

An Experimental Study of Some Control Parameters in Parallel Genetic Programming

Hongqing Cao^{1,*}, Jingxian Yu², Lishan Kang¹, R. I. Bob McKay³

¹State Key Laboratory of Software Engineering, Wuhan University,
Wuhan 430072, P. R. China

²Department of Chemistry, Wuhan University, Wuhan 430072, P. R. China

³School of Computer Science, University of New South Wales at ADFA, Northcott Drive,
Canberra, ACT 2600, Australia

Abstract Using the evolutionary modeling of system of ordinary differential equations (ODEs) as the test problem, this paper primarily investigates the influences of some important parallel control parameters within parallel genetic programming (GP), including the degree of connectivity between demes, the migration rate, the migration generation interval, and the migration policy, on the performance of the parallel evolutionary modeling algorithm (PEMA), which is measured from two perspectives: the solution quality and the parallel speedup. We compare the results with previous theoretical and experimental work in parallel genetic algorithms (GAs), and try to give some plausible analysis and explanations. The results may help to offer some useful design guidelines for researchers using parallel GP.

Keywords parallel genetic programming, parallel control parameters, evolutionary modeling, system of ordinary differential equations

1. INTRODUCTION

In recent years, the study of evolutionary algorithms (EAs) has gained much interests of many researchers in wide fields. EAs are adaptive methods for solving computational problems in many fields, which mimic the process of biological evolution and the mechanisms of natural selection and genetic variation. They use suitable codings to represent possible solutions to a problem, and guide the search by using some genetic operators and the principle of “survival of the fittest”. Due to their merits of self-adaptation, self-organization, self-learning, intrinsic parallelism and generality, EAs have succeeded in solving a large number of problems in machine learning, pattern recognition, economic prediction, optimization control, parallel processing and many other domains (Goldberg, 1989; Bäck et al., 1997).

Amenability to parallelization is an appealing feature of EAs. Bethke (Bethke, 1976) was the first to describe parallel implementations of a conventional genetic algorithm (GA) (Holland, 1975). Subsequently, a wide range of parallel GAs have been investigated, which can

be divided into four classes based on the connection topology: global single-population master-slave GAs, single-population fine-grained GAs, multiple-population coarse-grained GAs and hierarchical parallel GAs (Cantú-Paz, 2000). Of these, coarse-grained parallel GAs (also known as distributed GAs or “island” parallel GAs) have been the most popular parallel method (Alba & Troya, 1999), due to their simplicity in principle and the low-overhead experimental environment (which may readily be implemented with a network of workstations, or even a single-processor machine, using free software such as PVM or MPI). The most important characteristics of this class of parallel GAs are that they consist of several subpopulations (also called demes) that occasionally exchange some individuals in a process called migration. Many papers have been written describing innumerable aspects and details of their implementation (Belding, 1995; Lin et al., 1994; Tanese, 1989).

A specification of a parallel GA with multiple demes defines the size and number of the demes, the topology of the connections between them, the migration rate (the fraction of the subpopulation that migrates), the migration generation interval (the number of generations in every subpopulation between two successive exchanges, which affects the frequency of migrations), and the policy to select emigrants and to replace existing individuals with incoming migrants. The importance of these parallel control parameters on the quality of the search and on the efficiency of the algorithms has been recognized for a long time (Grefenstette, 1981; Grosso, 1985; Tanese, 1987). In this aspect, Cantú-Paz (Cantú-Paz, 1999; Cantú-Paz, 2001) has made some extensive studies in coarse-grained parallel GAs. He obtained some instructive theoretical and experimental results, on some relatively simple problems such as the 100-bit and 500-bit One-Max problems and 4-bit and 8-bit deceptive trap functions, using a parallel environment consisting of eight IBM RS 6000 workstations connected by an Ethernet.

The emergence of genetic programming (GP) (Koza, 1992; Koza, 1994) has given rise to further studies on parallel GP. Andre and Koza (Andre & Koza, 1996) reported the first parallel GP implementation on a PC 486 type computer and a network of 64 node transputers using a kind of “fine-grain” parallelism. They compared the computational effort required to solve the problem of symbolic regression of the Boolean even-5-parity function with different migration rates and achieved super-linear speedups. Nordin et al. (Nordin et al., 1999) implemented the parallization of an AIM-GP system by using the Parsytec Power Explorer with up to 64 Power PC processors. The system was tested on three problems, namely a Boolean problem, a function regression problem and an image classification problem. They also conducted some studies on migration frequency and migration strategies. Punch et al. (Punch et al., 1996) developed the royal tree problem as a benchmark problem for GP. The results they reported were somewhat surprising, as it appears that a single large population outperformed a group of smaller populations under all parallel conditions when solving the royal tree problem. Recently Fernández et al. (Fernández et al., 2000) have worked with a coarse-grained parallel GP using a client/server structure and a dynamic topology between processors to develop a medical decision classifier system capable of automatically diagnosing the degree of a burn. Again, despite an increasing breadth of studies on parallel GP, they mostly focus on applications of

parallel GP in particular problem domains and seldom involve an extensive study of control parameters within parallel GP. However, the different settings of those parameters can greatly affect the efficiency and accuracy of parallel GP.

In view of the above, the main objective of this paper is to present an exhaustive experimental study of some important parallel control parameters in parallel GP. We choose the evolutionary modeling of system of ordinary differential equations (ODEs) as the test problem and design the corresponding parallel evolutionary modeling algorithm (PEMA). The experimental environment consists of 128 Pentium III 500 type computers connected by an Ethernet. In this environment, a series of experiments are conducted to systematically test the influence of some important parallel parameters on the performance of the algorithm. Specifically, the four parallel parameters we examine are the degree of connectivity between demes, the migration rate, the migration generation interval and the migration policy. We measure the performance of the algorithm primarily from two perspectives: the solution quality and the parallel speedup. We make some comparisons with previous theoretical and experimental work.

The remainder of this paper is organized as follows. The next section gives a mathematical description of the test problem and the main procedure of the PEMA. Section 3 is the focus of this paper, documenting the parallel experiments in detail. A series of experiments have been conducted to test the influence of four important parallel parameters on the performance of PEMA. Based on the experimental results, we conduct some analysis and construct some explanations of the results. Finally, Section 4 summarizes the results of this study and outlines our future work.

2. THE EVOLUTIONARY MODELING OF SYSTEMS OF ODES

2.1. Problem Formulation

In deriving the model approach, we first define the norm of a matrix A where $A \in \mathbb{R}^{m \times n}$ as

$$\|A\| = \sqrt{\sum_{i=1}^m \sum_{j=1}^n (a_{ij})^2} \quad (1)$$

Suppose a dynamic system can be described by n correlated functions, $x_1(t), x_2(t), \dots, x_n(t)$, and a series of observed data collected at the time $t_i = t_0 + i \cdot \Delta t$ ($i = 0, 1, 2, \dots, m-1$). This can be written in the following form

$$X = \begin{pmatrix} x_1(t_0), & x_2(t_0), & \cdots, & x_n(t_0) \\ x_1(t_1), & x_2(t_1), & \cdots, & x_n(t_1) \\ \vdots & \vdots & \vdots & \vdots \\ x_1(t_{m-1}), & x_2(t_{m-1}), & \cdots, & x_n(t_{m-1}) \end{pmatrix} \quad (2)$$

where t_0 denotes the starting time, Δt denotes the interval between two observations, $x_j(t_i)$ ($j = 1, 2, \dots, n$) denotes the observed value of variable x_j at the time t_i .

Let us denote $X(t) = [x_1(t), x_2(t), \dots, x_n(t)]$, $f(t, X) = [f_1(t, X), f_2(t, X), \dots, f_n(t, X)]$ where $f_j(t,$

$X) = f_j(t, x_1(t), x_2(t), \dots, x_n(t))$ ($j = 1, 2, \dots, n$) is a composite function composed of elementary functions involving variables x_i ($i = 1, 2, \dots, n$) and t . The function space defined by those functions can be denoted as F . Then the modeling problem of a system of ODEs is to find a model, a system of first-order differential equations having the general form

$$dX^*/dt = f(t, X^*) \quad (3)$$

such that

$$\min\{\|X^* - X\|, \forall f \in F\}$$

where

$$\|X^* - X\| = \sqrt{\sum_{i=0}^{m-1} \sum_{j=1}^n [x_j^*(t_i) - x_j(t_i)]^2} \quad (4)$$

($x_j(t_i)$ and $x_j^*(t_i)$) are the simulated value and the fitting value of x_j at the time t_i respectively.)

and the values of x_i at the next time steps

$$\begin{pmatrix} x_1(t_m), & x_2(t_m), & \cdots, & x_n(t_m) \\ x_1(t_{m+1}), & x_2(t_{m+1}), & \cdots, & x_n(t_{m+1}) \\ \vdots & \vdots & \vdots & \vdots \\ x_1(t_{m+\tau-1}), & x_2(t_{m+\tau-1}), & \cdots, & x_n(t_{m+\tau-1}) \end{pmatrix} \quad (5)$$

can be predicted based on the model.

2.2. Parallel Evolutionary Modeling Algorithm (PEMA)

The parallel evolutionary modeling algorithm we design consists of four major steps:

Step1. Firstly a master process reads the modeling samples, some control parameters of EM (n , $Maxgeno$, D etc.) and some parallel control parameters (P , δ , g , ρ etc.). Then it creates P worker processes and broadcasts the above data to them.

The meanings of the above parameter are as follows:

- n is the number of individuals in the whole population.
- $Maxgeno$ is the maximum number of generations undergone by the master process.
- D is the maximum tree depth of GP, bounding the length of a model expression.
- P is the number of worker processes. Thus the total number of demes is $P+1$.
- δ is the degree of connectivity between demes, i.e. the number of neighbors of each deme.
- g is the migration generation interval, i.e. the migration of individuals between demes occurs every g generations.
- ρ is the migration rate, i.e. the percentage of individuals in a deme undergoing migration.

Step2. The master and the P workers initialize their own subpopulations and perform the GP EM procedure separately. At the same time, each process is allowed to exchange messages with other processes at intervals. That is:

- (1) Every g generations, each process selects copies of m individuals in the subpopulation and sends them to the δ neighboring processes. The neighbors of

a process are determined by a particular topology predefined. The value of m is determined by ρ and the size of each deme, n_d , where $n_d = n / (P + 1)$. Namely $m = \rho \times n_d$. The selection of the emigrated individuals is decided by the specified migration policy, with standard policies being selecting the best, and random selection.

(2) At each generation, each process sets breakpoints to detect whether individuals have arrived from other processes. If so, receive the individuals and use them to replace the same number of individuals in the deme. The selection of the replaced individuals is also decided by the specified migration policy, with standard policies being replacing the worst, and random replacement.

Step3. The master sends a termination message to P workers when it reaches *Maxgeno* generations. The workers terminate immediately on receipt of the message.

Step4. The master selects the best individual in its deme as the final model, and makes further predictions based on the model.

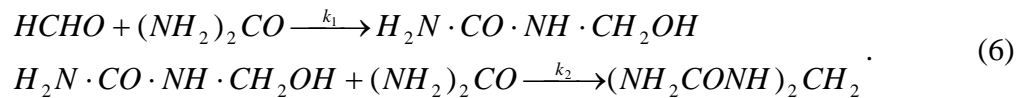
For more detail on the use of GP techniques to discover and optimize the model structure, the reader is referred to (Cao et al., 2000).

3. PARALLEL MODELING EXPERIMENTS

3.1. Modeling Example

We take a typical system of chemical consecutive reactions as the modeling example.

The reaction between formaldehyde (X_1) and carbamide in aqueous solution gives methylol urea (X_2), which continues to react with carbamide and form methylene urea (X_3). The reaction equations are



The reactions occur at 308.15K with an excess of carbamide above the concentration ($c_{2(0)}$) of $2\text{mol}\cdot\text{dm}^{-3}$. The concentration of hydrochloric acid as a catalyst is $0.0008\text{mol}\cdot\text{dm}^{-3}$ and the initial concentration of formaldehyde ($X_{1(0)}$) is $0.1\text{mol}\cdot\text{dm}^{-3}$. As a typical consecutive reaction, the concentrations of the three components in the system satisfy the following system of ODEs

$$\begin{cases} dX_1 / dt = -k_1' X_1 \\ dX_2 / dt = k_1' X_1 - k_2' X_2 \\ dX_3 / dt = k_2' X_2 \end{cases} \quad (7)$$

where $k_1' = k_1 c_{2(0)}$, $k_2' = k_2 c_{2(0)}$ with $k_1 = 0.007\text{dm}^3\cdot\text{mol}^{-1}\cdot\text{min}^{-1}$, $k_2 = 0.021\text{dm}^3\cdot\text{mol}^{-1}\cdot\text{min}^{-1}$.

According to the exact solution of the consecutive reaction

$$\begin{cases} X_1 = X_{1(0)} e^{-k_1 t} \\ X_2 = \frac{k_1 X_{1(0)}}{k_2 - k_1} (e^{-k_1 t} - e^{-k_2 t}) \\ X_3 = X_{1(0)} - X_1 - X_2 \end{cases} \quad (8)$$

we calculate the concentrations of X_1 , X_2 , X_3 every two minutes for 110 minutes after the reactions occur, and take them as the simulated data of our experiments. The first 50 points are used as modeling samples and the last 5 points are used as test samples to evaluate the predictive accuracy of the model.

3.2. Experimental Settings

All the experiments are performed on a simulated parallel environment consisting of 128 Pentium III 500 computers connected by a 10 Mbps Ethernet. We use PVM 3.3 as the parallel programming tool to communicate between computers. Most of the parameter settings of PEMA are listed in Table 1. The settings of the other parallel parameters vary with different experimental purposes. See Section 3.3. for more details.

Table 1. Parameter settings of the parallel evolutionary modeling algorithm

GP EM	Function set $F = \{+, -, *, /, \wedge, \sin, \cos, \exp, \ln\}$ where x^n represents x^n ($0 < n < 5$) Terminal set $T = \{x_1, x_2, x_3, t, c\}$ where c is a random constant $D = 3$ $Maxgen = 100$
Parallel Parameters	$P = 127$ $n = 6400$ $n_d = 50$

The performance of the parallel algorithm is measured mainly from two perspectives, namely the quality of the final solution and the parallel speedup. We judge a solution better or worse by calculating the fitting error (FE) (i.e. the fitness value) of the model as

$$FE = \sqrt{\sum_{i=1}^{50} \sum_{j=1}^3 [x_j^*(t_i) - x_j(t_i)]^2} \quad (9)$$

Clearly, the smaller FE, the better the model. In addition, we calculate the predicting error (PE) as

$$PE = \sqrt{\sum_{i=51}^{55} \sum_{j=1}^3 [x_j^*(t_i) - x_j(t_i)]^2} \quad (10)$$

to check the prediction results of the final model.

How to measure the parallel speedup is always controversial (Alba & Tomassini, 2002). As is traditional, we take the ratio of the serial execution time and the parallel execution time when the same number of generations predefined is reached as the speedup, regardless of the quality of the solutions reached by each algorithm. That is,

$$Sp = T_s / T_p. \quad (11)$$

Ten independent trials are carried out for each experiment. We calculate the average fitting error (AFE) of ten runs as a measure of solution quality, and measure the parallel speedup by calculating the average running time of the serial algorithm and the parallel algorithm respectively.

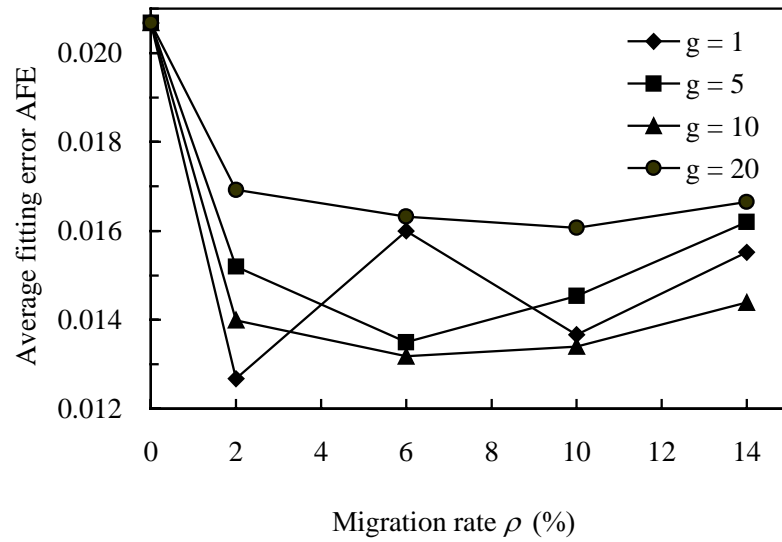
3.3. Experimental Results

3.3.1 The influence of three parallel parameters on the solution quality

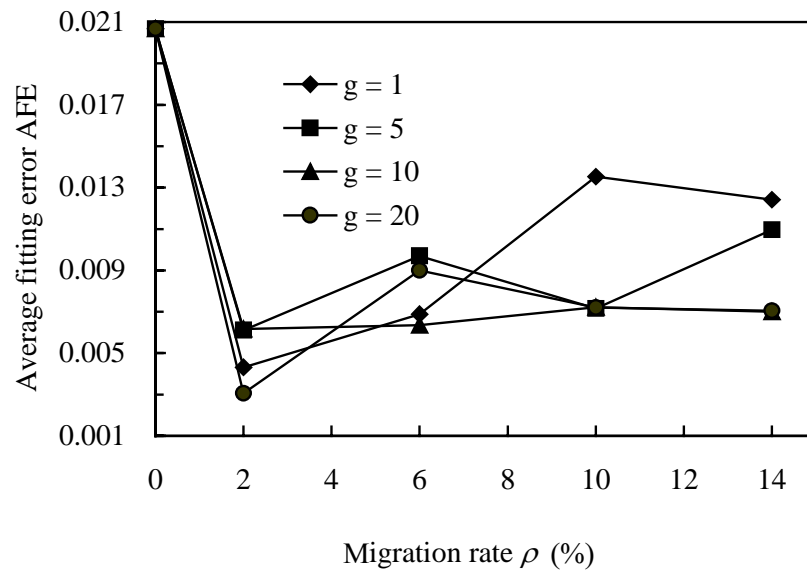
The goal of the experiments in this part is to examine the influence of three important parallel control parameters, namely the degree of connectivity between demes, δ , the migration generation interval, g , and the migration rate, ρ , on the quality of the final solution, and also to note any interaction effects.

Starting from Cantú-Paz' experimental verification (Cantú-Paz, 1999) that different topologies with the same degree reach almost identical solutions after any number of epochs, we only vary the degree of the topology, δ , in our parallel experiments, ignoring the possible specific topologies of communication. In other words, for a given degree δ , we use the simplest topology as follows. Let i denotes the serial number of a deme, its δ neighboring demes are designated as $\{(i + 1), (i + 2), \dots, (i + \delta)\} \bmod m_d$ sequentially where m_d is the total number of demes (herein $m_d = 128$). Without loss of generality, we take into account two extreme cases of $\delta = 1$ (singly-connected) and $\delta = 127$ (fully-connected), and an intermediate case of $\delta = 64$ (half-connected). In each case, a series of experiments are conducted by combining a variety of migration generation intervals with $g = 1, 5, 10, 20$ and different migration rates with $\rho = 0\%, 2\%, 6\%, 10\%, 14\%$ (i.e. $m = 0, 1, 3, 5, 7$). The migration policy we use is the most common one, just selecting the best individuals to migrate and replacing the worst ones in the receiving deme.

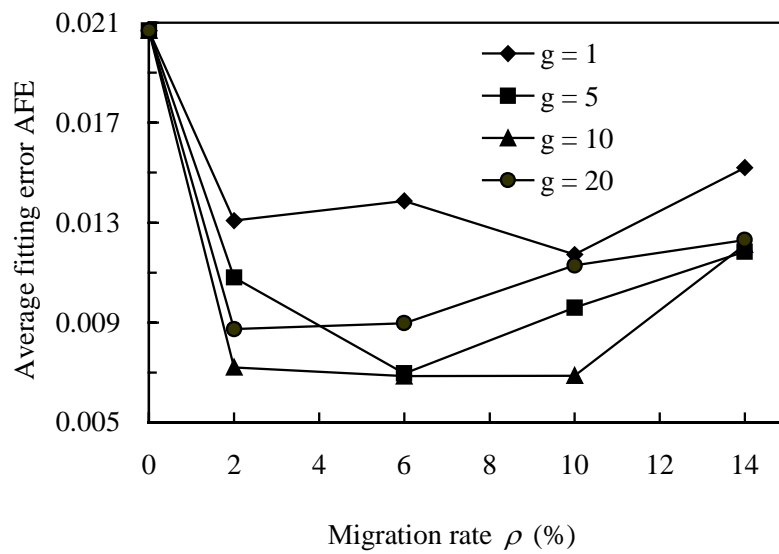
The AFEs in the three levels of connectivity, varying the migration interval and the migration rate, are illustrated in Figs. 1(a), (b) and (c) respectively.



(a)



(b)



(c)

Fig. 1. The average fitting errors in three cases of connectivity by varying the migration rate and the migration generation interval: (a) $\delta = 1$ (b) $\delta = 64$ (c) $\delta = 127$

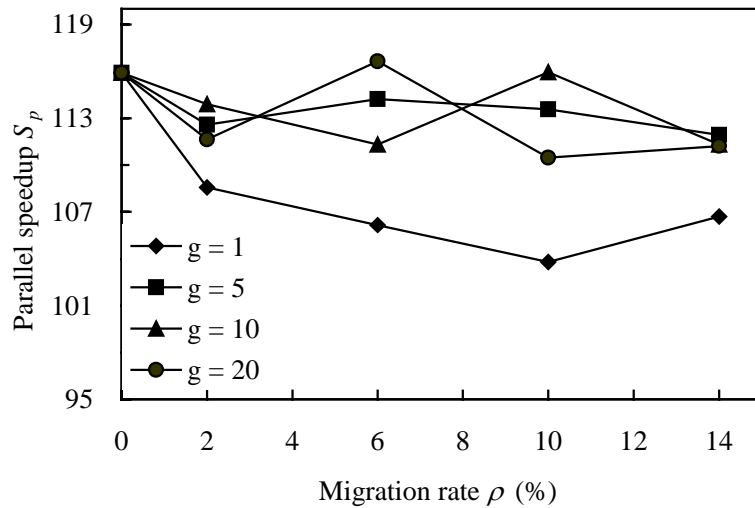
Fig. 1 has the following implications:

- (1) In all cases, the AFE is largest when no migration occurs between demes (i.e. $\rho = 0\%$). The AFE, 0.020678, is much larger than the averaged serial result over ten runs 0.016009. However once the demes begin to exchange individuals, the solution quality is improved significantly, no matter what values of g and ρ are assigned. This result is consistent with Grosso's observations (Grosso, 1985) that the quality of the solution found after convergence was worse when the demes were isolated than in the single population. However, with migration at intermediate migration rates, the final average quality increased significantly, and in some cases it was better than the final quality in a serial GA. This indicates that parallelization into multi-demes without migration is impractical as it will usually produce a poor solution. The results suggest that users should avoid the use of isolated demes.
- (2) Comparing the ranges of AFE over the three cases, we see that the degree of connectivity has a large effect on the solution's quality. When $\delta = 1$, the AFE is between 0.012680 and 0.016920; when $\delta = 64$, it is between 0.003051 and 0.013539; and when $\delta = 127$, it is between 0.006845 and 0.015196. The solution quality in the extreme cases is much worse than in the intermediate case. This suggests that a number of neighbors for exchanging individuals are necessary to guarantee a good solution, and an intermediate connectivity seems preferable than the two extremes of one neighbor or fully-connected.
- (3) Examining curves in Fig.1(a)(b)(c) in more detail, we see an intimate relationship exists between the three parallel parameters for not only balancing the communication cost but also guaranteeing a good solution.
 - (i) Noting that each curve oscillates, we can conclude that the solution quality is not a monotonic function of any parameter, especially the migration rate. It might be tempting to conclude that, since migration is desirable, simply increasing the number of neighbors, the migration rate, or the frequency of migration, would always be desirable. The results demonstrate that this conclusion would be erroneous. However, this result is in disagreement with Cantú-Paz's theoretical results (Cantú-Paz, 1998) that show that the solution's quality increases with higher migration rates.
 - (ii) Examining the worst solutions in the three cases concerned, when $\delta = 1$, we always obtain the worst solution at $g = 20$, and in most cases, we see poor solutions at $\rho = 2\%$ or 14% . Conversely, when $\delta = 127$, the worst solution is mostly obtained at $g = 1$ and $\rho = 14\%$, while when $\delta = 64$, the results are mixed. This suggests that when the demes are sparsely connected, it is undesirable to use an infrequent migration such as $g = 20$, or to migrate too few individuals each time. Conversely, when the demes are densely connected, over-frequent migration such as $g = 1$, or migrating large numbers individuals each time, should be avoided. Hence a kind of balance should be reached

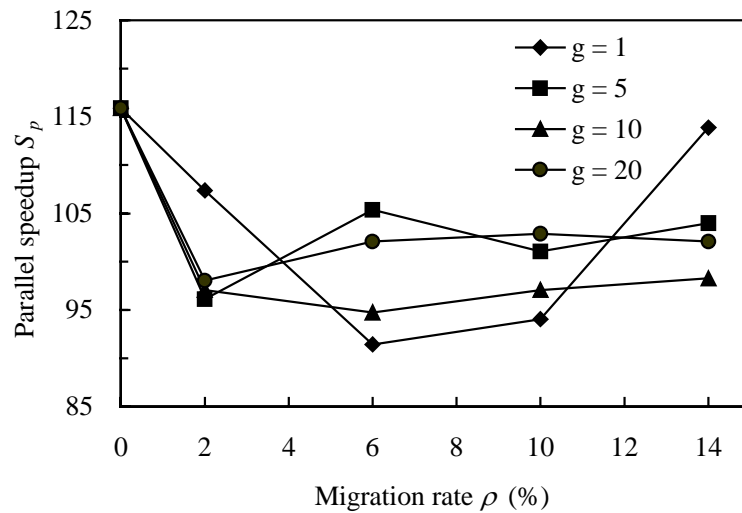
among the three parallel parameters so as to guarantee an adequate communication level to derive a good solution. Many studies have been conducted about how to set their optimal values, but no clear conclusions can be drawn. Based on our experimental results, a good solution is most likely when the demes are half-connected, and g and ρ are set to moderate values such as $g = 10$ and $\rho = 6\%$.

3.3.2 The influence of three parallel parameters on the parallel speedup

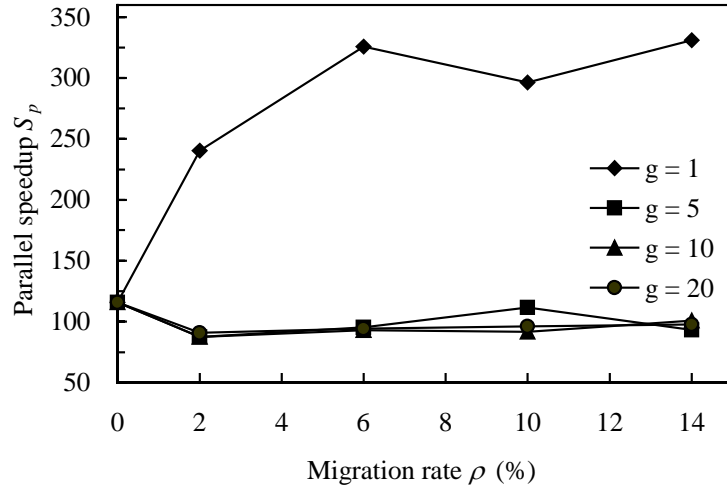
We depict the speedups in the three connectivity cases, varying the migration rate and the migration generation interval, in Figs. 2(a), (b) and (c).



(a)



(b)



(c)

Fig. 2. The parallel speedups in three cases of connectivity by varying the migration rate and the migration generation interval: (a) $\delta = 1$ (b) $\delta = 64$ (c) $\delta = 127$

It can be observed that the speedup curves reveal distinct characteristics for the three cases of connectivity.

- (i) In Fig. 2(a), when the demes are single-connected, under a fixed ρ , we always see the lowest speedup when the migration occurs every generation. This makes sense, because very frequent migration implies increased communication load, inevitably leading to increased parallel execution time. Beyond that, changing the values of ρ and g does not significantly impact the speedup.
- (ii) As shown in Fig. 2(b), when the demes are half-connected, the speedup changes dramatically with the increase of ρ at $g = 1$, but does not change so much at other g values. Moreover at $g = 1$, the speedup initially decreases with ρ , but ascends once the rate is greater than 6%. We think this arises from two competing effects of the increased rate. As discussed above, with more migrants between demes, there is an inevitable increase in communication load. On the other hand, it also causes faster convergence of the algorithm. With the injection of many individuals from its neighbors, the deme will quickly converge into a subpopulation with poor diversity generation. When ρ increases to such a degree that its effect on convergence speed exceeds that on communication load, the speedup will increase.
- (iii) From Fig. 2(c), we see that when the demes are fully-connected, for $g = 1$ we get a superlinear speedup for all values of ρ . But this phenomenon does not occur at other g values.

Superlinear speedups have been frequently claimed, by many researchers in parallel EAs (Andre & Koza, 1996; Belding, 1995). In their experimental designs, researchers usually set important parameters (such as the total number of individuals, the maximum number of generations) to be identical in the serial and parallel algorithms,

ignoring the possible significant effects of some crucial parallel parameters on the actual execution time of the algorithm. In our results, in the case of maximal degree of connectivity, performing the most frequent migration (every generation) reliably speeds up the convergence of demes. Moreover, the higher the migration rate, the faster the convergence speed. When the execution time of the parallel algorithm is reduced dramatically, naturally we get a superlinear speedup. At other levels of g , less frequent migration does not give rise to such large reductions of execution time. Of course, it is worth noting that the reduction in execution time comes at the cost of a degraded solution quality. From Fig. 1(c), we have seen that the AFE is between 0.011718 and 0.015196 when $\delta = 127$, $g = 1$, which is not much improvement on the serial result of 0.016009.

According to the discussions in above (i), (ii) and (iii), we conclude that among the three parallel parameters, the degree of connectivity has the greatest impact on the speedup. Such effects are most obvious when $g = 1$. In particular, when the demes are fully-connected, superlinear speedup always occurs.

3.3.3 The influence of migration policy on the solution quality

As discussed in Section 3.3.1(1), the reason for the improved quality of parallel solutions lies in the migration of individuals between demes. There are two alternatives to select the individuals that emigrate from a deme. They can be chosen randomly, or by selecting the best individuals in a deme. Likewise, there are two choices to replace existing individuals in the receiving deme with the incoming migrants: choose randomly or replace the worst. The four possible combinations of migration selection and replacement form the following migration policies:

Policy 1: best \rightarrow worst	Policy 2: best \rightarrow random
Policy 3: random \rightarrow worst	Policy 4: random \rightarrow random

We choose three experimental cases with different parallel parameter configurations to examine the influence of migration policy on the performance of the algorithm:

Case I: $\delta = 127$, $g = 1$, $\rho = 2\%$

Case II: $\delta = 64$, $g = 5$, $\rho = 6\%$

Case III: $\delta = 127$, $g = 5$, $\rho = 10\%$.

For each case, we try the four migration policies, conducting 10 independent runs for each experiment. The average fitting errors in the three experimental cases, varying the migration policy, are shown in Fig. 3.

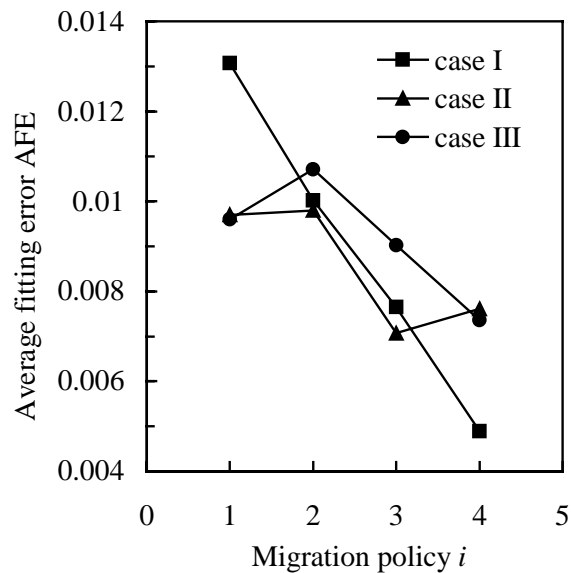


Fig. 3. The average fitting errors in three experimental cases by varying the migration policy

It can be seen from Fig. 3 that the choice of migration policy has a relatively large impact on the solution quality. Surprisingly, Policy 1, the policy most commonly used by researchers, is not the best, as its AFE is in most cases larger than that obtained by using any of the other three policies. Conversely, Policy 4, which selects migrants randomly and replaces random ones at the receiving deme, is more likely to obtain a better solution. We explain this as the effect of selection pressure. According to Cantú-Paz's calculations of selection intensity under different migration policies (Cantú-Paz, 2001), the migration policy with the highest intensity is when the best individuals migrate and replace the worst while the policy with the lowest intensity is when the random individuals replace the random ones. It has been well established that higher selection pressures causes faster convergence (Bäck, 1994). In other words, a weak selection pressure will slow down convergence, allowing the variation operators more time to create new solutions and thus more chance to eventually yield a good solution. Hence using Policy 1, the most eager policy, can easily cause the GP to converge prematurely to a poor solution. Conversely, using Policy 4, the weakest policy, can increase the deme's diversity and delay convergence, giving the algorithm more opportunity to find a good solution.

3.3.4 The influence of migration policy on the parallel speedup

We depict the parallel speedups in the three experimental cases by varying the migration policy in Fig. 4.

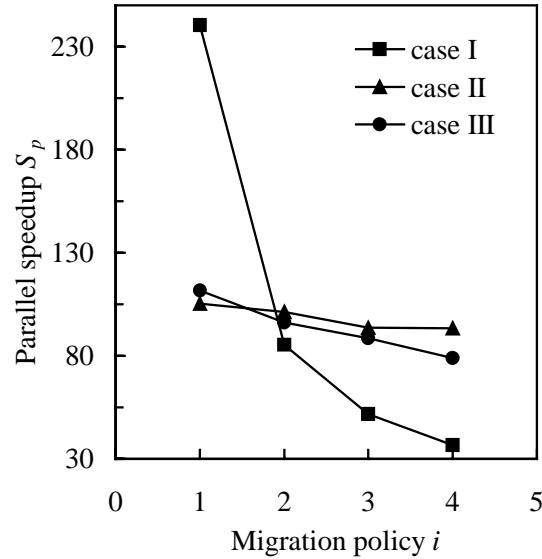


Fig. 4. The parallel speedups in three experimental cases by varying the migration policy

From the plots in Fig. 4, we observe that the speedup decreases as a migration policy with less selection pressure is used. Again, this is essentially the result of the different selection pressures of the four migration policies. Under Policy 1, the algorithm converges quickly, its execution time is shortest, so its speedup is highest. Under Policy 4, the algorithm converges slowly, delaying execution time and so achieving the lowest speedup. Regarding the superlinear speedup under Policy 1 in Case I, in addition to the possible reasons we have given in Section 3.3.2 (iii), we agree with Cantú-Paz's explanation (Cantú-Paz, 2001) that some claims of superlinear speedups are caused by an increase of selection pressure due to migration. That is, with Policy 1, the strictest migration policy in use, the algorithm converges fast and the execution time is reduced rapidly, resulting in superlinear speedup.

Summarizing Section 3.3.3 and Section 3.3.4, we can describe the influence of migration policy on the algorithm's performance as follows. It is surprising that the most commonly used policy, namely that the best migrants replace the worst individuals, is a poor choice in terms of solution quality. But combined with some other parallel parameters, it may bring about superlinear speedups. In contrast, a low pressure migration policy, namely that random migrants replace random individuals, is more likely to find a good solution, but it gains this by sacrificing execution time. Usually, with this policy, the algorithm runs slowly and the speedup is low.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we mainly conduct a very exhaustive experimental study on some crucial parallel parameters in parallel GP. Those parameters include the degree of connectivity between demes, the migration rate, the migration generation interval and the migration policy.

We take the evolutionary modeling of system of ordinary differential equations (ODEs) as the test problem and design the parallel evolutionary modeling algorithm (PEMA) to approach this. We investigate the influence of those parallel parameters on the performance of the algorithm mainly from two aspects: the solution quality and the parallel speedup. We compare our results with previous theoretical and experimental work and observe that some are in agreement, others in disagreement with previous studies. We attempt to give some plausible analysis and explanations of this. We summarize the conclusions of the study as follows. We hope they can offer some useful design guidelines for researchers using parallel GP to solve other GP problems.

- 1) To achieve a better solution quality, users should avoid the use of isolated demes, i.e. with no migration occurring between demes.
- 2) A reasonable number of neighbors for exchanging individuals are necessary to guarantee a good solution, and an intermediate connectivity seems preferable than the two extremes of one neighbor or fully-connected.
- 3) There is an intimate relationship between the three parallel parameters, namely the degree of connectivity between demes, the migration rate and the migration generation interval. To be specific, firstly the solution quality is not a monotonic function of any parameter, especially the migration rate. Secondly when the demes are sparsely connected, it is undesirable to use an infrequent migration such as $g = 20$, or to migrate too few individuals each time. Conversely, when the demes are densely connected, over-frequent migration such as $g = 1$, or migrating large numbers individuals each time, should be avoided. Thirdly how to set the optimal values of the three parameters is dependent on the specific problem. Based on our experimental results, a good solution is most likely when the demes are half-connected, and g and ρ are set to moderate values such as $g = 10$ and $\rho = 6\%$.
- 4) Among the three parallel parameters, the degree of connectivity has the greatest impact on the speedup. Such effects are most obvious when $g = 1$. In particular, when the demes are fully-connected, superlinear speedup always occurs. However it is worth noting that the reduction in execution time comes at the cost of a degraded solution quality.
- 5) It is surprising that the most commonly used policy, namely that the best migrants replace the worst individuals, is a poor choice in terms of solution quality. But combined with some other parallel parameters, it may bring about superlinear speedups.

In this study, we only choose the evolutionary modeling problem of ODEs as one test problem. To generalize the conclusions, we will use a set of problems that have been classically used for testing GP to examine the effects of those parallel parameters in the next step. In addition, there are also some other important parallel control parameters that should be considered, like the number of demes and the size of each deme whose values are fixed in this study. We will further investigate their effects on the performance of parallel algorithms by performing a large number of experiments.

ACKNOWLEDGEMENT

This work was supported by National Natural Science Foundation of China (No. 60133010, No. 70071042 and No. 60073043) and the Special Funds for Major State Basic Research Projects of China (No. G2000077307). The authors wish to thank Prof. Jacob Borgerson and Prof. David Goldberg for their kind provision of the valuable technical reports of the IlliGAL Laboratory.

REFERENCES

1. Alba, E., Troya J. M.(1999). A survey of parallel distributed genetic algorithms. *Complexity*, v.4, n.4., 31-52.
2. Alba, E., & Tomassini, M.(2002). Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, v.6, n.5, 443-462.
3. Andre, D., & Koza, J. R.(1996). Parallel genetic programming: A scalable implementation using the transputer network architecture. In *Advances in Genetic Programming 2*, P. Angeline, P., & Kinnear, K., Eds. Cambridge, MA: MIT Press, 317-337.
4. Bäck, T.(1994). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proc. First IEEE Conference on Evolutionary Computation*, Piscataway, NJ: IEEE Service Center, 57-62.
5. Bäck, T., Hammel, U., and Schwefel, H.-P.(1997). Evolutionary computation: comments on the history and current state. *IEEE Transaction on Evolutionary Computation*. v.1, n.1, 5-16.
6. Belding, T. C.(1995). The distributed genetic algorithm revisited. in *Proc. 6th International Conference on Genetic Algorithms*, Eshelman, L. J., Ed. San Francisco, CA: Morgan Kaufmann, 114-121.
7. Bethke, A. D.(1976). Comparison of genetic algorithms and gradient-based optimizers on parallel processors: efficiency of use of processing capacity. Tech. Rep. 197, University of Michigan, Logic of Computers Group, Ann Arbor, MI, 1976.
8. Cantú-Paz, E.(1998). Using markov chains to analyze a bounding case of parallel genetic algorithms. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, J. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. Fogel, M. Garzon, D. E. Goldberg, H. Iba and R. Riolo, Eds. San Francisco, CA: Morgan Kaufmann Publishers, 456-462.
9. Cantú-Paz, E.(1999). Topologies, migration rates, and multi-population parallel genetic algorithms. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds. San Francisco, CA: Morgan Kaufmann, 91-98.
10. Cantú-Paz, E.(2000). *Efficient and accurate parallel genetic algorithms*. Boston: Kluwer Academic Publishers.
11. Cantú-Paz, E.(2001). Migration policies, selection pressure, and parallel algorithms. *Journal of Heuristics*. v. 7, n.4, 311-334.
12. Cao, H. Q., Kang, L. S., Chen, Y. P., & Yu, J. X.(2000). Evolutionary modeling of system of ordinary differential equations with genetic programming. *Genetic Programming and*

Evolvable Machines, v.1, n.4, 309-337.

13. Fernández, F., Roa, L. M., Tomassini, M. & Sanchez, J. M.(2000). Multipopulation genetic programming applied to burn diagnosing. In *Proc. 2000 Congress on Evolutionary Computation*, IEEE Press, 1292-1296.
14. Goldberg, D. E.(1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison Welsey.
15. Grefenstette, J. J.(1981). Parallel adaptive algorithms for function optimization Tech. Rep. No. CS-81-19, Vanderbilt University, Computer Science Department, Nashville, TN.
16. Grosso, P. B.(1985). *Computer simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model*. Ph.D. dissertation, The University of Michigan, Ann Arbor.
17. Holland, J. H.(1975). *Adaptation in natural and artificial system*. Ann Arbor, MI: University of Michigan Press.
18. Koza, J. R.(1992). *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.
19. Koza, J. R.(1994). *Genetic programming II: automatic discovery of reusable programs*. Cambridge, MA: MIT Press.
20. Lin, S. -C., Punch, W., & Goodman, E.(1994). Coarse-grain parallel genetic algorithms: Categorization and new approach. In *Proc. 6th IEEE Symposium on Parallel and Distributed Processing*. Los Alamitos, CA: IEEE Computer Society Press.
21. Nordin, P., Hoffmann, F., Francone, F. D., Brameier M. & Banzhaf, W.(1999). AIM-GP and parallelism. In *Proc. 1999 Congress on Evolutionary Computation, IEEE Press*, 1059-1066.
22. Punch, W. F., Zongker, D., & Goodman, E. D.(1996). The royal tree problem: A benchmark for single and multiple population genetic programming. In *Advances in Genetic Programming 2*, Angeline, P. J. & Kinnear Jr, K. E. Eds. Cambridge, MA: MIT Press, 299-316.
23. Tanese, R.(1987). Parallel genetic algorithm for a hypercube. In *Proceedings of the Second International Conference on Genetic Algorithms*. Grefenstette, J. J., Ed., 177-183.
24. Tanese, R.(1989). Distributed genetic algorithms. In *Proc. 3rd International Conference on Genetic Algorithms*, 434-439.