9-1991

Parallel Genetic Algorithms with Application to Load Balancing for Parallel Computing

N. Mansouri
Syracuse University, Department of Engineering and Computer Science, namansou@ecs.syr.edu

Geoffrey C. Fox
Syracuse University

SU-CIS-91-48

Parallel Genetic Algorithms with Application to Load Balancing for Parallel Computing

N. Mansour and G.C. Fox

September, 1991

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, New York 13244-4100

PARALLEL GENETIC ALGORITHMS WITH APPLICATION TO LOAD BALANCING FOR PARALLEL COMPUTING

N. Mansour
School of Computer and Information Science
Syracuse Center for Computational Science

G. C. Fox
Northeast Parallel Architectures Center
Syracuse Center for Computational Science
Department of Physics
School of Computer and Information Science

SYRACUSE UNIVERSITY

ADDRESS FOR CORRESPONDENCE

Nashat Mansour

NPAC
111 College Place
Syracuse University
Syracuse NY 13244-4100

e-mail: nmansour@top.cis.syr.edu

Tel: (315) 443-4883

Abstract

A new coarse grain parallel genetic algorithm (PGA) and a new implementation of a data-parallel GA are presented in this paper. They are based on models of natural evolution in which the population is formed of discontinuous or continuous subpopulations. In addition to simulating natural evolution, the intrinsic parallelism in the two PGA's minimizes the possibility of premature convergence that the implementation of classic GA's often encounters. Intrinsic parallelism also allows the evolution of fit genotypes in a smaller number of generations in the PGA's than in sequential GA's, leading to superlinear speed-ups. The PGA's have been implemented on a hypercube and a Connection Machine, and their operation is demonstrated by applying them to the load balancing problem in parallel computing. The PGA's have found near-optimal solutions which are comparable to the solutions of a simulated annealing algorithm and are better than those produced by a sequential GA and by other load balancing methods. On one hand, The PGA's accentuate the advantage of parallel computers for simulating natural evolution. On the other hand, they represent new techniques for load balancing parallel computations.

Key words: Data allocation, data partitioning, load balancing, loosely-synchronous algorithms, natural evolution simulation, parallel genetic algorithms, physical optimization methods, task allocation.

1. INTRODUCTION

Genetic algorithms (GA's) are search techniques based on the mechanics of natural evolution, where species search for beneficial adaptations to a changing environment [12, 14]. In GA's, artificial evolution takes place over successive, usually discontinuous, generations for solving a problem. Each generation consists of a population of chromosomes, also called individuals, which represent possible solutions. The initial generation is created at random. Each consecutive generation is created by the individuals concurrently searching the adaptive topography. Firstly, individuals reproduce according to their fitness. Then, mates are selected and genetic operators are applied to create offsprings, which replace the parents. In this process, high-performance building blocks are propagated and combined to find fitter structures leading to optimal or near-optimal solutions. The parameters of this search strategy would be designed so that a balance between the exploitation of fitter structures and the exploration of the search space is secured for a sufficient number of generations.

Most of the GA work has considered the total population a single random mating unit from which parents can be selected. This model has two shortcomings. Firstly, the model is not quite relevant for species in nature. Natural populations are normally distributed in various ways that confine reproduction to subpopulations, with interaction among subpopulations. Secondly, the single mating unit population structure is one of the reasons for the premature convergence problem often encountered in the implementation of GA's [2] because it can allow the exploitation aspect of the genetic search to dominate. Therefore, the use of distributed population structures provides better models of natural evolution and helps in overcoming the problem of the convergence of the search into local optima.

Parallel GA's (PGA's) are suitable for simulating distributed population structures. Subpopulations

computer, and interactions among subpopulations can occur via the interconnection network. Different population structures can be modeled by different PGA's for suitable parallel computers. Because of this close association between distributed population GA's and their parallel implementation, we will henceforth not distinguish between them and will refer to both as parallel GA's. In addition to a more realistic mimic of natural evolution, PGA's obviously provide faster execution than sequential GA's. The use of the model of distributed population offers better speed-ups than straight parallelization of sequential GA's, because the latter requires global selection in the reproduction step and, thus, incurs the penalty of global interprocessor communication. In addition, straight parallelization does not contribute to the alleviation of premature convergence. Further, distributed population models enjoy intrinsic parallelism which leads to superlinear speed-ups. Intrinsic parallelism refers to the concurrent and independent exploration by the subpopulations of many different regions in the adaptive topography. A number of models for distributed natural population structures have been proposed in the population genetics literature [5, 13, 29, 30]; important models are summarized in section 4. Previous PGA's [3, 16, 20, 22, 23] share features with some of these models. Their operation have been demonstrated by solving problems such as the optimization of Walsh functions, the traveling salesperson problem...etc. These PGA's differ in the models they adopt for the population structure, in the mechanisms used for implementing some features of the models, and in the applications they deal with.

In this paper, new coarse-grain and fine-grain PGA's are presented. The coarse-grain PGA is based upon a model of discontinuous population structure and the theory of shifting balance of evolution [30]; it has been implemented on a hypercube. The fine-grain PGA is a data-parallel algorithm based on the isolation by distance model of populations with continuous distribution [29]; it has been implemented on the Connection Machine. The coarse-grain PGA offers faster convergence than do other coarse-grain PGA's, which is advantageous for many applications. The fine-grain PGA provides a model that exploits massive parallelism.

The operation of the PGA's is demonstrated in this work by applying them to an NP-complete optimization problem, namely the load balancing problem in parallel computing. However, it should be emphasized that the two PGA's represent general models which fit in the framework of physical computation [11]. The PGA's have general applicability, especially to time-demanding optimization problems. For example, genetic algorithms used in designing neural networks can take months on sequential computers and, thus, speed is of utmost importance for such an application [27]. Load balancing is concerned with equal distribution of the workload among the processors of a multicomputer. For loosely-synchronous algorithms, it is based on partitioning the underlying data set constituting the problem domain. This problem has been chosen as an application for the PGA's because it is an important problem in parallel computing and is a new application for GA's. Moreover, the results of a sequential GA [18] are available for comparison. Previous approaches to load balancing are based on techniques such as mincut-based heuristics, recursive bisection, simulated annealing, scattered decomposition and neural networks [1, 6, 7, 8, 9, 10, 19, 24, 28]. The performance of these techniques vary in terms of the quality of the solution produced and the execution time required. The experimental results reported below show that the proposed PGA's evolve near-optimal solutions which are superior to those produced by several previous methods.

This paper is organized as follows. Section 2 defines the load balancing problem. Section 3 presents a sequential GA, some of whose constituents are employed by the PGA's. Section 4 includes a brief summary of models of natural populations and a presentation of the PGA's based on two of these models. The experimental results are given and discussed in Sections 5 and 6. Conclusions are

given in Section 7.

2. LOAD BALANCING PROBLEM

Load balancing refers to the partitioning of a problem domain into disjoint subdomains and the assignment of the subdomains to the processors of a multicomputer such that an objective function, namely the total execution time, is minimized. Both the problem domain and the multicomputer are considered as graphs. The minimization of the objective function corresponds to balancing the calculations among the processors and minimizing/balancing the interprocessor communication. The objective function depends on the computation model. The model considered here is that of loose synchronicity [10], with all processors running the same code for the problem subdomains assigned to them. Loosely synchronous algorithms repeat a calculate-communicate cycle, where a processor carries out the calculations for its subdomain and then communicates with other processors to exchange necessary boundary information. In this computation model, the total execution time is determined by the slowest processor and, thus, the objective function is represented by the largest combined calculation-communication load. However, this exact objective function is too computationally expensive to use in genetic algorithms and is replaced by an approximate objective function given by

$$r^2 \sum_p N^2(p) + vR \sum_p \sum_q C(p,q) \quad ....(1)$$

where r is the ratio of the amount of calculation to the amount of communication per data element (a characteristic of the algorithm), $N(p)$ is the number of elements allocated to processor p, R is the ratio of the time needed to communicate a unit of information one unit distance to the time required for one calculation operation (a characteristic of the multicomputer), v is a constant scaling factor expressing the relative importance of communication with respect to calculation, and $C(p,q)$ is the communication cost between processors p and q. This objective function enjoys a locality property that greatly reduces the computational cost [18].

3. SEQUENTIAL GENETIC ALGORITHM

```
Read (problem graph and multicomputer graph);
Random Generation of initial population P(0) of size POP;
Evaluate fitness of individuals in P(0);
For (gen = 1 to maxgen) OR until convergence do
   Set (v, operator rates, freq-hillclimbing);
   Rank individuals in P(gen-1), and
        allocate reproduction trials stored in MATES[];
   /* produce new generation P(gen) */
   For (i = 1 to POP step 2) do
      Randomly select 2 parents from MATES [];
      Apply genetic operators (2-pt crossover, mutation, inversion);
      Hill-climbing by offsprings;
   endfor
   Evaluate fitness of individuals in P(gen);
   Retain the better of {fittest(gen) , fittest(gen-1)};
endfor
Solution = fittest individual
```

Fig. 1 An Outline of SGA.

A sequential hybrid genetic algorithm (SGA) has been proposed for load balancing [18], and is outlined in Figure 1. It combines a number of design choices and includes a hill climbing procedure for reducing the possibility of premature convergence and for significantly improving the efficiency of the genetic search. SGA is briefly presented here, concentrating on the constituents that are relevant to the PGA's.

The chromosomal encoding of an assignment of data to processors is given by a string of integers (allele values); an integer refers to a processor and its position in the string represents the assigned datum. The fitness of an individual in any generation is the reciprocal of the objective function defined in expression (1). The reproduction scheme is based on elitist ranking followed by random selection of parents from the list of reproduction trials allocated to the ranked individuals. In ranking, fitnesses are sorted first and reproduction trials are allocated to the individuals according to a predetermined scale of equidistant values. Elitism refers to the preservation of the fittest-so-far individual. In each generation, the fittest individual is considered a candidate solution. The genetic operators used are 2-point ring-like crossover, mutation, and inversion. The rates of these operators are

made variable in order to maintain diversity in the population. A heuristic procedure tailored to the load balancing problem is incorporated for hill climbing by individuals. In this procedure, the boundary elements of the subdomains assigned to the processors can transfer between processors. The transfer takes place only if it does not cause the fitness of the total structure to decrease. It has been found that the evolution associated with SGA goes through three stages. In the tuning stage, which is the last stage, the value of v in expression (1) is decreased in order to improve the results. Other features are also included in SGA for evading some computational costs and for reinforcing favorable aspects of the genetic search. The details and the advantages of these features are explained in [18].

4. POPULATION GENETICS AND PARALLEL GENETIC ALGORITHMS

In this section, PGA's based on models of natural evolution are presented. Important models in population genetics are briefly summarized in Subsection 4.1 as a prelude to the descriptions in Subsections 4.2 and 4.3.

4.1 Models of Population Structure

A natural population forming a species is usually spread over a large area. Hence, it does not constitute a single random mating unit, as viewed by the classic GA [12, 14], because the distance of individual movement would be much smaller than the entire distribution area of the population. The mating pool for selection is restricted to a certain range of distances and distant individuals would lie in different pools giving rise to some form of subpopulations. Associated with genetic drift, such population distribution leads to local differentiation in allele frequencies and to genetic divergence among subpopulations. Such geographic population structures can have profound effects on the evolution of species. In contrast with the case where the population is a single mating unit, variability across the populations persists and the problem of premature convergence is not encountered. Several models for population structures have been devised in population genetics. They involve various views for the subdivision of population and various schemes for intergroup selection and for genetic exchange or migration among the groups (subpopulations). Important and relevant models are summarized here. These models can be broadly divided into two categories according to whether the distribution of population is continuous or discontinuous.

Wright's island model of population structure [5] assumes that the population is large and is split into semi-isolated subpopulations or demes dispersed geographically like islands, each breeding at random within itself. Each generation, a deme exchanges a fraction of its members for migrants drawn at random from the rest of the population. If their number is not too small, the migrants can be considered representative of the subpopulations in terms of allele frequency, and incoming alleles can be assumed to be independent. The mathematical analysis for this model has shown that the coefficient of genetic differentiation is predominantly determined by the amount of migration and is independent of the mutation rate and the total number of alleles [5, 13]. The island model is not likely to be realized in nature since the immigrants usually come from adjacent demes and, thus, are not a random sample of the species. Kimura's stepping-stone models are based on the adjacency observation. These models assume certain geometrical patterns for the deme locations, such as linear arrays and rectangular grids [13]. Migration is allowed only between immediate neighbors.

The shifting balance theory of evolution [30] presents another model of discontinuous population structure. In Wright's view, this model offers a good chance for the population to avoid being hung up on a low adaptive peak and to evolve novel types of gene interactions. The shifting balance process iterates through three phases. The first phase is the random genetic drift phase, in which the allele frequencies drift to some extent and, thus, the demes explore their adaptive topography.

The second phase is for mass selection which permits the favorable gene combinations created in the first phase to rapidly become incorporated into the genome of the subpopulation by means of natural selection. Different demes now contain sets of allele frequencies that are likely to correspond, by chance, to various adaptive peaks with different heights. The third phase is for interdeme selection, where demes with higher fitness increase in size and shift the allele frequencies of adjacent demes by one-way migration until they come under the control of the higher fitness peak. The favorable genotypes become spread throughout the population in ever-widening concentric circles. In this fashion, larger parts of the adaptive topography can be explored, and a continual shifting of control from one adaptive peak to a higher one takes place. In contrast, Fisher argued against the shifting balance theory by suggesting that the adaptive peaks in multidimensional fitness landscapes are not very high and that they are connected by fairly high ridges, always shifting because of environmental changes [5]. Thus, the landscape is more analogous to waves and troughs in an ocean than to a static one.

In contrast with the above-mentioned models, Wright considered a model where the population is distributed uniformly over a large area, but interbreeding is restricted to small distances [29]. Genetic divergence within the population takes place merely due to isolation by distance. Each individual has its origin at a particular place and its parents are drawn at random from a small neighborhood. Fitter genotypes are spread throughout the population by diffusion rather than migration. The size of the neighborhood and the shape of the habitat play an important role in the analysis of the model.

4.2 Coarse-Grain PGA

The coarse-grain parallel genetic algorithm presented here is based on the shifting balance theory of evolution and is henceforth referred to as SBPGA. Previous Coarse-grain PGA's are based on other models of population structure. In [22], a PGA is presented for optimizing DeJong's functions. In this PGA, the population is split into subpopulations, and neighboring subpopulations exchange and insert into their local population the fittest individual in every generation. The distributed GA's in [26] and [3] share significant aspects with the stepping stone models. In these algorithms, subpopulations are assigned to the nodes of a hypercube, and migration occurs periodically every epoch of generations. Migrants are exchanged among all neighboring nodes. During a migration generation, subpopulations grow in size, and migrants are selected randomly in the originating subpopulations. After receiving the incoming individuals, the local population is reduced back to its original size. In [26], the PGA is used for optimizing Walsh functions, whereas in [3] it is applied to a VLSI problem.

SBPGA inherits the favorable aspects of the shifting balance model of evolution. The central aspect is the intrinsic parallelism which refers to the concurrent and independent exploration by the subpopulations, called demes, of different regions in the adaptive topography. The shifting balance model lends itself to an embarrassingly parallel decomposition, which makes it attractive for multicomputer (e.g. hypercube) implementation, because demes can be allocated to the nodes of the multicomputer and the interdeme selection is based on migration between immediate neighbors. Equally important is that the time required for the drift and mass selection phases associated with local calculations is much greater than that for the interdeme selection phase associated with interprocessor communication. The shifting balance model is more suitable for multicomputer implementation than, for example, Fisher's model since a multicomputer does represent a static environment with discontinuous locations, i.e. nodes. Furthermore, the shifting balance model has been adopted in this work because it supports a constant drive towards higher fitness peaks. Since a rapid evolution of the solution of the load balancing problem is sought, the bias towards better candi-

date solutions in the adaptive topography is convenient. It is our conjecture that although the shifting balance model may not be the most general model for natural evolution, it has advantages for artificial evolution and that it is faster than PGA's based on other models of population structure. However, the implementation of SBPGA described below deviates from the theory of shifting balance because natural evolution is slow and aims at continuously producing fitter individuals. In artificial evolution, the objective is convergence to as good a solution as possible in a reasonable time. Hence in our application, we do not want a long drift phase followed by a long interdeme selection phase in each shifting balance iteration in order to allow the fitter genotypes to spread throughout the population. Instead, the coverage of the whole population is accomplished over a number of shorter iterations.

An outline of SBPGA is presented in Figure 2 as a hypercube node algorithm. It assumes that the total population is evenly distributed as demes allocated to the nodes of a hypercube. Hence, demes and nodes become associated with each other. For example, the neighborhood of a deme is defined as the demes allocated to neighboring nodes, with physical connection one hop away. A sequential GA, such as SGA, is performed in each node for D generations as a simulation of the drift and mass selection phases of the deme's evolution for solving the load balancing problem. For this purpose, SGA can be simplified by removing features which are no longer necessary for maintaining diversity. These features include inversion and variable operator rates. Also, 1-point crossover and any acceptable selection scheme can be used instead of 2-point ring-like crossover and ranking. After a drift phase of D generations, one-way migration is carried out by allowing the demes with the higher adaptive peaks within their neighborhood to expand. Expansion is accomplished by sending copies of the M best individuals to the neighboring demes with lower peaks. It is assumed that limited resources are available for each deme and, thus, the M least fit individuals in the receiving deme are replaced by the immigrants. Then, the drift-migrate cycle is repeated.

```
Read (problem graph and multicomputer graph)
Random generation of initial deme.
Evaluate fitness of this deme.

For (DM drift-migrate cycles OR until convergence) do
  /* Drift and mass selection phases */
  For (D drift generations) do
    Perform Sequential GA
    If Tuning Stage, set D = D_tuning
  endfor
  /* 1-way migration phase (interdeme selection) */
  Find the highest fitness peak in the immediate
                neighborhood (including this deme)
  Exchange with neighbors the pair:
          (mynode, highest peak in my neighborhood)
  Save received pairs in nodelist[] , requestedlist[]
  If (mynode is in requestedlist[]) then
    Nonblocking send of copies of M migrants to
                  corresponding demes in nodelist[]
  endif
  If (mynode not contain highest peak in n' hood) then
    Blocking receive M migrants from the fittest
                          (requested) neighbor
    Replace M weakest individuals with immigrants
  endif
endfor/*drift-migrate*/
Solution = Fittest individual
```

Fig. 2 Outline of SBPGA (hypercube node algo).

The assumption of limited resources prevents any growth in the deme size and, thus, averts an increase in the implementation complexity. The length of the genetic drift phase, D, is dependent upon the deme size and the parameters of the sequential GA that affect the allele frequencies, such as the rates of the genetic operators. A good choice for D has been empirically estimated to be 0.1 to 0.2 of the maximum number of generations. The number of migrants should be big enough to force the shifting of control to higher adaptive peaks; but not too big that it swamps fit genotypes in the receiving deme. Further, it should increase for longer drift phases. An empirical estimate of 20 to 40 per cent of the deme size seems to be adequate.

4.3 Fine-Grain PGA

The fine-grain PGA described here is based upon the isolation by distance model, where the population has a continuous and uniform distribution over a large area. This model lends itself to data parallelism, which makes the Connection Machine (CM) an attractive choice for its simulation. A CM implementation of a PGA has appeared in [23] for a classifier system. However, Robertson's work is based on global selection and makes heavy use of the communication mechanisms of the CM. Another CM implementation has been independently developed for a graphics problem [15]. Its selection scheme is similar to ours; but, it does not fully exploit the massive parallelism of the CM. The CM has also been used to verify the superiority of local selection to panmictic selection [4]. The isolation by distance model has been employed in [20], [16] and [25] for solving quadratic assignment and traveling salesperson, graph partitioning, and GA-deceptive problems, respectively. These PGA's have been implemented on Transputer based systems and on a mesh-connected DAP. In [16] and [20], the global fittest individual is included in all local subpopulations during selection in order to increase the convergence speed. In this subsection, we describe an algorithm based on the isolation by distance model which exploits the massive parallelism of the Connection Machine and employs the load balancing problem as an application. The algorithm is henceforth referred to as IDPGA.

An outline of IDPGA is shown in Figure 3. The CM is configured as a 3-dimensional shape, as shown in Figure 4, where the population is distributed as follows. The number of virtual processors in the X-Y plane equals the population size. Each chromosome (individual) is distributed over a column of processors in the Z-direction, one gene per virtual processor. Each new generation is created in a distributed fashion by having each column of processors replace a parent by its offspring. In the reproduction step, the mating pool of each individual is restricted to a small local subpopulation, re-

```
Configure CM as a 3-dimensional shape
Random generation of initial population;
        one gene per processor
Read (problem graph, multicomputer graph)
Evaluate fitness

For (gen=1 to maxgen) OR until convergence do
  Set v
  Select from neighborhood {fittest OR random} in X-Y
  SPREAD in Z-direction the location of the mate
  Crossover (with mutation) the local individual
                  with the selected mate
   Hill-climbing by the resultant offspring
              (involves REDUCE comm.)
  Evaluate fitness (involves REDUCE comm.)
endfor
Solution = Fittest
```

Fig 3 Outline of IDPGA (CM implementation).

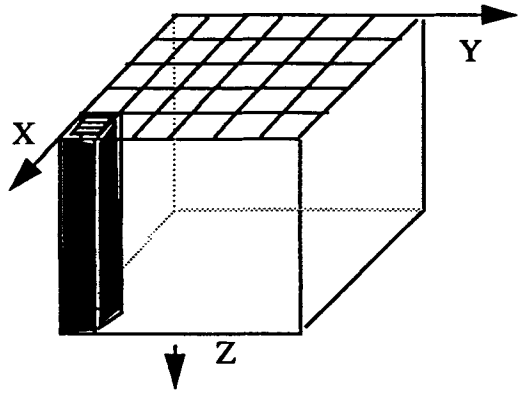


Fig. 4 CM configuration for IDPGA.
(Individual = dark column)

ferred to as neighborhood. Each individual selects a mate from its neighborhood that is either a random member or the fittest with equal probability. The local individual undergoes crossover with the selected mate and only the offspring is retained. 0.5-uniform crossover is used. It is particularly simple to implement with the allocation of genes to CM processors in IDPGA and carries no communication overhead. The second genetic operator, mutation, is a completely local operation for every gene. Then, hill-climbing and fitness evaluation are carried out for the new individuals. With the genes allocation described above, hill-climbing can no longer be performed sequentially as in

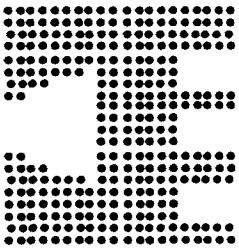


Fig. 5(a) 301-element GRID1.

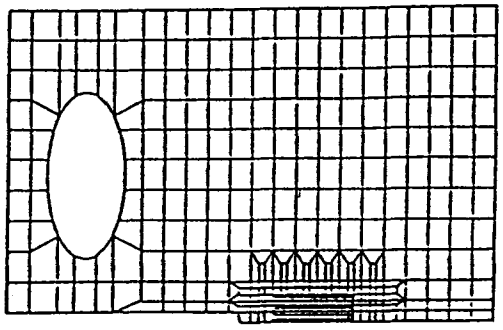
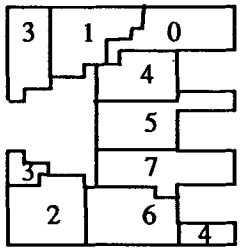


Fig. 5(b) 551-element GRID2.



Proc	Calc	Comm
0	37	15
1	36	16
2	39	10
3	39	11
4	36	19
5	38	15
6	38	15
7	38	15

Fig. 6 Best assignment of GRID1 by SBPGA, and processor loads.

SGA. In IDPGA, the boundary data elements concurrently attempt to transfer between neighboring multicomputer processors. Since the decision about a single transfer involves global terms, determined by the assignment of other data [18], concurrent hill-climbing involves errors. However, these erroneous decisions have been found not to affect the final solution, although they might have somewhat delayed the progress towards it. Interestingly, this problem, associated with concurrent hill-climbing, is identical to the problem of concurrent perturbations in parallel simulated annealing. Another important feature of IDPGA, indicated in Figure 3, is that most steps involve general CM communication operations. This is the price paid for exploiting the massive parallelism of the CM.

Clearly, subpopulations overlap in IDPGA and, thus, the fitter genotypes spread throughout the population by diffusion. The neighborhood of an individual is formed of the individuals within a certain distance in the X-Y plane. Obviously, this PGA also enjoys intrinsic parallelism. A small neighborhood size enhances the intrinsic parallelism and local differentiation and, thus, minimizes the possibility of premature convergence. However, it should not be too small otherwise it might take a long time for the search to find an acceptable solution. Another advantage of small neighborhoods is a smaller communication cost. In searching for the fittest in their neighborhoods, processors communicate along the same dimension at the same time, which reduces link contention. The choice of the fittest in the neighborhood as the second parent half the time in the reproduction step is a modification to the original isolation by distance model. Since the diffusion process is slow, this modification is justified for increasing the selection pressure and, hence, for speeding up the evolution of a solution. The better convergence caused by this modification is not expected to sacrifice the quality of the solution because of the small size of the neighborhoods, within which the fittest is sought, relative to the population size.

5. SIMULATION RESULTS

The results given below illustrate solutions for the load balancing problem and demonstrate the applicability and operation of the underlying evolutionary models. The test cases used are two irregular domains shown in Figure 5; GRID1 has irregular geometry and GRID2 is nonuniform. GRID1 and GRID2 are to be assigned to 8-node and 16-node hypercube multicomputers, respectively. These

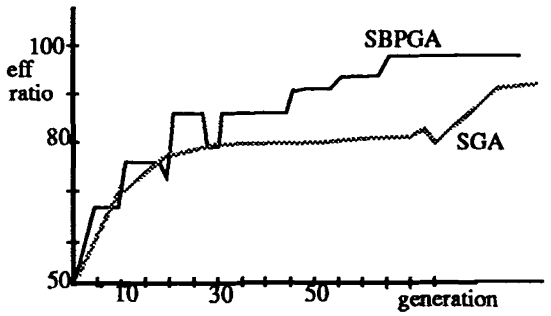


Fig. 7 Evolution of the efficiency of the assignment of GRID1 in 1 deme.

test cases have been chosen because of their irregularities and reasonable sizes, and because previous results for other algorithms, including SGA, are available for comparison. The performance measures for SBPGA and IDPGA applied to the load balancing problem are the quality of the solution and the number of evolving generations (execution time). The quality of the solution is given by the hypercube's efficiency, defined as the ratio of the sequential execution time to the product of the parallel time and the number of nodes. The evolution time is recorded in generations, in addition to seconds, so that it becomes possible to compare the evolution speed on different machines. Efficiency values are normalized with respect to a geometrically derived optimal value. The latter serves as a reference and to indicate how good the results are, since the optima are not known. The results presented here are based on 10 runs for each case, unless stated otherwise.

5.1 SBPGA Results

SBPGA has been implemented on a 16-node iPSC/2 hypercube. The required parameters are as follows. The total population size used is 320 for GRID1 and 512 for GRID2. The crossover rate is 0.7 and the mutation rate is 0.003. The number of drift generations, D, and the fraction of migrants, M, will be given below for each case. However, in the tuning stage of the search, D is halved to allow faster spreading of the genotypes produced by decreasing v in the objective function.

(a) The best assignment of GRID1 found by SBP-

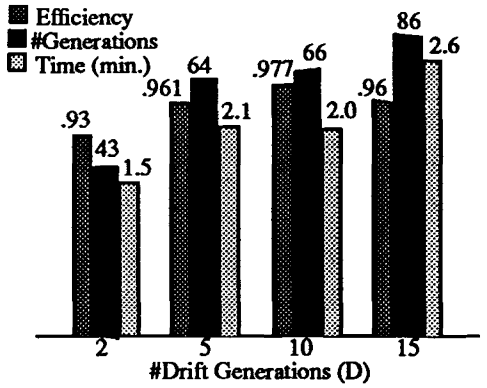


Fig. 8 Results for different lengths of drift phase (30% migration).

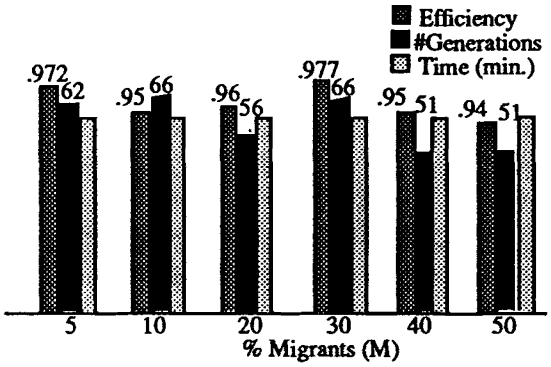


Fig. 9 Results for different percentages of migration (#drift gen=10). All time=2.0.

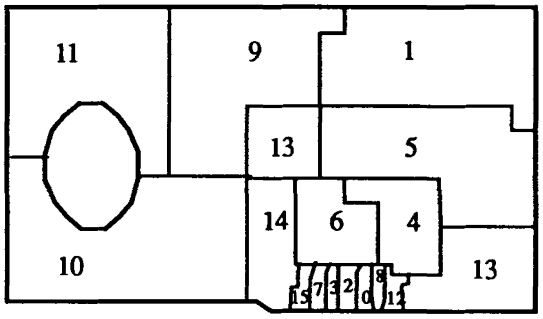


Fig. 10 Assignment of GRID2 by SBPGA.

GA is shown in Figure 6 for D=10 and M=30%. The evolution of the candidate solution in one deme is depicted in Figure 7. SBPGA finds a solution 97.7% of the geometric optimum in 66 generations, which takes about 2 minutes. The averages of 10 runs is 96% efficiency and 65 generations. The time taken by the interdeme selection phase

has been found to be 1.3% of that for the drift-mass selection phase. This makes the interprocessor communication time in the implementation of SBPGA negligible. The step improvement in the efficiency value of the candidate solution in a deme after the arrival of immigrants from the fittest neighbor is clearly manifested in Figure 7 at some points, such as generations 11 and 21. It can be seen from Figure 6 that SBPGA does not strictly insist on assigning equal number of elements to processors. Instead, it emphasizes the (approximate) balancing of the combined calculation and communication load, as required by the assumed computational model. Another feature of the solution in Figure 6 is that processors 3 and 4 are allocated discontiguous subdomains. This is not necessarily bad in our model of computation. In fact, for some problems, an optimal assignment can not be contiguous. This remark also applies to the other results below. The best solution found by SGA [18] is also shown in Figure 7 for comparison purposes; it is 97% of the optimum found in 118 generations.

(b) The efficiency of the assignment of GRID1, the number of generations required for evolving the assignment, and the time taken are illustrated in Figures 8 and 9 for different lengths of the drift phase and migration percentages, respectively. The values shown are the best of 5 runs. From Figure 8, it can be seen that if D is less than 5, the evolution model approaches that of a single mating unit and migrants increase the selection pressure, possibly leading to premature convergence. If D is greater than or equal to 15, the search becomes slow and inefficient. Therefore, D should be in the range 5 to 15 or, equivalently, 10 to 20 per cent of the maximum number of generations. In Figure 9, the results of M=5 are surprisingly good but are generally unsafe. It can be seen that if M is greater than 40% the selection pressure becomes too high, whereas a value less than 15% might not provide a sufficient shifting of control. It is concluded that 20 to 40 per cent of the deme size is a suitable range of values for M. Moreover, it is intuitive that as D decreases M should also drop to balance out the increase in the selection pressure.

(c) The best of 10 assignments of GRID2 is shown in Figure 10 for D=20 and M=30%. It corresponds to an efficiency ratio of 96% and is reached after 135 generations in about 11 minutes. The evolution of this solution is similar to that for GRID1. The time taken by migration and, thus, interprocessor communication is also negligible. In comparison, SGA has found a solution which is 93% of the optimum in 280 generations.

5.2 IDPGA Results

IDPGA has been implemented on an 8K-processor CM-2. The population sizes are the same as for SBPGA. The crossover and mutation rates are 1.0 and 0.004, respectively. The neighborhood size is chosen to be 24 unless stated otherwise. It is formed of the individuals that are within a distance of three units in the X-Y plane. The diagrams for the assignments are not included here due to limited space

(a) The best assignment found for GRID1 corresponds to an efficiency of 97.7% of the optimum and has been found in 46 generations. Each generation takes 14 seconds. The averages of 10 runs are 95.2% efficiency and 45 generations.

(b) Neighborhoods of sizes different than 24 have been experimented with. For a neighborhood size of 12 the best efficiency ratio is 96% found in 49 generations. For a larger size of 48, the best efficiency drops to 94.8% found in 42 generations. These results show that a range of sizes are suitable for the current implementation of IDPGA. However, it is intuitive that a very small neighborhood leads to longer search time, and that a very large one becomes closer to the single mating unit model and increases the communication time. Therefore, an appropriate neighborhood size is in the range of 5 to 10 percent of the population size.

(c) The best assignment of GRID2 by IDPGA corresponds to an efficiency ratio of 94.3% found in

62 generations. Each generation takes 58 seconds. The averages are 92.8% efficiency and 60 generations.

6. DISCUSSION

For the test cases considered, SBPGA and IDPGA have found better solutions than those produced by the sequential genetic algorithm [18], SGA, with identical parameter values. The results of SBPGA and IDPGA also compare favorably with those obtained by other techniques. For example, simulated annealing yields a comparable solution of 95% efficiency ratio for GRID1 [17]. Other faster methods produce lower quality solutions. In particular, a neural network [9], recursive bisection [8], scattered decomposition with patch size of 4 [19], and rectangular decomposition give 91%, 87%, 61%, and 74% efficiency for GRID1, respectively. Further, the two PGA's share the property of genetic algorithms that they do not show a bias towards particular problem configurations [17]. Hence, the favorable results for GRID1 and GRID2 are expected to extend to general configurations.

SBPGA and IDPGA exhibit superlinear speed-ups since they take a lesser number of generations than the sequential counterpart for evolving good suboptimal solutions. Thus, the PGA's are faster not only due to the parallel implementation but also due to the intrinsic parallelism of the underlying evolution models which has the potential to evolve fit genotypes faster than the single mating unit case. In comparison with other methods, the execution time of the sequential GA is comparable to that for a simulated annealing algorithm that produces comparable solutions [17]; other methods, such as those listed above, are faster at the expense of solution quality. Extrapolating this comparison of execution time to the parallel versions of these methods for hypercubes and the CM, a similar assessment might be made.

The results indicate that SBPGA and IDPGA are fairly robust. The intrinsic parallelism in the underlying evolution models provides a natural way for controlling the convergence of the evolving structures. Thus, the probability that the genetic search gets trapped in bad local optima is minimized, and the need for additional measures and parameters such as those used in SGA is obviated. The sensitivity of both PGA's to the parameter that determines the start of the tuning stage of the search is less than that of SGA, also due to the intrinsic parallelism. This insensitivity can be enhanced by using different parameter values in different subpopulations. Moreover, the results of SBPGA for different values of D and M indicate that a range of values are acceptable and an exact design value is not necessary. A similar comment applies to the neighborhood size in IDPGA.

Further work can be done to improve SBPGA and IDPGA. This would include the exploration of other CM configurations for IDPGA, corresponding to different geographic population distributions. Also, the chromosomes can be allocated to the CM columns of processors in a way that exploits the physical hardware for reducing communication cost. For example, chromosomes can be allocated to the 16-processor chips that form the nodes of the CM's cube, or contiguous segments of the chromosome can be allocated to the same physical processor. For SBPGA, asynchronous operation seems appealing. It would involve variable values for M and the use of different values for D in different demes accounted for by polling in every generation. These improvements are the subject of further research.

7. CONCLUSIONS

We have presented two PGA's, SBPGA and IDPGA, which are based on the shifting balance theory and the isolation by distance model for natural evolution, respectively. Their operation has been verified for the load balancing problem. The results of the artificial evolution demonstrate the applicability of the underlying natural evolution models. Further, the near-optimal solutions found and the speed-up obtained show the suitability of SBPGA and IDPGA as new load balancing techniques that

are faster and more robust than sequential GA's and that compare favorably to other load balancing methods.

As in evolutionary biology where different cases are simulated and explained by different models, we conjecture that the different characteristics of SBPGA and IDPGA makes the choice between them problem-dependent for general application. For example, IDPGA would be more suitable for problems that require large population sizes with simple and localized operations for the genes.

Although more applications are needed for a firm evaluation of the PGA's presented here and in other works, a preliminary assessment would support the view that "parallel computers are complex. The most complex systems have evolved in nature.... parallel computers can be better understood by models derived from natural sciences and these models can be simulated better by parallel computers." [21].

Acknowledgement

This work was supported by the Joint Tactical Fusion Program Office. The implementations were carried out using the iPSC/2 of the Theory Center at Cornell University and the CM-2 of the Northeast Parallel Architectures Center at Syracuse University. We would like to thank Tom Starmer for his comments on an earlier draft and Ernest Sibert for helpful CM implementation hints. The first author is thankful to Tom Starmer for rekindling his interest in Population Genetics.

References

1. M. Berger, and S. Bokhari, A Partitioning Strategy for Nonuniform Problems on Multiprocessors, IEEE Trans. Comp., Vol C-36,No. 5, May 1987, 570-580.

2. L. Booker, Improving Search in Genetic Algorithms, in Genetic Algorithms and Simulated Annealing, editor L. Davis, Morgan Kaufmann Publishers, 1987, 61-73.

3. J. P. Cohoon, S. U. Hedge, and W. N. Martin, Distributed Genetic Algorithms for the Floorplan Problem, IEEE Trans. CAD, April, 1991.

4. R. J. Collins and D. R. Jefferson, Selection in Massively Parallel Genetic Algorithms, ICGA'91, 249-256.

5. J. F. Crow, Basic Concepts in Population, Quantitative, and Evolutionary Genetics, Freeman, 1986.

6. F. Ercal, Heuristic Approaches to Task Allocation For Parallel Computing, Doctoral Dissertation, Ohio State University, 1988.

7. J. Flower, S. Otto, and M. Salama, A Preprocessor for Finite Element Problems, Proc. Symp Parallel Computations and Their Impact on Mechanics, ASME Winter Meeting, Dec. 1987.

8. G. C. Fox, A Graphical Approach to Load Balancing and Sparse Matrix Vector Multiplication on the Hypercube, Caltech C3P #327b, 1986.

9. G. C. Fox and W. Furmanski, Load Balancing Loosely Synchronous Problems with a Neural Network, Proc 3rd Conf. Hypercube Concurrent Computers, and Applications, 1988, 241-278.

10. G. C. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, Solving Problems on Concurrent Processors, Prentice Hall, 1988.

11. G. C. Fox, Physical Computation, Int. Conf. Parallel Computing: Achievements, Problems and Prospects, Italy, June 1990.

12. D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.

13. D. L. Hartl, and A. Clark, Principles of Population Genetics, Sinauer Associates, 1989.

14. J. H. Holland, Adaptation in Natural and Artificial Systems, Univ. of Michigan Press, 1975.

15. C. Kosak, J. Marks, and S. Shieber, A Parallel Genetic Algorithm for Network-Diagram Layout, ICGA'91, 458-465.

16. G. Laszewski, Intelligent Structural Operators for the k-way Graph Partitioning Problem, ICGA'91, 45-52.

17. N. Mansour and G.C. Fox, Physical Optimization Methods for Allocating Data to Multicomputer Nodes,

Syracuse Center for Computational Science, SCCS-122, 1991.

18. N. Mansour and G.C. Fox, A Hybrid Genetic Algorithm for Task Allocation in Multicomputers, ICGA'91, 466-473.

19. R. Morison and S. Otto, The Scattered Decomposition for Finite Elements, J. Scientific Computing, vol 2, no. 1, Mar. 1987, 59-76.

20. H. Muhlenbein, Parallel Genetic Algorithms, Population Genetics, and Combinatorial Optimization, ICGA'89, 416-421.

21. H. Muhlenbein, M. Gorges-Schleuter, and O. Kramer, New Solutions to the Mapping Problem of Parallel Systems: The Evolution Approach, Parallel Computing 4, 1987, 269-279.

22. C. Pettey, M. Leuze, and J. Grefenstette, A Parallel Genetic Algorithm, ICGA'87, 155-161.

23. G. Robertson, Parallel Implementation of Genetic Algorithms in a Classifier System, in Genetic Algorithms and Simulated Annealing, editor L. Davis, Morgan Kaufmann Publishers, 1987, 129-140.

24. H. Simon, Partitioning of Unstructured Mesh Problems for Parallel Processing, Proc. Conf. Parallel Methods on Large Scale Structural Analysis and Physics Applications, Permagon Press, 1991.

25. P. Spiessens and B. Manderick, A Massively Parallel Genetic Algorithm, ICGA'91, 279-287.

26. R. Tanese, Distributed Genetic Algorithms, ICGA'89, 434-440.

27. D. Whitley, Workshop on Genetic Algorithms and Neural Networks, ICGA'91.

28. R. D. Williams. Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations. Submitted to Concurrency Practice and Experience, 1990.

29. S. Wright, Isolation by Distance, Genetics 28 : 114, March 1943, 114-137.

30. S. Wright, Evolution and the Genetics of Populations, Vol. 3, Univ. of Chicago Press, 1977.