-running is done, the sub-populations exchange their individuals with the share stack. The best individuals will be re-injects diversity into converging sub-populations. Then, different sub-populations will be tended to explore different parts of the search space in this thread synchronization process.

In this paper, Experiments are running on a Hewlet-Packard BL25 blade server with AMD Opteron 265 1.8G CPU, 2G memory, Windows 2003 operation SP1 system and Microsoft .NET Framework 2.0 platform.

Table 2. General settings used in the sequence induction (SI), the "V" function and the sunspot prediction (SS) problems

	SI_{GEP}	SI_{PNGEP}	V_{GEP}	V_{PNGEP}	SS_{GEP}	SS_{PNGEP}
Number of runs	*100*	*100*	*100*	*100*	*100*	*100*
Number of generations	*100*	*100*	*200*	*200*	*200*	*200*
Population size	*200*	*50*	*200*	*50*	*200*	*50*
Niche number	*1*	*4*	*1*	*4*	*1*	*5*
Sub-population number	---	*2*	---	*4*	---	*4*
Number of fitness cases	*10*	*10*	*20*	*20*	*50*	*50*
Function set	{+,-,*,/}		{+,-,*,/, $ln,e^x,log,10^x,sin,cos$}		{+,-,*,/}	
Terminal set	{*a*}	{*a*}	{*a*}	{*a*}	{*a-j*}	{*a-j*}
Head length	*6*	*6*	*6*	*6*	*8*	*8*
Number of genes	*7*	*7*	*5*	*5*	*3*	*3*
Linking function	+	+	+	+	+	+
Chromosome length	*140*	*140*	*100*	*100*	*78*	*78*
Mutation rate	*0.044*	*0.044*	*0.044*	*0.044*	*0.044*	*0.044*
One-point recombination rate	*0.3*	*0.3*	*0.3*	*0.3*	*0.3*	*0.3*
Two-point recombination rate	*0.3*	*0.3*	*0.3*	*0.3*	*0.3*	*0.3*
Multipoint recombination rate	*0.3*	*0.3*	*0.3*	*0.3*	*0.3*	*0.3*
Gene recombination rate	*0.1*	*0.1*	*0.1*	*0.1*	*0.1*	*0.1*
IS transposition rate	*0.1*	*0.1*	*0.1*	*0.1*	*0.1*	*0.1*
IS element length	*1,2,3*	*1,2,3*	*1,2,3*	*1,2,3*	*1,2,3*	*1,2,3*
RIS transposition rate	*0.1*	*0.1*	*0.1*	*0.1*	*0.1*	*0.1*
RIS element length	*1,2,3*	*1,2,3*	*1,2,3*	*1,2,3*	*1,2,3*	*1,2,3*
Selection range	*20%*	*20%*	*100%*	*100%*	*1000%*	*1000%*
Pool size	---	*4*	---	*8*	---	*8*
Pool exchange rate	---	*0.2*	---	*0.2*	---	*0.2*
Average best-of-run fitness	*151.674*	*184.370*	*1648.21*	*1780.04*	*88609.6*	*88643.1*
Average best-of-run *NMSE*	*0.0011*	*0.0005*	*0.0252*	*0.0132*	*0.3233*	*0.3108*
Average running time(second)	*<30*	*<30*	*572.52*	*106.34*	*215.46*	*84.73*
Success rate	*40%*	*71%*	---	---	---	---

4.2 Experimental Analysis

In the experiments, the selection is made by roulette-wheel sampling coupled with simple elitism and the performance is evaluated over 100 independent runs. The six experiments are summarized in Table 2.

The first problem of sequence induction can be exactly solved by the basic GEP and the PNGEP. The success rate of the basic GEP is 40% and the PNGEP is 71%. Both algorithms' running time is less than 30s. The PNGEP' precision is higher than the basic GEP.

To find the "V" shaped function, we use function set F = {+, -, *, /, *ln*, e^x, *log*, 10^x, *sin*, *cos*}. The basic GEP's average best fitness is 1648.21, the average best *NMSE* is 0.0252 and the running time is 572.52s. The PNGEP's average best fitness is 1780.04, the average best *NMSE* is 0.0132 and the running time is 106.34s. The basic GEP's running time is about five times longer than the PNGEP.

For the sunspot prediction problem, the basic GEP's average best fitness is 88609.6, the average best *NMSE* is 0.3233 and running time is 215.46s. The PNGEP's average best fitness is 88643.1, the average best *NMSE* is 0.3108 and the running time is 84.73s. The basic GEP's running time is about four times longer than the PNGEP. From the comparisons in table 2, we can see that the PNGEP is taken less computation time but with higher accuracy than the basic GEP.

The basic GEP often enters a local optimization and jumps out of the local optimization at random probability. On the other hand, PNGEP can jump out of the local optimization at a greater probability with *N* sub-population. For the sunspot prediction problem, the basic GEP search the solution space only with one population, but the PNGEP using the 4 sub-population. Although the basic GEP individuals' number is for times than the PNGEP, the PNGEP's individuals have more diversity by using the niche method. The comparison in figure 4 shows that PNGEP has better search ability than the basic GEP.

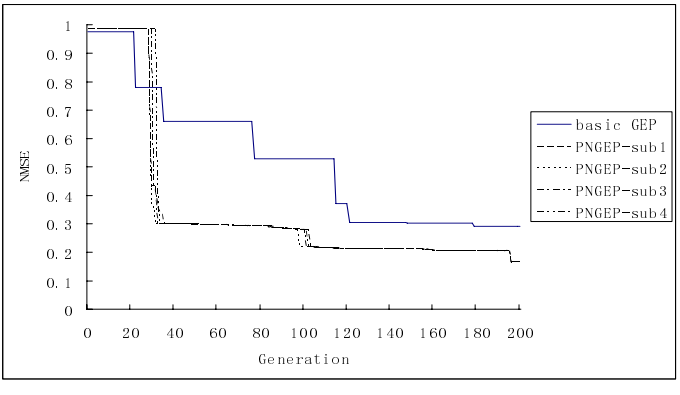

Fig. 4. The best solution's evolution in sunspot prediction problem between the basic GEP algorithm and the *4* sub-population PNGEP algorithm

Niching method tries to keep diversity in the population and to use this diversity as resource for exploratory evolution. The niche method of parallel model makes GEP with more flexibility and power of exploring the search space and converging to optimal result. From the comparisons of the success rate, the fitness value and the *NMSE* value, we can know that the PNGEP algorithm is better than the basic GEP.

5 Conclusion

In GEP algorithm, programs are represented as linear character strings of fixed-length which can be expressed as expression trees of different sizes and shapes. This separation of genotype and phenotype has endowed GEP with more flexibility and power of exploring the entire search space.

In this paper, a niching GEP based on parallel model is described and the advantages are demonstrated by its application. Experimental results on the sequence induction, the "V" shaped function and the sunspot prediction problem show that this parallel model of niching GEP algorithm, which called PNGEP, not only gains in the optimal results but also in better performance. It has higher precision and better search ability than the basic GEP. In the future, we will use the MPI parallel model and other clustering algorithm to improve the performance of this algorithm.

Acknowledgments

This research has been funded by the National Natural Science Foundation of Guangdong Province (07006474), Sci & Tech Research Project of Guangdong Province (2007B010200044) and Sci & Tech Research Project of Guangzhou (2006Z3-D3051).

References

1. Ferreira, C.: Gene Expression Programming: a New Adaptive Algorithm for Solving Problems. Complex Systems 13, 87–129 (2001)
2. Ferreira, C.: Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence. Angra do Heroismo Portugal (2002)
3. Ferreira, C.: Automatically Defined Functions in Gene Expression Programming. Studies in Computational Intelligence 13, 21–56 (2006)
4. Jie, Z., Changjie, T., Chuan, L.: Time Series Prediction Based on Gene Expression Programming. In: Proceedings of the Fifth International Conference on Web-Age Information Management, Dalian, China (2004)
5. Gang, P., Iimura, I., Nakatsuru, T.: Efficiency of Local Genetic Algorithm in Parallel Processing. In: PDCAT 2005. Parallel and Distributed Computing, Applications and Technologies, pp. 620–623 (2005)
6. Goldberg, D.: Sizing population for serial and parallel genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, California, pp. 70-79 (1989)
7. Andre, D., Koza, J.R.: Parallel genetic programming: A scalable implementation using the transporter network architecture. In: Angeline, P., Kinnear, K. (eds.) Advances in Genetic Programming 2, Cambridge, MA, pp. 317–337 (1993)
8. Oussaidkne, M., Chopard, B., Pictet, O.: Parallel genetic programming and its application to trading model induction. Parallel Computing 23, 1183–1198 (1997)
9. Siwei, J., Zhihua, C., Dang, Z.: Parallel Gene Expression Programming Algorithm Based on Simulated Annealing Method. ACTA Electronic Sinica 33, 2017–2021 (2005)

10. Goldberg, D., Richardson, J.: Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of the 2nd International Conference on Genetic Algorithms, pp. 41–49 (1987)
11. De Jong, K.: An analysis of the behavior of a class of genetic algorithms. Dissertation Abstracts International 36(10), 5140B (1975)
12. Mahfoud, S.W.: Crowding and preselection revisited. Parallel Problem Solving from Nature II, 27–36 (1992)
13. Ferreira, C.: Gene Expression Programming and the Evolution of Computer Programs. In: Recent Developments in Biologically Inspired Computing, pp. 82–103. Idea Group Publishing (2004)
14. Yang, H., Ch, F., Li, C., Wang, M.: A density clustering based niching genetic algorithm for multimodal optimization. In: Machine Learning and Cybernetics. Proceedings of 2005 International Conference, vol. 3, pp. 1599–1604 (2005)
15. Ferreira, C.: Function Finding and the Creation of Numerical Constants in Gene Expression Programming. In: Proceedings of the 7th Online World Conference on Soft Computing in Industrial Applications (2002)