Various island-based parallel genetic algorithms for the 2-page drawing problem

VARIOUS ISLAND-BASED PARALLEL GENETIC ALGORITHMS FOR THE 2-PAGE DRAWING PROBLEM[1]

Hongmei He, Ondrej Sýkora, Ana Salagean
Department of Computer Science, Loughborough University
Loughborough, Leicestershire LE11 3TU, The United Kingdom
{*H.He, A.M.Salagean@lboro.ac.uk*}

ABSTRACT
Genetic algorithms have been applied to solve the 2-page drawing problem successfully, but they work with one global population, so the search time and space are limited. Parallelization provides an attractive prospect in improving the efficiency and solution quality of genetic algorithms. One of the most popular tools for parallel computing is Message Passing Interface (MPI). In this paper, we present four island models of Parallel Genetic Algorithms with MPI: island models with linear, grid, random graph topologies, and island model with periodical synchronisation. We compare their efficiency and quality of solutions for the 2-page drawing problem on a variety of graphs.

KEY WORDS
Island-based Parallel genetic algorithms, 2-page book crossing number, Cartesian topologies, Periodical synchronization, Computation-to-Communication ratio, Population-to-Chromosome ratio.

1 Introduction

The simplest graph drawing method is that of putting the vertices of a graph on a line and drawing the edges as half-circles. Such drawings are called book drawings, and they correspond to the linear VLSI design. Edge crossing minimisation is the most important goal in the linear VLSI design, since smaller number of crossings means cheaper design. The minimal number of edge crossings in all possible book drawings of a graph is called book crossing number of the graph [15].

In the 2-page drawing one places the vertices of a graph G along a line called spine and every edge is completely drawn in one of two pages. The smallest number of crossings in all possible drawings of G is called the 2-page crossing number of G, denoted by $\nu_2(G)$. Equivalently, the vertices can be put on a circle and the edges can be drawn as straight lines and coloured by two colours. The 2-page crossing number is the same as the smallest crossing number of edges with the same colour. The problem is NP-hard [11].

[1]This research was supported by the EPSRC grant GR/S76694/01.

Genetic algorithms (GAs) have shown to be good global optimizers for a broad range of optimisation problems, and they have been used successfully for drawing graphs [1, 5, 8, 13].

Parallel processing, the method of having many small tasks solve one large problem, has emerged as a key enabling technology in modern computing. Recent studies have shown that parallelization can significantly improve the performance of genetic algorithms, and parallel genetic algorithms(PGAs) are particularly easy to be implemented and guarantee performance [2]. One of the most popular tools for parallel computing is Message Passing Interface(MPI), which provides flexible Cartesian virtual topologies of processors. It is easy to implement the communication of processors with these topologies of MPI. So in this paper we discuss three types of topologies: linear array, Cartesian grid and random graph, for the island model. An island model with periodical synchronization is presented as well. We compare the island model with fine-grained and master-slave models for the 2-page drawing problem on the benchmark library of Rome Graphs from GDToolkit [16], and investigate the performance of four island PGAs by testing them on a kind of special graphs, Xtree graphs.

2 Serial genetic algorithm(SGA) solving the 2-page crossing problem

We first describe the serial genetic algorithms as in [8]. The first population of solutions is generated randomly. Fitness, as a measure of quality of solutions, usually expressed in the form of one or multiple functions, is used to select the better solutions from the current population. The selected solutions undergo the operators of crossover and mutation in order to create a population of new solutions (the offspring population). The process is repeated until the termination criteria given by the user are met.

One of the most important questions is that of determining the characteristics of a problem, which makes it well-fitted for the genetic approach. The 2-page drawing problem is simplified to find a good order of vertices and distribution of edges so that the crossing number

of a 2-page drawing is as small as possible. According to the problem to be solved, we define four most important aspects of genetic algorithms for the 2-page drawing problem.

2.1 Chromosome

The key point of 2-page drawing is to find an ordering of the vertices and a distribution of edges minimising the number of edge crossings. So for an n-vertex, m-edge graph, G, a chromosome should include two parts, a permutation of all vertices, $\pi = (v_0, v_1, \ldots, v_{n-1})$ and a string, $S = (b_0, b_1, \ldots, b_{m-1})$, where $b_i \in \{1, 2\}$. Each b_i corresponds to one edge, i.e. b_i=1 indicates the corresponding edge is in page 1, b_i=2 indicates the corresponding edge is in page 2. If we consider κ page drawing, it is easy to extend our genetic algorithms by setting $b_i \in \{1, 2, \ldots, \kappa\}$.

2.2 The fitness function

Fitness functions are used to evaluate the quality of solutions. Our goal is to minimise the 2-page crossing number, therefore we directly define the fitness as the 2-page crossing number ν_2. The fitness function, $f(G, \pi, S)$, depends on the vertex order, π, and the edge distribution, S, in the current layout of a graph G. We use a table, adj, as an adjacent matrix in the current drawing $D(\pi, S)$ of G. If an element of S, which corresponds to an edge $e(u, v)$, has the value x (i.e. the edge $e(u, v)$ is drawn in page x, with $x = 1, \ldots, \kappa$), and the vertex u is in position i and the vertex v in position j in the current permutation π, then we set $adj[i][j] = adj[j][i] = x$. Otherwise, we set $adj[i][j] = adj[j][i] = 0$. We can calculate the number of crossings in a κ-page drawing of G with the following formula[10]. The calculation takes time $O(n^2)$.

$$v_\kappa(G) = \sum_{i=0}^{n-4} \sum_{j=i+2}^{n-2} \sum_{k=i+1}^{j} \sum_{l=j+1}^{n-1} adj[i][j] \bigodot adj[k][l]. \tag{1}$$

where

$$adj[i][j] \bigodot adj[k][l] = \begin{cases} 1 & \text{if } adj[i][k] = adj[j][l] \neq 0; \\ 0 & otherwise. \end{cases} \tag{2}$$

2.3 The genetic operators

Selection

Various selection criteria can be used so that sufficiently good individuals are picked for mating (and subsequent crossover). One of the simplest criteria is to select the best individual in each generation. All individuals will crossover with the best solution, and keep the better one of the two children generated by genetic operators. We call it as best-select criterion. The running time is $O(\varrho)$, where ϱ is the size of population.

Another usual select criterion is roulette. A probability, *prob*, is used to decide the selection operator, and the probability of each individual in the current population is in reverse ratio to the square of the fitness value. The smaller the crossing number is, the larger is the probability. The probability can be calculated by the following formula, where D_i is the drawing corresponding to the i-th individual in a population:

$$prob(D_i) = \frac{\frac{1}{\nu_2^2(D_i)+1}}{\sum_{k=0}^{\varrho-1} \frac{1}{\nu_2^2(D_k)+1}} \times 100\%. \tag{3}$$

When a random number is produced, and it is located in the probability range of a chromosome, the chromosome will be selected. The select operator is similar with the one used by He et al. [7] for outerplanar drawing problem. The running time is $O(\varrho)$.

Crossover

The purpose of crossover is to create new solutions by combining current solutions that have shown to be good temporary solutions. Depending on the presentation of chromosomes, different crossover operators are used. For the 2-page drawing problem, the chromosome includes two parts, permutation of vertices, π, and distribution of edges, S, therefore the crossover will act on both parts. In the implementation, we use two circle queues to maintain the permutation and edge distribution, respectively, so that the variation of permutation and edge distribution by crossover operators is double as that by using normal queues.

(1) Crossover on permutation, π

Here we use the Order Crossover(OX) [12]: two parental permutations, π_1 and π_2, are chosen randomly depending on the probability of being chosen. A continuous interval of the permutation π_1 is chosen, and also an interval starting at the same position and of the same length from π_2. Two new permutations, π_1' and π_2', are created such that π_1' contains the interval from π_2 with the rest being the other elements of π_1 in the same order as they were in π_1. π_2' contains the interval from π_1 with the rest being the other elements of π_2 in the same order as they were in π_2. The crossover operator on π is similar to the one in paper [7]. The running time of crossover(π_1,π_2) is $O(n)$.

(2) Crossover on page distribution of edges, S

We use Multi-Point Crossover (MPX) [12] on two parental strings, s_1 and s_2. A continuous interval of the string s_1 and an interval starting at the same position and length from s_2 are chosen. The two parents, s_1 and s_2 swap the two selected intervals to get two new distributions of edges, s_1' and s_2'. The worst run-

ning time is $O(m)$.

Mutation

After creating ϱ children by the crossover operator, the step of mutation is executed. The mutation is done with some probability on each child in the population. Actually, there is a small chance that the crossover on π gets synchronous performance as the crossover on S. Our preliminary experiments shown that a large probability of mutation is needed. Good results are achieved where the probability of mutation is 40%. The mutation operator acts on both parts of the chromosome, π and S. On π, the mutation is the swap of two randomly picked elements in a permutation; On S, the mutation is the change of a randomly picked element in a string, which indicates that the corresponding edge is changed to the opposite page from its current page. The running time is $O(1)$. Finally, the historical best individual will replace the worst one in the new generation.

2.4 Termination Criteria

The termination criterion is an important parameter, which determines the running time and final result of the algorithm. For the 2-page drawing problem, it is usually related to the number of edges or the number of the vertices. There are two termination criteria.

One is that the GA process will be terminated when the search space becomes very small [8]. In other words, when the chance of improvement is close to 0, GA process will stop. For example, we can define: $duration = min\{2(|V| + |E|) + 100, 300\}$, and the evolution procedure will be repeated until the best solution shows no improvement up to *duration* generations. In this way, if the *duration* is large, the evolution will run for a long time, but might get better results, and the exact number of evolution generations is related to the randomness of solutions in each run.

The other termination criterion is to define a maximal number of evolution generations. For our problem, the problem size is considered as a factor to affect the number of maximal generations *maxgn*. In our experiments, we define: $maxgn = min\{20(|V| + |E|) + 100, 3000\}$. In this way, the termination criterion is only related to the problem size, not to the randomness of solutions in each run.

3 Models of PGA

GAs are good candidates for effective parallelisation, given their inspiring principle of evolving in parallel a population of individuals [3]. There are three main types of PGAs: (1) global single-population master-slave GAs, (2) single-population fine-grained, and (3) multi-population coarse-grained GAs (also known as "island" PGAs) [2, 3].

Usually, a classical genetic algorithm is implemented in a master-slave (MS) model, and the master takes charge of select, crossover, and mutation operation, while the slaves make evaluation for each individual respectively. Here the MS model has a little change, in which the master only does a select operation, all other operations, including fitness calculation, are done on slaves. Independently from machine architecture, the big problem for MS model is that all processors work synchronously in each generation.

Fine-grained (FG) PGAs have only one population, and one individual on each processor, but it has a special structure that limits the interactions between individuals. An individual can only compete and mate with their neighbours, but since the neighbours overlap, a good individual can flow and spread like a migration to the neighbours of the deme [2]. This approach has the advantage of working with large populations enabling fast convergence, and reducing the number of iterations and the execution time, however, it needs huge hardware resource.

As an improvement of fine-grained model, in an island model, several isolated sub-populations (of size $k > 1$) evolve in parallel and a serial GA works on each processor. Periodically, their best individuals migrate to neighbours, and if the best neighbour is better than the local best individual, the island will replace the local best individual with the best neighbour, otherwise, replace the current worst individual with the best neighbour. We denote it as ISLAND. In this paper, we mainly discuss various ISLAND models and their performance.

4 Implementations of ISLAND model

There are a diversity of implementations for an ISLAND model. MPI provides flexibility of topologies, such as linear, cartesian grid, and random graph topology. We will describe these three topologies. Moreover, using a feature of MS model, an ISLAND model with periodical synchronization is implemented.

4.1 Linear Topology

The simplest topology is the one-dimension ring (linear) topology, which is the default topology in MPI. We denote it as ISLAND-Linear, which is similar with the "Island-SGA" in [6]. In the implementation, each island receives information from its left neighbour and sends information to its right neighbour. Each island gets the best individual of the left neighbour, compares it with the local best solution, and replaces the local best solution with the received one, if it is better than the local one, otherwise replaces the worst individual in the current subpopulation if it is better than the worst individual. In this model, a good solution can be shifted through the whole population. We

use the rank of a processor to denote a current island (Fig. 1). A fixed interval, *step*, between an island and its neighbour is set (default $step = 1$). So, $neb_{left} = (rank - step) \bmod M$, and $neb_{right} = (rank + step) \bmod M$, where M is the number of processors.

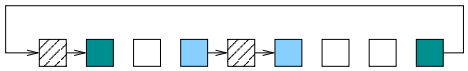
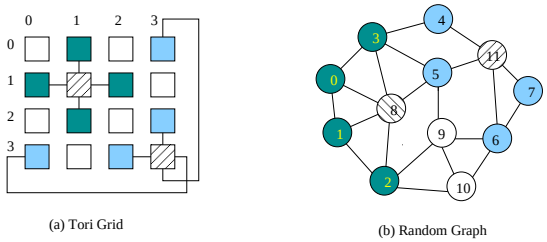


Figure 1. Linear Topology

4.2 Grid topology

MPI provides two types of Cartesian topologies: Cartesian grid and random graph. Many applications use a grid-based topology, two or more dimensional grids, or tori. We implement an ISLAND model with the 2D toroidal grid topology of the cluster, and denote it as ISLAND-Grid. All islands can communicate with their neighbours. Fig 2 (a) shows the 2-dimensional toroidal grid and the same-colour neighbours of each island. Each island, $\Gamma(i, j)$, has 4 neighbours. For example, if the grid of the cluster is $z \times z$, then the neighbours of each island are $neb_{up} = \Gamma(i, (j-1) \bmod z)$, $neb_{down} = \Gamma(i, (j+1) \bmod z)$, $neb_{left} = \Gamma((i-1) \bmod z, j)$, $neb_{right} = \Gamma((i+1) \bmod z, j)$. Each island receives information from its neighbours, and gets the best individual in 4 neighbours. If the best neighbour is better than the local best solution, the local best will be replaced with the best neighbour, otherwise, if the best neighbour is better than the worst individual in current subpopulation, the worst individual will be replaced with the best neighbour.



(a) Tori Grid (b) Random Graph

Figure 2. Cartesian Topologies

4.3 Graph topology

The other Cartesian topology supported by MPI is the random connected graph topology, and we denote it as ISLAND-Graph. Each island is mapped to a vertex of the virtual graph. Usually, we use a random biconnected graph G_{top}. All neighbours of an island are the virtual vertices, which are connected to the virtual vertex mapped to the island (Fig. 2 (b)). If the degree of a virtual vertex in G_{top} is d, then the island mapped to the virtual vertex will communicate with d neighbours.

4.4 A model with periodical synchronization

Discarding the useless and developing the useful features of MS model, we present a novel implementation for ISLAND model. In the implementation, each island runs a serial genetic algorithm, and periodically, island 0 will collect the best solutions of all islands, get the global best solution, and send the global best solution to all islands. Each island replaces the local best solution with the global best solution. We denote it as ISLAND-PS. Algorithm 1 briefly describes the implementation of the ISLAND model, where L is the migration period, k is the size of subpopulation, *subpop* is used to store a subpopulation, and G is the graph to be tested.

Algorithm 1 island_ps(G)

```
1:  MPI_Bcast(G) from rank 0;
2:  Initialisation(maxgn, L, k, subpop);
3:  while (gn < maxgn) do
4:    run operators of SGA
5:    if (gn mod L =0) then
6:      MPI_Gather(localbest) to lBests of rank 0;
7:      gBest=getGlobalBest(lBests) on rank 0;
8:      MPI_Bcast(gBest) to localbest of each island;
9:    end if
10: end while
```

5 Complexity of ISLAND PGAs

The 2-page drawing problem is NP-hard [11], and the GA runs as unsupervised evolution. If we use the first termination criterion described in Section 2.4, then the number of generations of each run will not be a constant for the same graph. So we use the second termination criterion described in Section 2.4 to guarantee every run of GA has the same number of generations for same graph. Therefore, the running time in the following sections is an average time of every generation. There are three main factors that determine the performance of ISLAND-based PGAs.

- **Chromosome size (ς)**, which includes two parameters: vertex number, n, and edge number, m, for the 2-page drawing problem. Chromosome size affects the running time of genetic operators, such as crossover, mutation, and fitness calculation directly.
- **Population size (ϱ)**, for SGAs, a small population size indicates the variation of chromosomes is small in each generation, and search time is short,

but it may lead to premature convergence of solutions. A large population size indicates the variation of chromosomes is large, and the search time is longer, but it may get better solutions. Therefore, the population size affects the final solution and running time directly. For ISLAND PGAs, the global population size is decided by the subpopulation size of each island and the cluster size. A large subpopulation indicates longer computing time, and a large cluster indicates the global population is larger, but the communication time could be longer or shorter depending on the topology of the cluster used and the architecture of the parallel machine.

- **Period (L)**, parallel GAs are more complex than their serial counterparts. In particular, the migration of individuals from one island (deme) to another is controlled by several parameters like: (a) the topology that defines the connections between the subpopulations, (b) a migration rate that control how many islands migrate, and (c) a migration interval that affects the frequency of migrations [2]. We will investigate the effect of the migration interval (period) L on the performance of ISLAND model with different topologies on fixed migration rate (100%). The migration period L, after which each island transfers the local best solution to its neighbours, will decide the speed, at which a good solution will pass through the whole population, therefore it affects the convergence of PGAs.

Chromosome size indicates the problem size, while population size indicates the space each generation explores. So the Population size-to-Chromosome size ratio ($PTC = \frac{g}{\varsigma}$) is an important factor that affects the quality of solution. Usually, small PTC indicates that each generation explores a small space relative to the whole space; Conversely large PTC indicates that each generation explores a large space relative to the whole space, therefore, there is a large chance to get a better solution. If each island keeps the same size of subpopulation, and PGAs run on same size cluster, then they have same PTC for same problem.

The running time T of each generation in PGAs can be divided into two parts, t_{comp} and t_{comm}, where t_{comp} is computation time, and t_{comm} is communication time in each generation. For evaluating performance of PGAs, the Computation-to-Communication ratio definitely plays an important role [4]. We denote it as $CTC = \frac{t_{comp}}{t_{comm}}$. In all ISLAND implementations, the computation time on each island is given by $t_{comp} = t_{crossover} + t_{mutation} + t_{fitness} + t_{select}$ for each graph tested. The former three items are related to the chromosome size, while the last item is related to the subpopulation size k. The difference in CTC between ISLAND models mainly results from the communication time. Therefore, we will mainly discuss the communication time (t_{comm}) of each ISLAND model below.

For ISLAND-Linear, the population size does not affect the running time T, as each island will communicate with 2 neighbours periodically, receiving from left neighbour and sending to right neighbour. We denote the cluster size as M, and the time for one cycle of receiving and sending a chromosome as t_{cycle}. The communication time in the period of L generations, $t_{comm} = t_{cycle}$. The t_{cycle} depends on the size of data transferred (chromosome size), network speed, and network traffic. Passing through a good solution to the whole cluster needs $M - 1$ periods in the worst case.

For ISLAND-Grid, the communication time is given by $t_{comm} = 4t_{cycle}$, as each island will communicate with 4 neighbours. The population size also does not affect the running time significantly. Passing through a good solution to the whole cluster needs $M/2$ periods in the worst case.

For ISLAND-Graph, a vertex degree d indicates the island mapped to the vertex has d neighbours, so the communication time in the period of L generations is given by $t_{comm} = dt_{cycle}$. Passing through a good solution to the whole population needs $M - 1$ periods in the worst case, if the topology, random biconnected graph, is only a circle. Usually, it needs $log(m(G_{top}))$ periods to pass through a good solution to the whole cluster, where $m(G_{top})$ is the number of edges in the random biconnected graph topology.

For ISLAND-PS, the time of communication in the period of L generations is given by $t_{comm} = Mt_{cycle}$, as island 0 collects all local best solutions, and sends the global best solution to all islands. The two routines, *MPI_Gather* and *MPI_Bcast*, provided by MPI standard make the implementation easy, and the communication faster than the pair of *MPI_send* and *MPI_receive* do. In this way, the communication time in the period of L generations is given by $t_{comm} = t_{Gather} + t_{Bcast}$. Namely, all islands are synchronized in one period of L generations. Table 1 shows the details of CTC for all ISLAND models.

Table 1. time complexity of each generation on each deme for 4 ISLAND models, where n=vertex number, m=edge number, M=cluster size, k=subpopulation size, L=migration period, d=degree of vertex in the random graph topology

Models	$t_{comp} = t_{fit} + t_{sel} + t_{cross} + t_{mut}$	t_{comm}
Linear	$O(n^2) + O(M) + O(k(n+m)) + O(k)$	$O((n+m)/L)$
Grid	$O(n^2) + O(M) + O(k(n+m)) + O(k)$	$O(4(m+n)/L)$
Graph	$O(n^2) + O(M) + O(k(n+m)) + O(k)$	$O(d(n+m)/L)$
PS	$O(n^2) + O(M) + O(k(n+m)) + O(k)$	$O(M(n+m)/L)$

Comparison with FG and MS models

The biggest different is that the size of subpopulation k is larger than 1 for ISLAND model, while for FG and MS models $k = 1$, and usually MS model does not affect the behavior of GA [2]. If FG model and ISLAND model use same topology, then t_{comp}(ISLAND)=kt_{comp}(FG), while t_{comm}(ISLAND)=t_{comm}(FG)/L. Comparing with the MS model, each deme of the ISLAND model communicates with other demes in the period of L generations, so the t_{comm}(ISLAND) $\leq t_{comm}$(MS)/L, and t_{comp}(ISLAND) = kt_{comp}(MS).

6 Experimental results

Our test platform is a SGI Altix 350 parallel machine, which offers global shared memory in configurations of 32 Intel Itanium 2 microprocessors with 1.5GHz and 1.6GHz, based on the 64-bit Linux operating system and 384GB of memory. At startup we broadcast the graph tested to all processors, then every processor runs in its own RAM. Therefore our parallel programs are as independent as possible from the architecture of the parallel machine.

6.1 Test on Rome graphs with three PGAs

We use a subset of Rome graphs [16], RND_BUP, which is a set of Random Biconnected Undirected Planar graphs. To examine the performance of three parallel models, ISLAND, FG, and MS models, our experiments are done on a fixed PTC, namely, for the same graph, all models run on nearly same population size. We use the second termination criterion described in section 2.4, so that the number of evolution generations is not related to the randomness of solutions in each run of PGAs, but only related to the chromosome size, which means that the number of evolution generations of the three PGA models is the same for the same graph. Moreover, for the MS model, all slaves are synchronous with the master. Our preliminary experiments show that the MS model got premature convergence, when we used the best-select criterion described in section 2.3. Therefore, in the implementation of the MS model, we use the roulette select criterion. We tested the ISLAND model with subpopulation $k = 4, 8$ (denoted as ISLAND-4,ISLAND-8 respectively), where the ISLAND model is implemented with a linear topology. Table 2 lists the conditions of all tests.

RND_BUP is a set of Random Biconnected Undirected Planar graphs in Rome graphs from GDToolkit [16], and includes 10 groups of graphs, for which the vertex number ranges from 10 to 100, and each group has 20 different graphs with same vertex number. The experimental results and running times are the average values of 20 graphs in each group. Fig. 3 shows that

Table 2. Test conditions of the three PGA models

Model	M	k	ϱ	L	select	topology
MS	16	1	15	1	roulette	NA
FG	16	1	16	1	best-select	Linear
ISLAND-4	4	4	16	50	best-select	Linear
ISLAND-8	16	8	128	50	best-select	Linear

the ISLAND model achieves the best results, although the running time is longer than the other two models. When the chromosome size is smaller, the MS model gets slightly better results than the FG model. The MS and FG models have nearly same running time. The running time is mainly decided by t_{comp} rather than t_{comm}, as we use SGI Altix shared memory parallel machine. Fig. 3 indicates that the ISLAND model can improve the solution when having a same PTC as MS and FG models. When the PTC increases, the solutions are further improved (see the curve for ISLAND-8 in Fig. 3 (a)). The running time of ISLAND-8 is nearly double one of ISLAND-4, as the subpopulation size of ISLAND-8 is double one of ISLAND-4. In the following subsection, we will further explore the performance of ISLAND models.

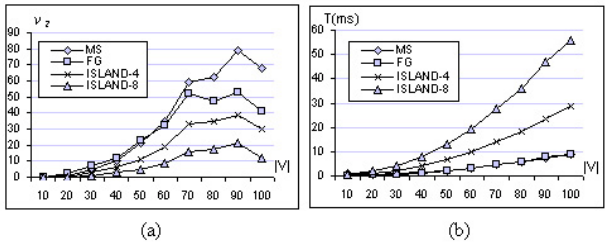


Figure 3. Test of three basic PGA models on RND_BUP

6.2 Test on Xtree with four ISLAND models

(1)Different chromosome size:

We tested a special graph, Xtree. An Xtree consists of a complete binary tree, in which the vertices of each level are connected in turn. We denote an Xtree with x levels as Xt(x). Our experiments are done on a fixed cluster size 16. For the ISLAND-Graph model, we use a random biconnected graph with the same edge density as that of the toroidal grid. Table 3 shows the test results of the four ISLAND models, which are average values of 10 tests. From the experimental results, the 2-page drawing crossing number is 0 for Xt(4) and Xt(5). For Xt(8), ISLAND-Graph achieves the best results out of the four ISLAND models. Table 3 indicates that the position of ISLAND-Graph becomes better with the rise of chromosome size, if we list the

four ISLAND models in the order of crossing numbers obtained.

Table 3. Crossing number on Xtree with all PGAs

Xtree	Linear	Grid	Graph	PS
Xt(4)	0	0	0	0
Xt(5)	0	1	1	2
Xt(6)	12	16	17	13
Xt(7)	185	208	171	144
Xt(8)	1831	1902	1470	1644

(2)Different cluster size:

To examine the effects of the different cluster sizes, we test a fixed graph $Xt(8)$ with all ISLAND models on different cluster sizes (4, 8, 16). Namely we fix the chromosome size ς and subpopulation size k. Obviously, from Fig.4 (a), with the rise of cluster size, the results become better for all ISLAND models. However, the interesting thing is that the running time of each generation for all ISLAND models on Xt(8) even decreases (Fig.4 (b)). To examine the running time, Xt(4) - Xt(7) are investigated as well. The running time has a increasing tendency with the rise of processor number, but the increasing tendency with the rise of cluster size becomes smaller and fluctuation becomes larger for a larger graph. By Xt(7), the running time has a decreasing tendency with the rise of cluster size, but still a little fluctuation. However, regardless of the architecture of the parallel machine, the running time should not change, except some fluctuation caused by network traffic, for the same test conditions, such as same graph, subpopulation size, migration period and number of evolution generations.

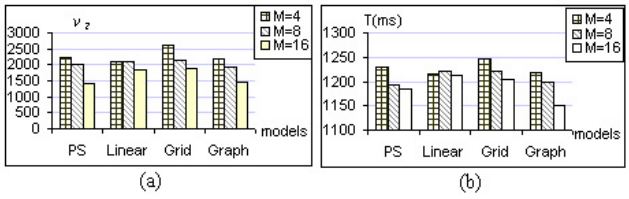


Figure 4. Average results of 10 tests with the four ISLAND models on different cluster size

(3)Different migration period:

To examine the effect of the migration period on convergence, we tested all ISLAND models with migration periods, $L = \{50, 100, 200\}$ respectively, on the graph Xt(8). Fig. 5 shows that the results, when the migration period L is 50, are the best for all ISLAND models. The results of all ISLAND models become worse with the rise of the migration period. For ISLAND-PS model, the difference of crossings caused by migration period is the largest in all ISLAND models.

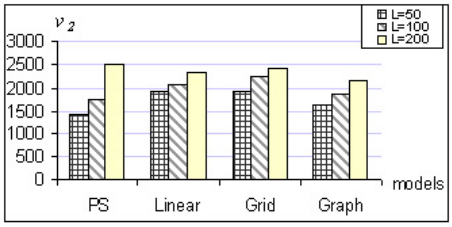


Figure 5. Crossing number of Xt(8) by 4 ISLAND models based on difference migration periods

(4)Test of "speedup"

The most important goal of parallelising a serial algorithm is to speed it up. The "speedup" can be defined as $speedup = \frac{T(SGA)}{T(PGA)}$. However, this may be ubsuitable because of different hardware platforms. So usually we express the "speedup" as the ratio of the expected running time for one processor to that for the M parallel processors ($speedup = \frac{T(1processor)}{T(Mprocessors)}$) [14]. For a stochastic algorithm there is an inherent difficulty [14], however, as we use the second termination described in Section 2.4 to guarantee the same number of generations for each tested graph. Namely we fix the "PTC" for each run, so that there is the same chance to get a good solution for all tests of a graph. To examine the "speedup", we keep the global population size at 32, and compare the results by running GA with population size=32 on one processor and by running ISLAND PGAs with subpopulation sizes 16,8,4,2 on 2,4,8,16 processors respectively. Fig. 6 presents the results and the "speedup" of the four ISLAND models. It can be seen that the four ISLAND models have similar "speedup" for a range of processor numbers, and achieve nearly same best results when running on 2 processors, this is because with 2 processors all topologies coincide. However, when more than 2 processors, difference of results for all ISLAND models is revealed. Fig. 6 also shows the results are different at different sizes of subpopulation (k) even when the size of the global population is same, as subpopulation size affects the convergence of SGA on each island. Obviously, for the problem tested, all models, except ISLAND-Grid, achieve the best results at $k = 16, 8$, but the model with $k = 8$ has a double speed of the model with $k = 16$. For a larger problem, it is necessary to increase the size of global population to get a larger search space. Namely, PTC should be large enough to guarantee having a larger chance to get the best solution. Therefore for an ISLAND model there should be a combination of parameters that gives the

best performance. For a real larger problem, it is important to look for the best combination of parameters. Fig. 6 (a) indicates that ISLAND-PS achieves the best performance, and ISLAND-Graph has a similar performance for a large problem.

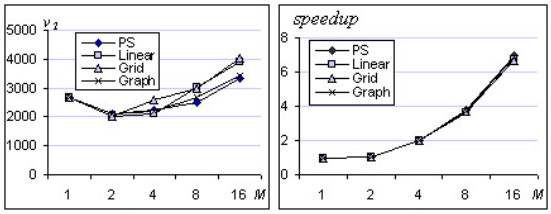


Figure 6. GA on one processor and ISLAND models on the fixed global population of 32 for a range of processor numbers

7 Conclusion

Serial genetic algorithms are a powerful search tool, but they work with one global population, so the search time and space are limited. Parallel genetic algorithms are a perfect way to improve genetic algorithms both in efficiency and search space. In this paper we examine the ISLAND model of PGAs. Comparing with MS and FG models, ISLAND model can avoid a large amount of communication, have more diversity, and prevent premature convergence. There are two evaluation measures, Computation-to-Communication ratio(CTC), and Population size-to-Chromosome size ratio(PTC). The former can be a measure of efficiency for PGAs, the later links to effectiveness of PGAs. A large number of parameters of PGAs can affect these two measures. We investigate the effect of *PTC* and *CTC* on the quality of solutions and efficiency of PGAs by testing four implementations of the ISLAND model: ISLAND-Linear, ISLAND_Grid, ISLAND_Graph, and ISLAND_PS, on different population sizes (including subpopulation sizes), cluster sizes, and migration periods. The experimental results show that ISLAND-PS achieves the best performance, and ISLAND-Graph has a similar performance for a large problem.

References

[1] Barreto, A.M.S., Barbosa, H.J.C.: Graph layout using a genetic algorithm. *Proceedings*, VI Brazilian Symposium on Neural Networks (SBRN'00), 2000, pp.179-184.

[2] Cantú-Paz, E.: A survey of parallel genetic algorithms. Calculateurs Parallèles, Reseaux et Systems Repartis, 10(2),(1998) pp.141-171.

[3] Dorigo, M., Maaniezzo, V.: Parallel Genetic Algorithms: Introduction and Overview of Current Research, J. Stender, Ed. IOS Press, 1993.

[4] Digalakis, J., Margaritis, K.: Parallel Evolutionary Algorithms on MPI, Inernational Conference on Computer, Communication and Control Technologies, Orlando, USA, CCCT'2003.

[5] Eloranta, T., Mäkinen, E.: TimGA: a genetic algorithm for drawing undirected graphs. *Divulg. Mat.* **9** (2001), no. 2, pp.155-170.

[6] Gordon, V.S., Whitley,D.: Serial and parallel genetic algorithms as function optimizers. Proceedings of the Fifth International Conference on Genetic Algorithms (S. Forrest, ed.), (1993) pp.177-183.

[7] He, H, Newton, M. C., Sýkora, O.: Genetic Algorithms for Bipartite and Outerplanar Graph Drawings are best!, in: Communications. *SOFSEM'2005*, pp.51-60.

[8] He, H., Sykora, O., Erkki Mäkinen: Genetic algorithms for 2-page drawing problem, submitted, Journal of Heuristics.

[9] He, H, Sýkora, O: New Circular Drawing Algorihtms, in: Proc. ITAT'2004.

[10] He, H., Sykora, O., Vrt'o, I.: Heuristic Crossing Minimisation Algorithms For 2-page Drawings. The Electronic Notes in Discrete Mathematics (ENDM) **22** (2005) pp. 527-534.

[11] Masuda, S., Kashiwabara, T., Nakajima, K., Fujisawa, T.: Crossing minimization in linear embeddings of graphs, *IEEE Trans. Comput.* **39** (1990) pp.124-127.

[12] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, Second, Extended Edition, Springer 1994, p.211.

[13] Radwan, Ahmed A. A., El-Sayed, Mohamed A.: Using genetic algorithm for drawing triangulated planar graphs. *J. Inst. Math. Comput. Sci.* Comput. Sci. Ser. **15** (2004), no. 1, pp.137-147.

[14] Shonkwiler, R. Parallel Genetic Algorithms. In Forrest S. (ed.): Proceedings of the Fifth International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1993) pp.199-205

[15] Shahrokhi, F., Sýkora, O., Székely, L.A., Vrt'o, I.: The book crossing number of a graph, Journal of Graph Theory 21 (1996), pp.413-424.

[16] GDToolkit. http://www.dia.uniroma3.it/∼ gdt/