Parallelisation of Genetic Algorithms for the 2-page Crossing Number Problem [1]

Hongmei He[1], Ondrej Sýkora[1], Ana Salagean[1], Erkki Mäkinen[2]

[1]*Department of Computer Science, Loughborough University Loughborough, Leicestershire LE11 3TU, The United Kingdom*

[2]*Department of Computer Sciences, P.O. Box 607, FIN-33014 University of Tampere, Finland*

Abstract

Genetic algorithms have been applied to solve the 2-page crossing number problem successfully, but since they work with one global population, the search time and space are limited. Parallelisation provides an attractive prospect to improve the efficiency and solution quality of genetic algorithms. This paper investigates the complexity of parallel genetic algorithms (PGAs) based on two evaluation measures: Computation-time to Communication-time and Population-size to Chromosome-size. Moreover, the paper unifies the framework of PGA models with the function *PGA*(*subpopulation size*, *cluster size*, *migration period*, *topology*), and explores the performance of PGAs for the 2-page crossing number problem.

Key words: 2-page crossing number, Parallel genetic algorithms, Evaluation measures.

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 main goal in the linear VLSI design, since smaller number of crossings means cheaper design. The minimal number of edge crossings over all book drawings of a graph is called the *book crossing number* (29).

Email address: [1]`H.He, A.M.Salagean@lboro.ac.uk`, [2]`em@cs.uta.fi` (Hongmei He[1], Ondrej Sýkora[1], Ana Salagean[1], Erkki Mäkinen[2]).

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

In the 2-page book 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 over all 2-page drawings of G is called the 2-page crossing number of G, denoted by $\nu_2(G)$. Figure 1 shows sample 2-page drawings of K_6, for which $\nu_2(G) = 3$, with five (Fig. 1 (a)) and three crossings (Fig. 1 (b)). Equivalently, the vertices can be put on a circle and the edges can be drawn as straight lines coloured by two colours. The 2-page crossing number is the same as the minimal number of crossings of edges with the same colour. The problem is NP-hard (26). In contrast, the *book thickness problem* is to minimise the number of pages used so that edges embedded on the same page do not intersect.

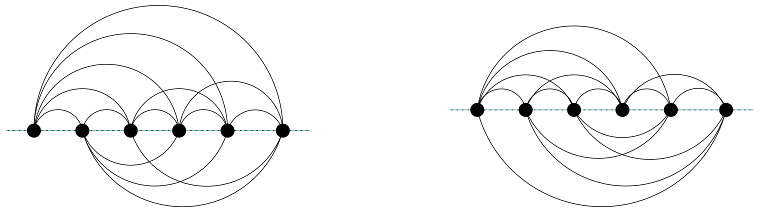

(a) Five crossings (b) Three crossings

Fig. 1. 2-page drawings of K_6 with five and three crossings.

Genetic algorithms (GAs) have proved to be good global optimizers for a broad range of optimisation problems, and they have been used successfully for drawing graphs (3; 14; 22; 28). Moreover, Kapoor et al. presented a genetic algorithm for the book thickness problem in (25).

Cimikowski (9) tested eight different heuristic algorithms where the order of vertices was determined by finding a Hamiltonian cycle. Hence, contrary to our genetic algorithm, in his tests the order of vertices was always fixed. Recently, Winterbach (33) proposed heuristics for the 2-page crossing numbers and applied them to estimating the plane crossing number of some small complete multipartite graphs. A tabu algorithm was used to produce a good arrangement of vertices, and then either GreedySide algorithm or the neural network of Cimikowski and Shope (10) was used to find a good distribution of edges. Our genetic algorithm finds a good vertex order and edge distribution directly, and we got the same results for the complete multipartite graphs tested by Winterbach, except for a single graph (22).

Parallelisation can significantly improve the performance of genetic algorithms (1; 2; 15; 16; 17), and parallel genetic algorithms are easy to implement (5). One of the most popular tools for parallel computing is Message Passing Interface (MPI) (34), which provides flexible Cartesian virtual topologies of processors.

In this paper we analyse the complexity of three basic models of parallel genetic algorithms: master-slave, fine-grained, and coarse-grained models. We also examine the effect of different topologies and parameters on the performance of PGAs by testing them on the benchmark graph library (Rome graphs (35)) and on a special kind of graphs (Xtrees), when solving the 2-page crossing problem.

2 Sequential genetic algorithm solving the 2-page crossing problem

We have described sequential genetic algorithms (SGA) for the 2-page crossing problem in (22). The first population of solutions is generated randomly. Fitness, as a measure of quality of solutions, is used to select the better solutions from the current population. The selected solutions undergo the operations 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 issues is to abstract the characteristics of the problem to make the problem well-fitted for the genetic approach. According to the 2-page crossing number problem, we have defined the four most important aspects of genetic algorithms as described below.

2.1 Chromosome

The crux of 2-page drawing is to find an order of the vertices and a distribution of the edges minimising the number of edge crossings. Thus, for an n-vertex, m-edge graph, 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 element of S corresponds to an edge, and $b_i = 1$ indicates that the corresponding edge is in page 1, and $b_i = 2$ indicates that the corresponding edge is in page 2. As an example, consider a graph with 6 vertices and the sorted list $((v_0, v_2), (v_0, v_4), (v_0, v_5), (v_1, v_2), (v_1, v_4), (v_2, v_3), (v_2, v_5), (v_3, v_5), (v_4, v_5))$ of edges. Now, a random individual is described as $\pi = (v_3, v_0, v_2, v_4, v_5, v_1)$, and $S = (1, 1, 1, 2, 2, 2, 1, 1, 2)$. Fig. 2 shows the 2-page book drawing corresponding to this individual. If we consider κ page drawings, it is easy to extend our genetic algorithms by setting $b_i \in \{1, 2, \ldots, \kappa\}$. It is also applicable with small modifications for the book thickness problem.

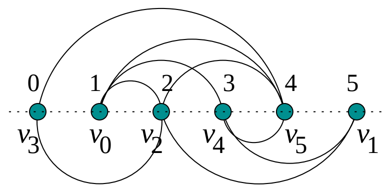


Fig. 2. The 2-page drawing of a random graph corresponding to an random individual.

2.2 The fitness function

The quality of solutions is evaluated by fitness functions. 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, $f(\pi, S)$, depends on the vertex order, π, and the edge distribution, S. 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 $x \in \{1, \ldots, \kappa\}$), the vertex u is in position i, and the vertex v in position j in the current permutation π, then we set $adj[i][j] = adj[j][i] = x$. If there is no edge between the vertices in positions i and j, we set $adj[i][j] = adj[j][i] = 0$. We can calculate in time $O(n^2)$ the number of crossings in a κ-page drawing of G with the following formula (24):

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

where

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

For the 2-page drawing in Fig. 2, we have the adjacency matrix

$$adj = \begin{pmatrix} 0\ 0\ 2\ 0\ 1\ 0 \\ 0\ 0\ 1\ 1\ 1\ 0 \\ 2\ 1\ 0\ 0\ 1\ 2 \\ 0\ 1\ 0\ 0\ 2\ 2 \\ 1\ 1\ 1\ 2\ 0\ 0 \\ 0\ 0\ 2\ 2\ 0\ 0 \end{pmatrix},$$

and the 2-page crossing number is 1.

2.3 The genetic operators

2.3.1 Selection

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

Another common selection scheme is roulette-wheel. A probability, *prob*, is used to decide the selection operator, and the probability of each individual in the current population is inversely proportional to the square of the fitness value. The smaller the crossing number is, the larger the probability. The probability can be calculated by the following formula, where D_i is the drawing corresponding to the i-th individual in the 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 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. (21) for the outerplanar drawing problem. The running time is $O(\varrho)$.

2.3.2 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 crossing number problem, the chromosome includes two parts, permutation of vertices and distribution of edges, and the crossover will act on both parts. In the implementation, we use two circular queues to maintain the permutation and edge distribution, respectively, so that the variation of permutation and edge distribution by crossover operators is double compared to the case of using normal queues.

For the crossover on permutation we use Order Crossover (OX) (27) (Fig. 3): 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. (21). The running time of crossover(π_1,π_2) is $O(n)$.

For the crossover on page distribution of edges we use Multi-Point Crossover (MPX) (27) 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. 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 running time is $O(m)$.

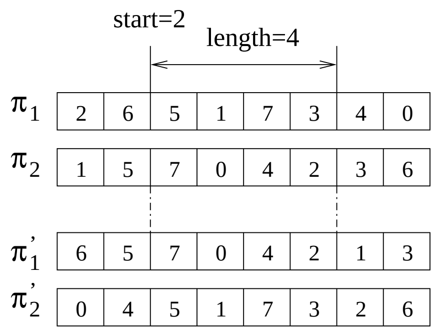


Fig. 3. An example of the crossover operation

2.3.3 Mutation

After crossover operation, the step of mutation is executed. The mutation is done randomly with some probability on each child in the population. Good results are achieved when 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 so far best individual found will replace the worst one in the new generation.

2.4 The termination criteria

The termination criteria are important parameters, which greatly affect the running time and the final crossing number of the algorithm. For the 2-page crossing number problem, they are usually related to the number of edges or the number of vertices. There are two termination criteria.

One is that the GA process will be terminated when the chance of improvement is close to 0 (22). The evolution procedure will be repeated until the best solution shows no further improvement up to a before-hand defined number *due* of generations or the minimal crossing number is 0. If *due* is large, the evolution will run for a long time, but might get better solutions. The *due* parameter is fixed for a graph, but the exact number of generations is not only related to evolution procedure, but also to the randomness of evolution in each run. Our preliminary tests showed that a suitable definition for *due* is $due = min\{3n + 3m + 100, 300\}$.

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 maximal number of generations $maxgn$. Based on our preliminary tests, we define $maxgn = min\{20(n + m) + 100, 3000\}$. In this way, the termination criterion is only related to the problem size, not to the randomness of solutions in each run.

3 Parallelisation of genetic algorithms

Genetic algorithms are good candidates for effective parallelisation, given their inspiring principle of evolving in parallel a population of individuals (13). There are three basic types of parallel genetic algorithms (4; 5; 6; 8; 13): (1) global single-population master-slave GAs, (2) single-population fine-grained, and (3) multi-population coarse-grained GAs (also known as "island" PGAs).

3.1 Master-slave model

Usually in the master-slave (MS) model, the master takes charge of select, crossover, and mutation operations, while the slaves do the evaluation for the individuals. In our implementation, the MS model has changed so that the master only performs the select operation. All other operations, including the fitness calculation, are done on slaves. The procedure is repeated until the termination criterion is met. Independently from machine architecture, the main problem for the MS model is that all processors work synchronously in each generation. Each slave processor holds one individual of the population. Thus, a large number of processors is needed and required to be synchronised with the master processor.

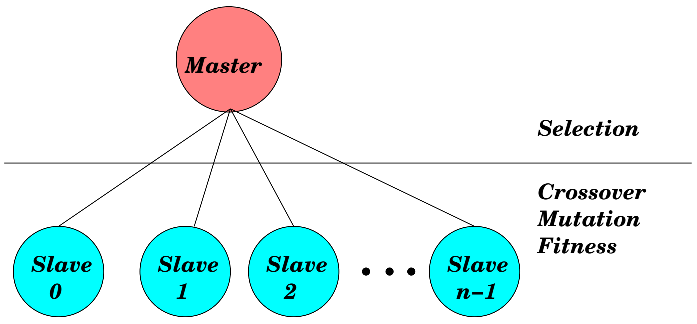


Fig. 4. Master-Slave Model

3.2 Fine-grained model

Fine-grained (FG) PGAs have only one population, but it has a special structure that limits the interactions between individuals. As in the master-slave model, also the FG model has the property that each processor holds one individual of the population. An individual can only compete and mate with its neighbours. If we describe the computation on each cell with an automaton, each cell has three states. The states of all cells are updated according to a local rule, called a transition function, which replaces the selection operator in the sequential genetic algorithm (Fig. 5):

- State 1: Randomly generate a chromosome, permutation (ordering of vertices) and string (distribution of edges), and calculate the fitness value of each individual.
- State 2: Get the best neighbour after the transition function is run.
- State 3: Get two offsprings after applying crossover between the current chromosome and the best neighbour. Then the mutation operator, with probability p_m, is applied to the two offsprings, of which, the fitter one is used to update the current chromosome (state 1 is updated).

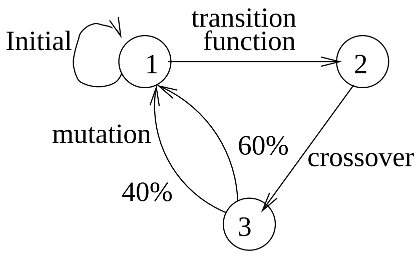


Fig. 5. Automaton of each cell

The state of the entire automaton is evolved step by step. The global behaviour of the system is determined by the evolution of the states of all the cells as a result of multiple interactions. Since the neighbours overlap, a good individual can flow and spread like a migration to neighbours of the cell (4). So a good solution can diffuse rapidly through the whole population. This approach has the advantage of fast convergence, and reducing the number of iterations and the execution time. However, a large population size indicates that there is a need of more hardware resources, as the cluster size is given by $M = \varrho$.

In the implementation of the FG model, we use a local hill-climbing strategy as in ASPARAGOS (19; 20): after the chromosome has not been improved for N generations, a new random chromosome takes effect at the beginning of the next iteration.

3.3 Coarse-grained model

In the coarse-grained model, several isolated subpopulations (of size $k > 1$) evolve in parallel and an SGA works on each processor. Periodically, the best individuals of each subpopulation migrate to the neighbouring subpopulations (Fig. 6). If the best neighbour received by an island is better than the best local solution, then the island will replace the best local solution with the best neighbour. Otherwise, if the best neighbour is better than the worst individual in the subpopulation, then the island will replace the worst individual with the best neighbour. We denote the coarse-grained model as ISLAND.

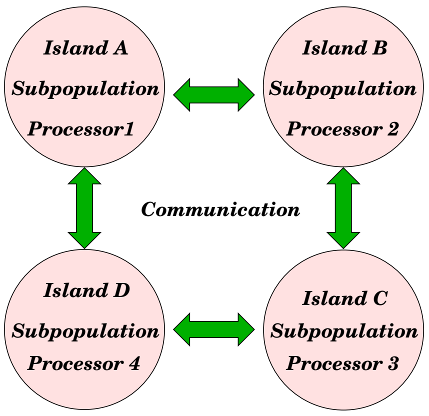


Fig. 6. ISLAND Model

4 Time complexity of PGAs

There are two major factors that determine the performance of PGAs, the chromosome size (ς) and the population size (ϱ). In the 2-page crossing number problem the chromosome size depends on the size of the input graph, and it directly affects the running time of genetic operators, such as crossover, mutation, and fitness calculation.

In SGAs, a small population size indicates that 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 that the variation of chromosomes is large, and the search time is longer, but it may get better solutions. Therefore, the population size directly affects the final solution and running time.

The Population size-to-Chromosome size ratio ($PTC = \frac{\varrho}{\varsigma}$) is an important factor that affects the quality of solution. Usually, a small PTC indicates that each generation explores a small space relative to the whole space. Conversely a large PTC indicates that each generation explores a large space related to the whole space, so there is a large chance to get a better solution.

The running time of each generation in PGAs can be divided into computation time t_{comp} and communication t_{comm}. For evaluating performance of PGAs, the Computation-to-Communication ratio $CTC = \frac{t_{comp}}{t_{comm}}$ plays an important role (12).

4.1 MS model

Since each slave processor holds a single individual, the number of processors needed is one more than the size of the population. The population size affects the running time significantly, as the master must work with the slaves synchronously, and all the slaves will communicate with the master in each generation. We denote the number of processors (the cluster size) as M, and the time for one cycle of receiving and sending a chromosome as t_{cycle}.

The communication time t_{cycle} for each generation is the time needed for sending a chromosome to a slave and, after processing it, back to the master. Of course, the processing time is excluded from t_{cycle}. The communication time is affected not only by the chromosome size, but also by the network traffic and speed. However, we assume that the local network speed is constant. Then t_{cycle} is directly proportional to the size of data transmitted, i.e., the chromosome size, and we have $t_{comm} = O(M(n + m))$.

The computation time in each generation is $t_{comp} = t_{crossover} + t_{mutation} + t_{fitness} + t_{select}$. The first three items are related to the chromosome size, while the last item is related to the population size.

4.2 FG model

For the FG model, the communication time of each generation on each cell is given by $t_{comm} = l \times t_{cycle}$, where l is the number of neighbours and and t_{cycle} is the time for one cycle of receiving and sending a chromosome. The number of neighbours depends on the topology of the cluster, while t_{cycle} depends on the chromosome size and the network speed. Again, the computation time is given by $t_{comp} = t_{crossover} + t_{mutation} + t_{fitness} + t_{select}$, where the first three items depend on the chromosome size, and the last item t_{select} is related to the number of neighbours. The difference of running time of the FG model with different topologies results mainly from t_{comm}, which is $O(l(m + n))$, where l is the number of neighbours of current cell in the cluster topology.

4.3 ISLAND model

For the ISLAND model, the migration of individuals from one island (also called a *deme*) to another is controlled by several parameters: (a) the topology that defines the connections between the subpopulations, (b) the migration rate that controls how many individuals migrate, and (c) the migration interval that affects the frequency of migrations (4). We investigate the effect of the (*migration period*) L on the performance of the ISLAND model with different topologies on fixed migration rate (100%). The migration period, after which each island transfers the best local solution to its neighbours, defines the speed at which a good solution will spread through the whole population. Therefore it affects the convergence of PGAs.

The computation time is given by $t_{comp} = t_{crossover} + t_{mutation} + t_{select} + t_{fitness}$, where $t_{crossover}$ and $t_{mutation}$ are related to the chromosome size, while t_{select} is related to the size of subpopulation, k. The communication time of each generation t_{comm} is $O(l(n + m)/L)$. As in the FG model, a good solution can cross through the whole population.

4.4 Comparison of the three models

Table 1 shows the components of computation time in each generation for each model. The computation time is the sum of the evaluation time and the genetic operation time. The numbers of edges and vertices in a graph always fulfill the condition $m < n^2$. For the FG model, the number of neighbours is less than the cluster size M, and usually $l < n$. Therefore, we can write t_{comp} and t_{comm} for each model as shown in Table 2.

Table 1
Computation time of each generation on each processor for SGA and PGAs

	fitness	crossover	mutate	select
SGA	$O(\varrho n^2)$	$O(\varrho(n+m))$	$O(\varrho)$	$O(\varrho)$
MS	$O(n^2)$	$O(n+m)$	$O(1)$	$O(M)$
FG	$O(n^2)$	$O(n+m)$	$O(1)$	$O(l)$
ISLAND	$O(kn^2)$	$O(k(n+m))$	$O(k)$	$O(k)$

Table 2
Computation time and communication time in each generation on each processor for SGA and PGAs

	t_{comp}	t_{comm}
SGA	$O(\varrho n^2)$	0
MS	$O(n^2+M)$	$O(Mn^2)$
FG	$O(n^2)$	$O(ln^2)$
ISLAND	$O(kn^2)$	$O(ln^2/L)$

From Table 2 we get the parameters CTC and PTC as shown in Table 3. Usually the MS model retains the behavior of SGA (4), while the ISLAND and FG models do not. If the FG and ISLAND models use the same topology, then $t_{comp}(\text{ISLAND}) = kt_{comp}(\text{FG})$, while $t_{comm}(\text{ISLAND}) = t_{comm}(\text{FG})/L$. Since each deme of the ISLAND model communicates with other demes in the period of L generations, we have $t_{comm}(\text{ISLAND}) \leq t_{comm}(\text{MS})/L$, and $t_{comp}(\text{ISLAND}) = kt_{comp}(\text{MS})$. Actually, the FG model can be viewed as a special case of the ISLAND model with subpopulation size $k = 1$ and migration period $L = 1$.

As Table 3 shows, the ISLAND model has bigger PTC than the other models. Theoretically, this should mean that the ISLAND model explores a larger portion if the search space than the other models. For any reasonable choose of the topology, the ISLAND model has also bigger CTC than the other models.

5 Different topologies for the implementation of PGAs

The topology is an important factor affecting the performance of the PGAs because it determines how fast a good solution disseminates to other demes

Table 3
CTC and PTC for SGA and PGAs

	CTC	PTC
SGA	-	$\frac{\varrho}{\varsigma}$
MS	$O(\frac{n^2+M}{Mn^2})$	$\frac{M}{\varsigma}$
FG	$O(\frac{1}{l})$	$\frac{M}{\varsigma}$
ISLAND	$O(\frac{kL}{l})$	$\frac{kM}{\varsigma}$

(4). If the topology has a dense connectivity (or a short diameter, or both) good solutions will spread rapidly to all the demes and may quickly take over the population. On the other hand, if the topology is sparsely connected (or has a large diameter), solutions will spread slower and the demes will be more isolated from each other, permitting the appearance of different solutions. Different solutions may come together at a later time and recombine to form better individuals. The communication topology is also a major factor in the cost of migration. A densely connected topology may promote a better mixing of individuals, but it also entails higher communication costs.

5.1 Linear topology

The simplest topology is the one-dimensional linear (ring) topology, which is the default topology in MPI, abbreviated as Linear. Gordon and Whitley (18) implemented a multi-deme PGA with ring topology. In this topology, each processor receives information from its left neighbour and sends information to its right neighbour. We use the rank of a processor to denote the current processor.

5.2 Grid topology

MPI provides two types of Cartesian topologies: Cartesian grid and random graph. It is common to use a 2-dimensional grid topology in fine-grained PGAs, because in many massive parallel computers the processing elements are connected with this topology (4). Cantú-Paz and Mejía-Olver considered the 4×4 toroidal mesh in (7). In the 4x4 toroidal mesh topology (7), each processor can communicate with all its neighbours. For example, if the grid is $z \times z$, the neighbours of processor $\Gamma(i, j)$ are $neb_{up} = \Gamma(i, (j-1) \bmod z)$, $neb_{down} = \Gamma(i, (j+1) \bmod z)$, $neb_{left} = \Gamma((i-1) \bmod z, j)$, and $neb_{right} = \Gamma((i+1) \bmod z, j)$. Each processor receives information from its neighbours, and gets the best individual from four neighbours. We denote the 2D toroidal mesh as Grid.

5.3 Graph topology

The other Cartesian topology supported by MPI is the random connected graph topology abbreviated as Graph. Each island is mapped to a vertex of the virtual graph G_{top}. All neighbours of a processor are the virtual vertices which are connected to the virtual vertex mapped to the island. 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. There has been a lot of research on PGAs with dense structural topologies, such as hypercubes (11; 31; 32), 4×4 toroidal mesh (7), and bidirectional rings. However, we use a random biconnected graph.

5.4 A novel ISLAND model

A novel implementation of the ISLAND model was presented in (23): each island runs a sequential genetic algorithm, and periodically, island 0 will collect the best local solutions of all islands, find out the best global solution, and send it to all islands. Each island replaces the best local solution with the best global solution. We denote the new model as ISLAND-PS. Two MPI functions, *MPI_ Gather* and *MPI_ Bcast* make the implementation quite easy. Actually, the novel ISLAND model is a special case of the ISLAND model with the equivalent complete graph topology with M vertices, but we do not really set the topology with the virtual complete graph. Algorithm 1 sketches the implementation of the ISLAND-PS model, where L is the migration period, k is the size of subpopulation, *subpop* is used to store a subpopulation, G is the graph to be tested, and *lBests* is an array on island 0 of M chromosomes, which is used for storing the best local solution from each island periodically.

Algorithm 1 island_ps(G)

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

6 A unified formalisation of PGAs

A large number of parameters can affect the performance of PGAs. In this paper, we consider the following four factors: subpopulation size (k), cluster size (M), migration period (L), and topology (G_{top}), which fix the framework of basic PGA models. If we formalise an island model of PGAs with the function $PGA(k, M, L, G_{top})$, then the models can be viewed as special cases of the ISLAND model (see Table 4).

ISLAND-PS is synchronized periodically by collecting the best local solution of each island and broadcasting the best global solution to all islands. So, ISLAND-PS can be viewed as the ISLAND model with topology K_M (the complete graph with M vertices).

In the FG model, each cell can be viewed as an island in the ISLAND model with subpopulation size $k = 1$ and migration period $L = 1$.

The MS model is synchronized in each generation. If the master runs the best-select operation described in Section 2.3, the MS model is equivalent to the FG model with topology K_M. However, if the master runs the select operator "roulette-wheel", the MS model is equivalent to the FG model with a dynamic topology (G_{dyn}) depending on the roulette-wheel selection, although all cells of the MS model are synchronized in each generation. SGA can be viewed as the ISLAND model on one processor without migration and topology.

Table 4
Formalization of different models of PGAs

Model	Formula
ISLAND	$PGA(k, M, L, G_{top})$
ISLAND-PS	$PGA(k, M, L, K_M)$
FG	$PGA(1, M, 1, G_{top})$
MS	$PGA(1, M, 1, K_M)$ or $PGA(1, M, 1, G_{dyn})$
SGA	$PGA(k, 1, null, null)$

7 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. We use the model of the SGA described in Section 2, which obtains good performance and the running time does not fluctuate randomly (22). Moreover, we use the same crossover rate (100%) and mutation rate (40%) as in (22). To observe the effect of the four basic parameters in the function described in Section 6, we performed experiments by adjusting these parameters on the following test suites:

(1) RND_BUP, a subset of Rome graphs (35), is a set of random biconnected undirected planar graphs. It includes 10 groups of graphs, for which the

vertex number ranges from 10 to 100, and each group has 20 different graphs with the same vertex number. The experimental results and running times are the average values of 20 graphs in each group.

(2) An Xtree consists of a complete binary tree, in which the vertices of each level are connected in turn. An Xtree with h levels is denoted by $\mathrm{Xt}(h)$. We use Xtrees for testing the effect of different parameters on the performance of PGAs, as it has a special feature – the graph size will be doubled when the level is increased by 1. Thus, the effect of parameters will be distinct. The data are averages of 10 tests for each graph.

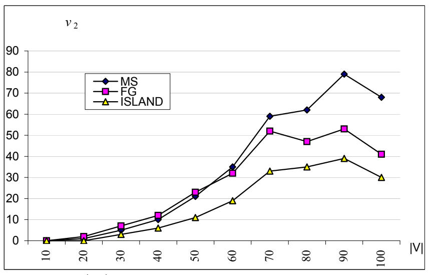
7.1 Tests of three models on RND_BUP

To examine the performance of three parallel models, ISLAND, FG, and MS, our experiments were done on a fixed PTC. Namely, for the same graph, all models ran on approximately the same population size. We used the second termination criterion described in Section 2.4, so that the number of evolution generations was not related to the randomness of solutions in each run of PGAs, but only related to the chromosome size. This means that the number of evolution generations of the three PGA models is the same for each graph. Moreover, for the MS model, all slaves are synchronous with the master. Our preliminary experiments showed that the MS model reached premature convergence, when we used the best-select criterion described in Section 2.3, and moreover, that the “roulette-wheel” was better than the “best-select” for the MS model. Therefore, in the implementation of the MS model, we used the roulette-wheel select criterion. The FG and ISLAND models were implemented with a linear topology, and the size of subpopulation of the ISLAND model was four. Table 5 lists the conditions of all tests.

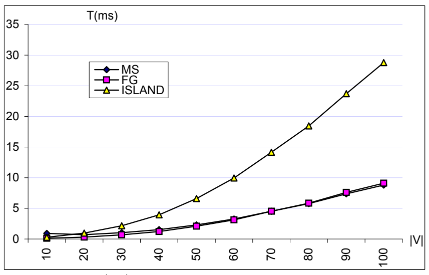
Table 5
Test conditions of the three PGA models

Model	M	k	ϱ	L	Select	Topology
MS	16	1	15	1	roulette-wheel	NA
FG	16	1	16	1	best-select	Linear
ISLAND	4	4	16	50	best-select	Linear

Fig. 7 shows that the ISLAND model achieves the best results, although the running time is longer than that of the other 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 the same running time. The running time is mainly determined by t_{comp} rather than t_{comm}, as we use SGI Altix shared memory parallel machine. Fig. 7 also indicates that the ISLAND model can improve the solution when having the same PTC as the MS and FG models.



(a) Crossing numbers



(b) Running times

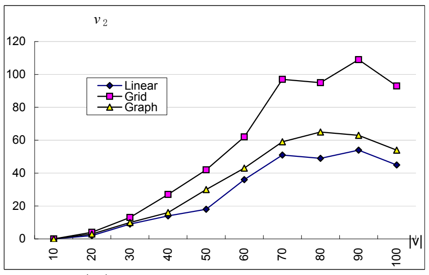
Fig. 7. Test of three basic PGA models on RND_BUP

7.2 Tests with different topologies on RND_BUP

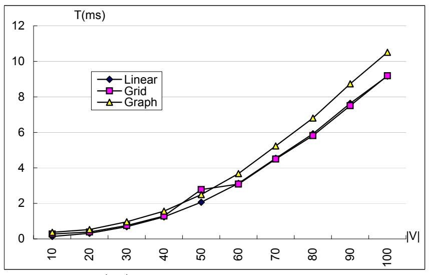
To examine the effects of topologies on the performance of PGAs, we tested Linear, Grid and Graph topologies. A random biconnected graph with the same edge density as the grid with the same vertex number was used for the graph topology in the implementation of PGAs.

7.2.1 ***Tests of FG models with different topologies***

First, we tested the effect of different topologies on the performance of the FG models. Our experiments were done on a fixed size cluster ($M = 16$), which means that the population size was fixed to be 16. Fig. 8 (a) and (b) show the crossing numbers and running times obtained by the FG-Linear, FG-Grid and FG-Graph models, respectively. The FG-Linear achieved the best performance both in results and running times. FG-Graph is slightly worse than FG-Linear, but the running time of FG-Graph was the worst of the three topologies. FG-Grid got the worst results, but it had almost the same running time as FG-Linear.



(a) Crossing numbers

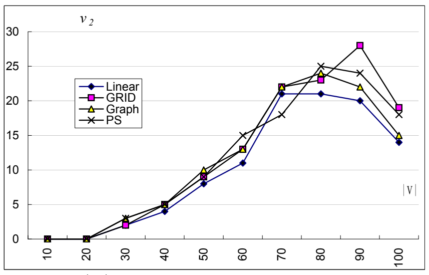


(b) Running times

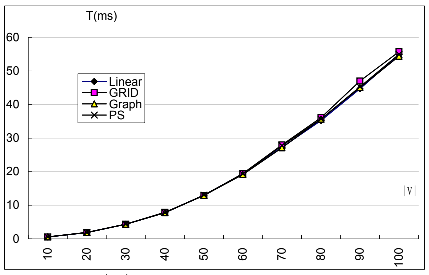
Fig. 8. Crossing number and Running time of FG models with different topologies on RND_BUP

7.2.2 ***Tests of ISLAND models with different topologies***

We tested the ISLAND models, including the ISLAND-PS, on RND_BUP to examine the effects of topologies. Our experiments were done on a fixed size cluster ($M = 9$) and a fixed size subpopulation ($k = 8$). The migration period was fixed to be 50. Fig. 9 shows the results and the running times. Obviously, this case differs from the FG tests. We cannot say which topology obtains the best performance. The running times for different topologies are about the same, because the migration is done once per migration period. Consequently, the average cost of communication in each generation for different topologies is nearly the same. The cases for ISLAND-Linear and ISLAND-Graph are similar with the cases for FG-Linear and FG-Graph. However, ISLAND-Grid differs from FG-Grid. When the chromosome size was smaller, ISLAND-Grid obtained almost the same crossing numbers as ISLAND-Graph, but when the vertex number was larger than 80, the results became much worse than the one of ISLAND-Graph. ISLAND-PS is the most variable one, some time it got the best result (e.g. when vertex number is 70), some time it shuttled between the other three models. When $|V| = 100$, ISLAND-Graph has results similar to ISLAND-Linear, and ISLAND-Grid has results similar to ISLAND-PS.



(a) Crossing numbers



(b) Running times

Fig. 9. Crossing numbers and running times of ISLAND models with different topologies on RND_BUP

7.3 Tests of ISLAND PGAs with different population size on Xtree

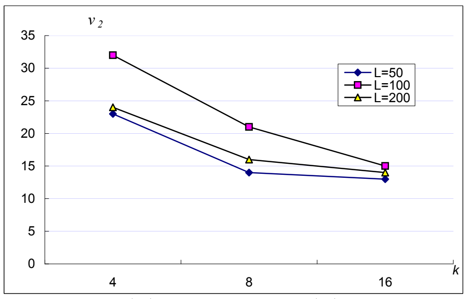
There are two ways to increase the population size. When the cluster size is limited, we can increase the subpopulation size. However, if we increase the subpopulation size too much, the running times for large graphs become intolerable. An alternative way to enlarge the population size is to keep the subpopulation size fixed while increasing the cluster size.

7.3.1 ***Different subpopulation sizes on fixed machine size***

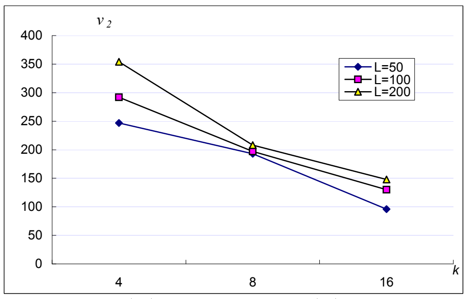
To examine the effect of subpopulation size, we tested the ISLAND-PS model with subpopulation sizes $k = 4, 8, 16$ on 16 processors, for $Xt(6)$ and $Xt(7)$. Fig. 10 shows the crossing numbers. The results become better with the rise of subpopulation size for each migration period.

7.3.2 ***Different cluster size on fixed subpopulation size***

To examine the effects of the different cluster sizes, we tested the ISLAND-Graph and ISLAND-Grid models with fixed subpopulation size ($k = 8$) on



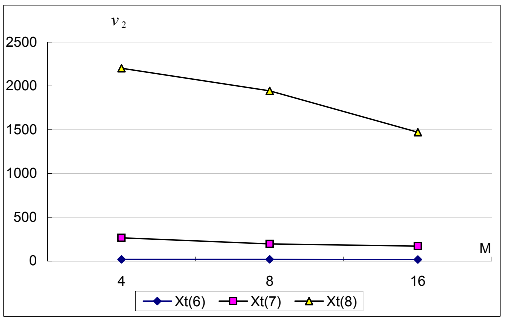
(a) Test on Xt(6)



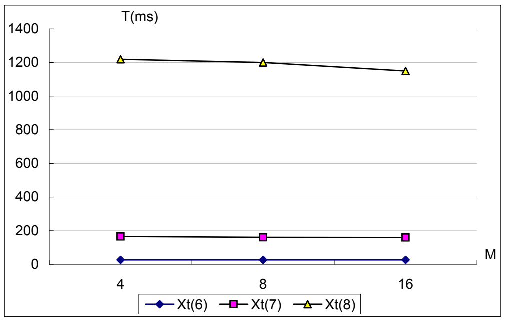
(b) Test on Xt(7)

Fig. 10. Crossing numbers when testing ISLAND-PS on Xt(6) and Xt(7) for a range of subpopulation size k and migration period L

different cluster sizes (4, 8, 16) for a group of graphs $Xt(6)$, $Xt(7)$, and $Xt(8)$. Fig.11 (a) and Fig. 12 (a) show the results become better with the increase of cluster size. However, the running time of each generation for both the ISLAND-Graph and ISLAND-Grid models on a fixed graph tested decreases with the rise of cluster size (Fig.11 (b) and Fig. 12 (b)). The reduction is more distinct when a larger graph is tested. However, regardless of the architecture of the parallel machine, the running time should not change, except for some fluctuation caused by network traffic.

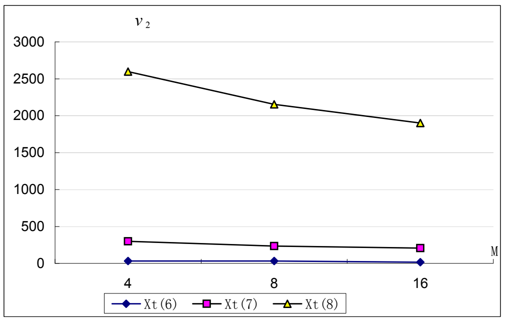


(a) Crossing numbers

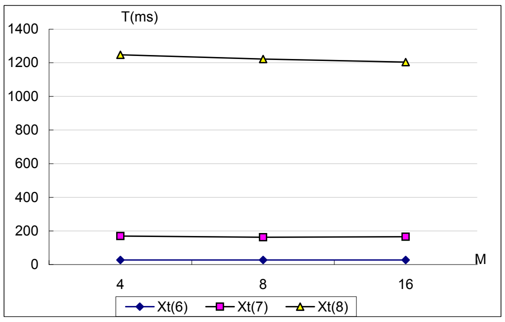


(b) Running times

Fig. 11. Average values of 10 tests with the ISLAND-Graph model on different cluster size



(a) Crossing numbers



(b) Running times

Fig. 12. Average values of 10 tests with the ISLAND-Grid model on different cluster size

7.4 Different migration periods

To examine the effect of the migration period on convergence, we tested the ISLAND-PS model with different migration periods, $L = 50$, 100, and 200 on 16 processors for $Xt(6)$ and $Xt(7)$. The best results were obtained when the migration period L was 50, as shown in Fig. 10. The results become worse with the rise of the migration period. There exists a value of k for which the difference of the results caused by the migration period is not significant. Figure 8(a) shows such a point for Xt(6), when $k = 8$, and Figure 8(b) for Xt(7), when $k = 16$. Fig. 13 shows the effect of migration period by testing the four island models with a fixed subpopulation size $k = 8$ on 16 processors with different topologies for a range of migration periods, $L = 50$, 100, and 200 on $Xt(8)$. For all models, similarly with the results of ISLAND-PS on Xt(6) and Xt(7), the results are the best when $L = 50$. For any L, ISLAND-Graph gets the best results with ISLAND-PS following close behind. The results of ISLAND-Linear and ISLAND-Grid are very similar to each other, and worse than the other two.

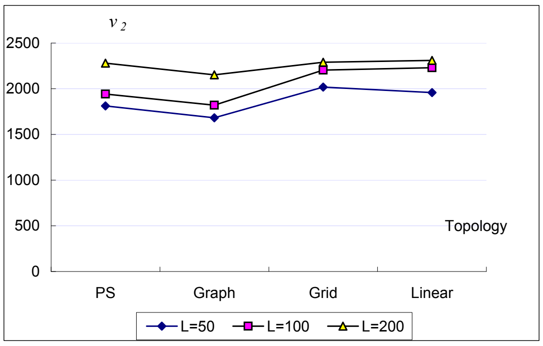


Fig. 13. Crossing numbers of Xt(8) by four ISLAND models with different topologies for different migration periods

7.5 A test of speedup

The most important goal of parallelising a sequential algorithm is to speed it up. The speedup can be defined as $speedup = \frac{T(SGA)}{T(PGA)}$. However, this may be unsuitable 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 M parallel processors ($speedup = \frac{T(1processor)}{T(Mprocessors)}$) (30). For a stochastic algorithm there is an inherent difficulty (30).However, as we use the second termination criterion 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 64, and compare the results by running GA with population size = 64 on one processor and by running ISLAND PGAs with subpopulation sizes 16, 8, 4 on 4, 8, 16 processors respectively. Fig. 14 presents the results and the speedup of the four ISLAND models. It can be seen that the four ISLAND models have nearly the same speedup for a range of processor numbers, and achieve super linear speedup (Fig. 14 (b)). From the point of view of the quality of solutions, for the smaller graphs, e.g., Xt(6) and Xt(7), the solutions produced by each model of ISLAND PGAs are very close, but for larger graphs, e.g. Xt(8), the solutions obtained are different (Fig. 14 (a)), and ISLAND_Graph model achieved the best performance of the four models. It is also shown that all four models achieved the best results when subpopulation size k is 16, and the results become worse with the decrease of subpopulation size. This is because subpopulation size affects the convergence of SGA on each island. 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 an increase of chance of obtaining the best solution. Therefore, it is important to look for the best combination of parameters. Fig. 14 (a) indicates that ISLAND-Graph achieves the best results and has similar running time with the other models.

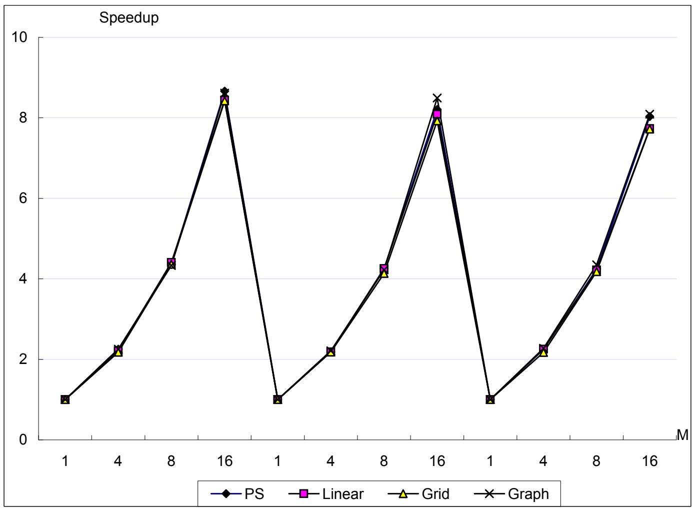
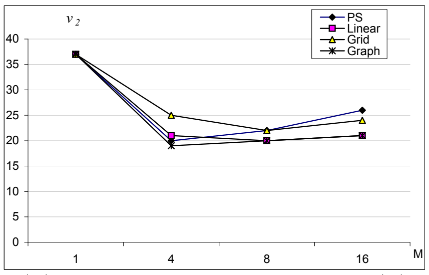


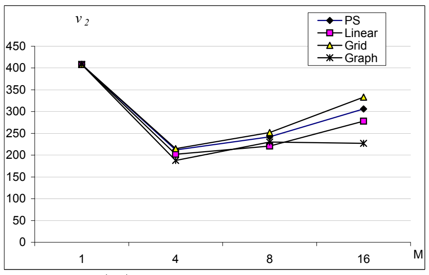
Fig. 14. The rates of the running times of the GA on one processor and the ISLAND models with the same global population size 32 as the GA for a range of processor numbers

8 Conclusions

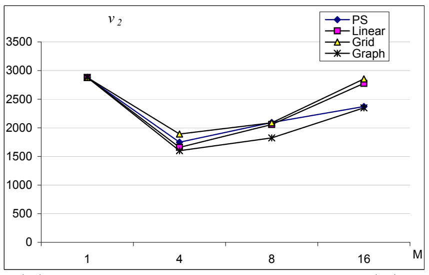
Sequential genetic algorithms are a powerful search tool, but they work with one global population, so the search time and space are limited. Parallel genetic algorithms are a perfect way to improve genetic algorithms both in efficiency and search space. In this paper we implemented three basic models of paral-



(a) Crossing numbers on Xt(6)



(b) Running times



(c) Crossing numbers on Xt(8)

Fig. 15. Crossing numbers of Xt(6), Xt(7) and Xt(8) tested with the four models on different sizes of the cluster

lel genetic algorithms: master-slave, fine-grained, and coarse-grained models. Comparing with other two basic models of PGAs, the ISLAND model achieves the best results. Especially for a large problem size, the ISLAND model has a complete superiority over the other models.

MPI provides automatic topology structure for the cluster. We have examined three topologies, Linear, Grid, and Graph. For the FG model, the Grid topology is overshadowed by the other two topologies. This might be because the FG-Grid passes a best solution through the whole population in several generations rapidly, resulting in a premature convergence. For the ISLAND model, the difference in the results of the topology is not significant.

There are two evaluating measures, Computation-time to Communication-time 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, such as population size, cluster size, size of subpopulation, migration period and so on. We unified all models of parallel genetic algorithms using a function $PGA(subpopulation\ size,\ cluster\ size,\ migration\ period,\ topology)$. The experimental data show that the ISLAND model has more diversity and it prevents premature convergence. Moreover, ISLAND-Linear achieves the best performance for smaller graphs, and ISLAND-Graph (with the topology of processors being a random biconnected graph) is the best for larger graphs, while ISLAND-PS (with the equivalent complete connected topology) is in the shade of ISLAND-Graph.

Acknowledgment

We would like to thank Prof. Robert Cimikowski for his useful comments.

References

[1] Alba, E., Nebro, A.J. and Troya, J.M.: Heterogeneous computing and parallel genetic algorithms. *Journal of Parallel and Distributed Computing* **65**, (2002), pp. 1362-1385.

[2] Alba, E. and Tomassini, M.: Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* **6**, 5, (2002), pp. 443-462.

[3] Barreto, A.M.S. and Barbosa, H.J.C.: Graph layout using a genetic algorithm. In Proc. of the VI Brazilian Symposium on Neural Networks (SBRN'00), (2000), pp. 179-184.

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

[5] Cantú-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms.* Kluwer, (2000).

[6] Cantú-Paz, E. and Goldberg, D. E.: Parallel Genetic Algorithms: Theory and Practice. *Computer Methods in Applied Machinics and Engineering* **186**, (2000), pp. 221-238.

[7] Cantú-Paz, E. and Mejía-Olver, M.: Experimental Results in Distributioed Genetic Algorihtms. In Proc. of the International Symposium on Applied Corporate Computing, Monterrey, Mexico, (1994), pp. 99-108.

[8] Chipperfield, A. and Fleming, P.: Parallel Genetic Algorithms. *Parallel and Distributed Computing Handbook* (Zomaya, A.Y. ed.) McGraw-Hill, (1996), pp. 1118-1143.

[9] Cimikowski, R.: Algorithms for the fixed linear crossing number problem, *Discrete Applied Mathematics* **122**, (2002), pp. 93-115.

[10] Cimikowski, R. and Shope, P.: A neural network algorithm for a graph layout problem, *IEEE Trans. on Neural Networks* **7**(2), (1996), pp. 341-346.

[11] Cohoon, J.P., Martin, W.N. and Richards, D.S.: Genetic Algorithms and Punctuated Equilibria in VLSI. *In Parallel Problem Solving from Nature* (Schwefel H.P., and Männer R. Eds.), Springer-Verlag (Berlin), (1991), pp. 134-144.

[12] Digalakis, J. and Margaritis, K.: Parallel Evolutionary Algorithms on MPI. In Proc. of the International Conference on Computer, Communication and Control Technologies, Orlando, USA, CCCT'2003.

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

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

[15] Goldberg, D.E.: *Genetic Algorithms in Search, and Machine Learning.* Addison-Wesley, Reading, MA, (1989).

[16] Goldberg, D.E.: The design of innovation: Lessons from genetic algorithms, lessons for the real world. *Technological Forecasting and Social Change*, (2000).

[17] Gong,Y., Nakamura, M., and Tamaki, S.: Parallel genetic algorithms on line topology of heterogeneous computing resourses. In *Proc. of the 2005 Conference on Genetic and Evolutionary Computation* (Bauer H.-G., Ed), 2005, pp. 1447-1454.

[18] Gordon, V.S. and Whitley, D.: Serial and Parallel Genetic Algorithms as Function Optimizers. In Proc. of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann (San Mateo, CA), (1993), pp. 177-183.

[19] Gorges-Schleuter, M.: ASPARAGOS: A population genetics approach to genetic algorithms. *Evolution and Optimization*(Voigt H.-M., Mühlenbein, H., Schwefel, H.-P. Eds.) **89**, (1989), pp. 86-94.

[20] Gorges-Schleuter, M.: ASPARAGOS: A asynchronous parallel genetic optimization strategy. In Proc. of the Third International Conference on Genetic Algorithms (Schaffer J. D. Ed.), Morgan Kaufmann (San Mateo, CA), (1989), pp. 422-428.

[21] He, H., Newton, M. C. and Sýkora, O.: Genetic Algorithms for Bipartite and Outerplanar Graph Drawings are best! In Communications of the 31st Annual Conference on Current Trends in Theory and Practice of Informatics. SOFSEM'2005, pp. 51-60.

[22] He, H., Sýkora, O. and Mäkinen, E.: Genetic algorithms for the 2-page book crossing number problem of graphs. Accepted, *Journal of Heuristics.*

[23] He, H., Sýkora, O. and Salagean, A. M.: Various Island-based Parallel Genetic Algorithms for the 2-page Drawing Problem, Forthcoming, IASTED International Conference on Parallel and Distributed Computing and Networks, PDCN2006.

[24] He, H., Sýkora, O. and Vrt'o, I.: Heuristic Crossing Minimisation Algorithms For 2-page Drawings (extended abstract). In Proc. of the 7th International Colloquium on Graph Theory, ICGT'05. *The Electronic Notes in Discrete Mathematics* (ENDM) **22**, (2005), pp. 527-534.

[25] Kapoor, N., Russell, M., Stojmenovic, I. and Zomaya, A.: A genetic algorithm for finding the pagenumber of interconnection networks, Journal of Parallel and Distributed Computing, **62**, 2, (2002), pp. 267-283.

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

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

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

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

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

[31] Tanese, R.: Distributed genetic algorithms. In Proc. of the Third Internatioinal Conference on Genetic Algorithms (Schaffer J. D., Ed.), (1989), pp. 434-439.

[32] Tanese, R.: Distributed genetic algorithms for function optimization. Unpublished doctoral dissertation, University of Michigan, Ann Arbor, (1989).

[33] Winterbach, W.: The crossing number of a graph in the plane, Master's thesis, Dept. Appl. Math, University of Stellenbosch, SA. (2005).

[34] The Message Passing Interface (MPI) standard: http://www-unix.mcs.anl.gov/mpi/.

[35] GDToolKit: http://www.dia.uniroma3.it/ gdt/.