Scalable Parallel Genetic Algorithms

Wilson Rivera *
Electrical and Computer Engineering Department, University of Puerto Rico at Mayaguez

November 9, 2000

Abstract. Genetic algorithms, search algorithms based on the genetic processes observed in natural evolution, have been used to solve difficult problems in many different disciplines. When applied to very large-scale problems, genetic algorithms exhibit high computational cost and degradation of the quality of the solutions because of the increased complexity. One of the most relevant research trends in genetic algorithms is the implementation of parallel genetic algorithms with the goal of obtaining quality of solutions efficiently. This paper first reviews the state-of-the-art in parallel genetic algorithms. Parallelization strategies and emerging implementations are reviewed and relevant results are discussed. Second, this paper discusses important issues regarding scalability of parallel genetic algorithms.

Keywords: parallel genetic algorithms, coarse grained implementation, fine grained implementation, parallel systems, scalability metrics, cost efficiency

1. Introduction

Genetic Algorithms (GAs) are computational models of evolutionary processes observed in nature, which have been applied to a wide spectrum of problems. For example, Caldwell and Johnston (1991), Fourman (1985), Maini et al. (1994), Jin et al. (1997), and Rivera (1998). Theoretical concepts were first introduced by Holland (1975), and later described by Golberg (1989).

As complexity of the applications increases, GAs exhibit high computational cost and degradation of the quality of the solutions. Efforts for solving these shortcomings have been developed in several directions, and parallel GAs (PGAs) is one of the most significant. However, in order to guarantee the effective use of the parallel resources, it is necessary to ensure high scalability.

Scalability measures the ability of a parallel machine and a parallel algorithm to use efficiently a larger number of processors. Thus, since scalability depends on both the communication patterns of the algorithm and the infrastructure provided by the machine, a good measure of scalability should at least adequately reflect the interaction between

* University of Puerto Rico, P. O. Box 9042, Mayaguez PR 00681-9042, USA. email:wrivera@ece.uprm.edu

these two aspects. Moreover, a scalability metric should not only indicate if a parallel system is scalable, but also provide information regarding the specific conditions required to achieve high performance.

Scalable PGAs are an important alternative for solving large-scale problems. This paper explores the state-of-the-art in PGAs, and studies the scalability of PGAs. Section 2 outlines GAs and describes different parallelization strategies. Section 3 reviews some relevant results. Section 4 discusses scalability of parallel systems and scalability metrics. Section 5 deals with the performance of PGAs. Finally, section 6 presents conclusions.

2. Parallel Genetic Algorithms

In this section a brief description on GAs is presented and different parallelization strategies are discussed. Detailed background information on GAs can be found in Davis (1991), Golberg (1989), and Mitchell (1997). A complete indexed bibliography of PGAs can be found in Alander (1999).

2.1. Genetic algorithms

GAs start with an initial population of individuals, which is generated either randomly or with domain specific knowledge. Each individual *(chromosome)* consists of a data structure, and represents a possible solution in the search space of the problem. Usually, a problem-specific *fitness function* maps the representation of the chromosome into a *fitness value*. The fitness value measures the quality of the individual as an optimal solution.

GAs evolve into new generations of individuals by using knowledge from previous generations. The fundamental principle of GAs is that chromosomes which include blocks of genetic information that are contained in the optimal solution will increase in frequency if the opportunity of reproduction of each chromosome is related, in some way, to its fitness value. Thus, GAs are both explorative and exploitative methods for solving problems that are not affordable by traditional methods.

A typical example occurs when a potential solution of a problem may be represented as a set of parameters, which in their turn are represented by strings of characters. The parameters are put together to form a longer string of characters which will be the chromosome structure. In order to create a new population, pairs of individual are selected, based on their fitness values, and recombined using a crossover

operator. The crossover is controlled by a crossover probability parameter. For example, in the *one-point crossover*, a biased coin is tossed which comes up true with the specific crossover probability. Next, a crossover point is randomly chosen. Then, both selected chromosomes are cut at the crossover point, and the chromosome parts of the individuals are swapped creating two new offspring. In addition, a mutation operator is applied such that bits in the chromosomes are changed with a certain mutation probability. The idea behind mutation is to enhance diversity among the population so that premature convergence and suboptimality are avoided.

The performance of a GA is subject to stochastic errors. Hence, because the population is not infinite, at early iterations, high fitness individuals may dominate the population, so that the GA converges to a wrong solution. Usually, mutation contributes to solve this problem.

Thus, a new population is generated by using *genetic operators* (selection, crossover, and mutation). The fitness values of the new individuals are evaluated, and the process is repeated until a termination condition is satisfied.

2.2. Parallelization strategies

In general, PGAs exhibit a different behavior compared to the standard GA. As a consequence, PGAs constitute a novel class of complex algorithms for which further research is necessary. We identify four possible strategies to parallelize GAs.

1. A *global parallelization* in which only the evaluation of individuals' fitness values is parallelized by assigning a fraction of the population to each processor to be evaluated. Also, the genetic operators can be parallelized within this scheme. However, in most cases it is not efficient because the genetic operators are not as time-consuming as the fitness evaluation. A master processor performs the genetic operators and distributes the individuals among a set of slave processors that evaluate the fitness values (Figure **??**). This strategy preserves the behavior of the original GA because there is only one population just as in the sequential case. This method of parallelization is particularly effective for complicated fitness evaluation, and improvements in the computational time with regard to the sequential GA can be expected.

2. A *coarse-grained parallelization* in which the entire population is partitioned into subpopulations (*demes*). A GA is run on each subpopulation, and exchange of information between demes (*migration*) is performed eventually. The critical issue is how the migration

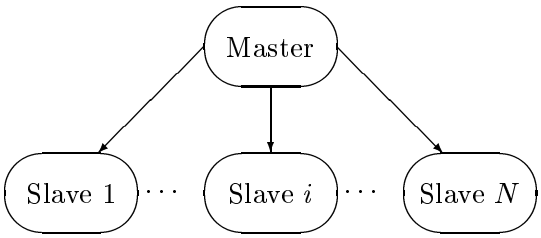

Figure 1. global parallelization

among demes should be implemented. For example, the event that causes the migration, the number of individuals that migrate, and the communication topology are important factors to be considered in this strategy. Each one of the factors before mentioned is associated with a control parameter. As an example, Figure **??** shows a coarse-grained parallelization on an unidirectional ring network with N processors.

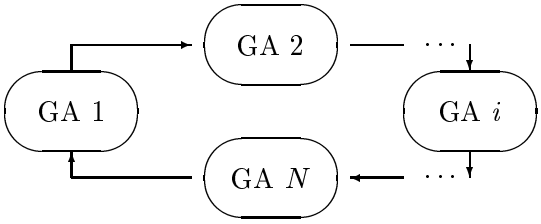

Figure 2. Coarse-grained parallelization on an unidirectional ring

3. A *fine-grained parallelization* in which exactly one individual is assigned to each processor. The genetic operators take place in parallel only among adjacent processors, and the individual in each processor is replaced by the new offspring as new generations come out. Hence, the topology of the network strongly determines the behavior of the GA. The local nature of the genetic operators allows for a natural diversity in many applications. Usually, fine-grained parallelization is implemented on massively parallel machines (Figure **??**).

4. A *hybrid parallelization* in which several parallelization approaches are combined. The complexity of these hybrid PGAs depends on the level of hybridization.

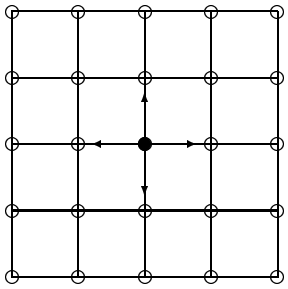

Figure 3. Fine-grained parallelization on a mesh

The different alternatives to parallelize GAs have resulted in a varied number of implementations of PGAs. Table **??** outlines some of the most important PGAs currently available.

Table I. Parallel GAs software

PGA	**Parallelism**	**Topology**
PGAPack	global	-
EnGENEer	global	-
GENITOR II	coarse	ring
DGENESIS	coarse	any
GALOPPS	coarse	any
iiGA	coarse	any
ASPARAGOS	fine	ladder
ASPARAGOS96	hybrid	ring

PGAPack (Levine, 1995) is a PGA library that supports global parallelization by a master/slave model. This package is written in C and communication is carried out using MPI. PGAPack allows for multiple levels of control. At the first level, the PGA is encapsulated so that the user only has to specify the data type, the string length, and the optimization type. At the next level, the user has more control on the genetic operators. At the high level, the user can define functions and operators within an object oriented frame. In general, the computational cost is reduced with this PGA implementation. However, the speedup that can be achieved with the master/slave model is limited by the amount of computation associated with the fitness evaluation that can be executed in parallel.

ASPARAGOS (ASynchronous PARAllel Genetic Otimization Strategy; Gorges-Scheleuter, 1989) is a fine-grained PGA that uses a ladder

topology with connected ties to communicate the individuals. This package incorporates the idea of isolation-by-distance by using a topologically structured population where the interaction among individuals is local. The new version, ASPARAGOS96 (Gorges-Scheleuter, 1997), uses a ring structure and supports hierarchy of subpopulations. The ring structure allows for a higher local differentiation compared to the ladder topology, and in addition, the population can be reduced without degradation of quality. ASPARAGOS96 is indeed a hybridized implementation because migration among subpopulations is permitted. When one of the subpopulation converges, it receives the best individual from the local subpopulations.

DGENESIS (Distributed GENESIS; Mejia-Olivera & Cantu-Paz, 1994), based on GENESIS 5.0 (Grefenstette, 1990), is a coarse-grained implementation where each subpopulation is controlled by an UNIX process and communication is carried out using Berkely sockets. DGENESIS allows for different combinations of topologies, migration rates, migration events, and genetic operators.

iiGAS (Injection Island GAs; Lin et al., 1994) uses hierarchical heterogeneous subpopulations with asynchronous migrations among demes. Each subpopulation encodes the problem using a different resolution, and evolves under different parameters and genetic operators. Migration occurs from low resolution nodes to high resolution nodes. When a subpopulation converges, it injects its best individual to the neighboring high resolution nodes. Lin et al. (1994) showed that better results may be obtained without speed degradation by using a dynamic topology based on population similarity instead of a static topology.

GENITOR II (Whitley & Starkweather, 1990) is a modular steady-state PGA package with ranked selection and several order based crossover operators. GALOPPS (Genetic ALgorithm Optimized for Portability and Parallelism; Goodman, 1996) is a coarse-grained implementation coded in C and PVM for communicating. EnGENEer (Schraudolp & Grefenstette, 1991) is a commercial software written in C that runs under Unix. EnGENEer has a high level of abstraction language that permit the user to define the structure of the genetic problem within an interactive interface.

3. Relevant Advances

Regarding coarse-grained parallelization, researchers have intended to answer questions about how often migration should be performed (*migration interval*), how many individuals may migrate (*migration rate*), and which is the most efficient communication network (*connectivity*).

Early studies by Grosso (1985) and Tanese (1989) showed that for low migration rates, the migration of individuals does not have any significant effect on the performance of the PGA. Moreover, there is evidence regarding the existence of a critical migration rate below which migration is not effective. For example, Tanese's experiments with isolated subpopulations showed that such a coarse-grained PGA could find solutions of the same quality as the traditional GA at some point during the searching, but the average fitness of the population at the last generation was degraded. On the other hand, when migration was allowed the average fitness was systematically improved.

Similar results for migration rate were obtained by Starkweather et al. (1991). In their coarse-grained PGA they proposed a sort of adaptive mutation was implemented in which the probability mutation was augmented as the similarity between parents was increased. Thus, the premature convergence on the subpopulations, which is critical for a large number of demes, was overcame.

Results in Lin et al. (1994) and Hart et al. (1996) showed that asynchronous implementations exhibit a reduced function evaluation, and low overhead and cost compared to synchronous implementations. For relaxed synchronous PGAs, the number of function evaluations decreases as well as communication overhead and cost.

Hart et al. (1996) considered the effect of relaxed synchronization on the performance of a coarse-grained topologically structured PGA. Their implementation uses a toroidal, two-dimensional population that is distributed among the available processors. Each processor communicates with a fixed number of neighbors. In the selection process individuals interact with a fixed subset of the population that partially overlaps with subsets used by other individuals. Communication is required for selection and crossover operators, and to stop all the processors when one of the subpopulations converges. The synchronization guarantees that each processor has already communicated with its neighbors before initiating the next iteration.

Other authors who investigated the effect of migration rate include Cohoon et al. (1987), Braun (1990), Rebaudengo & Reorda (1993), Belding (1995), Bianchini et al. (1995), and East & Rowe (1996).

Gordon et al. (1992) showed that the critical path of a fine-grained PGA is shorter than that of a coarse-grained PGA, demonstrating the suitability of fine-grained implementation for massively parallel machines. Gordon & Whitley (1993) compared different models of PGAs applied to optimization problems. For the problems that they were considering, coarse-grained implementations showed a better performance. Gordon (1994) studied spatial locality of memory references in different models of PGAs.

Gorges-Schleuter (1991) proposed a fine-grained PGA with elitist strategy such that a new offspring was accepted only if it obtained a better fitness value than the local parent. The PGA was implemented on a sparse graph topology.

Results from Shapiro & Navetta (1994) and Sarma & Jong (1996) showed how changes in the size and topology of the neighborhood for fine-grained implementations affect the quality of the solutions in the selection process.

Theoretical analysis of PGAs is difficult because of the strong interaction of the parameters involved in parallel implementations. A recent analysis by Cantú-Paz (1998) used Markov chains to model the behavior of a coarse-grained PGA. In this work it is assumed that migration occurs only after each subpopulation converges. The analysis is based on the prediction of the quality of the solutions after an arbitrary number of generations. Cantú-Paz ensures that for this specific migration event the major improvement in quality occurs at the first two epochs (intervals between migrations). Thus, only the first two epochs are considered for the analysis. Interesting results can be extracted from this work. First, the quality of the solutions improves with higher migration rates. The communication cost will be independent of the migration rate. Migration only occurs when a subpopulation converges so that just one individual is sent to the neighboring subdomains, and then it can be replicated according to the migration rate. Second, partitioning the population does not result in a degradation of quality of the solutions as long as the migration rate is not very low, and only a few epochs are necessary to achieve the same solution as the sequential GA. Third, increasing the connectivity improves the quality of the solutions, but a high degree of connectivity increases the communication cost.

The main deficiency of this analysis is the lack of generality. The model corresponds to a very particular migration event. A more general approach requires that performance be expressed in terms of something more than just the quality of the solutions.

Earlier, Suzuki (1993) had already used Markov chains to analyze the performance of sequential genetic algorithms. Muhammad et al. (1997) developed a fine-grained PGA based on a Markov chain. The implementation used an one-dimensional toroidal topology. The transition rules for selection, crossover and mutation operators are integrated into a Markov transition rule to model the fine-grained PGA. From the resultant transition matrix, the mutation rate emerges as the control parameter for the Markov chain that models the fine-grained PGA. Muhammad et al. demonstrated the convergence of the fine-grained PGA to a stationary distribution when the mutation rate is held constant. The convergence to a stationary distribution does guar-

antee the asymptotic convergence properties, but it does not assure the convergence to a global solution.

4. Scalability of Parallel Systems

A parallel system is defined as the combination of a parallel algorithm and a parallel machine. Intuitively, scalability is the measure of the ability of a parallel system to effectively utilize an increasing number of processors. By increasing the number of processors, the efficiency provided by an architecture for a given algorithm decreases because of the need of synchronization and communication. Amdahl (1967), for example, pointed out that when the problem size is fixed, speedup is upper bounded by the reciprocal of the sequential part of the algorithms without regard to the number of processors used. Actually, for fixed size problems, speedup is limited because of the overhead, which grows with increasing number of processors, or because the number of processors exceeds the degree of concurrence of the algorithm, that is, the maximum number of tasks which can be executed simultaneously. As a consequence, the problem size should be increased in order to achieve high performance.

4.1. Isoefficiency

Grama et al. (1993) and Gupta & Kumar (1993) stated that the relation between the problem size and the maximum number of processors that can be utilized in a cost optimal fashion for solving a problem is given by the isoefficiency function. They define problem size W as the number of basic operations required by the fastest serial algorithm to solve a given problem. $T_s(W) = \Theta(W)$ is the sequential execution time of the best serial algorithm, and $T_p(W, p)$ is the parallel execution time of the algorithm on p processors. The cost of a parallel system is defined by the product pT_p, and a parallel system is cost-optimal if and only if the cost is asymptotically of the same order of magnitude as T_s, that is, $pT_p = \Theta(W)$.

The overhead due to communication cost and synchronization can be written as

$$T_o(W, p) = pT_p + W. \tag{1}$$

Thus, efficiency, which is the ratio between the sequential execution time and the cost of the parallel system, is given by

$$E = \frac{T_s}{pT_p} = \frac{W}{W + T_o(W,p)} = \frac{1}{1 + \frac{T_o(W,p)}{W}}. \tag{2}$$

If the system size p increases, then E decreases because $T_o(W,p)$ increases with p. On the other hand, if W is increased, with a fixed p, then E increases because $T_o(W,p)$ grows slower than W. Hence, the efficiency may be maintained constant by increasing W as p increases. This relation is referred to as isoefficiency function.

Manipulating algebraically the equation (2), we can obtain the isoefficiency function

$$W = KT_o(W,p), \quad \text{where} \quad K = \frac{E}{1-E}. \tag{3}$$

A parallel system is scalable if and only if its isoefficiency function exists. If W needs to grow exponentially with respect to p, the parallel system is poorly scalable because it is difficult to hold the efficiency constant unless W is enormous. On the contrary, if W grows nearly linear with p, the parallel system is highly scalable.

4.2. Effectiveness

Additional factors regarding the applications that use the parallel system must be considered in order to get a realistic metric to measure scalability.

Singh et al. (1993) proposed a methodology to evaluate the scaling of parallel systems regarding application parameters. First, it is necessary to understand the relationships among the application parameters in terms of their error contribution. Second, an equal-error principle is used to generate scaling rules for the parameters. The equal-error principle states that the sources of error contribute approximately in similar quantities to the overall error. Finally, the impact of the scaling rules on the effectiveness and architectural design parameters is evaluated.

Luke et al. (1997) proposed a metric based on the relation between cost and performance of parallel systems, where the methodology presented in Singh et al. (1993) is implicit into the concept of scaling path. Let A be a set of algorithms to solve a given application, and $W(\alpha)$ the number of operations required by an algorithm α to solve the problem. Work is defined as the minimum number of operations required to solve the application, that is

$$\mathcal{W} = \min_{\alpha \in A} W(\alpha). \tag{4}$$

The work needed to perform an application depends on certain application parameters $x_1, x_2, \ldots, x_n$. A scaling path ξ describes the application parameters such that $\mathcal{W}(x_1(\xi), x_2(\xi), \ldots, x_n(\xi))$ increases as ξ increases. Thus, we have that

$$\begin{aligned} Performance &= \frac{\mathcal{W}(\xi)}{T_p(\alpha, H, p, \xi)}. \\ Cost &= pT_p(\alpha, H, p, \xi). \end{aligned} \tag{5}$$

Notice that while work depends on the application parameters, the parallel execution time depends on the application parameters as well as the algorithm, the architecture, and the system size.

Luke et al. (1997) defined optimal effectiveness as

$$\Gamma_{opt} = \max_p \{\frac{\mathcal{W}(\xi)}{pT_p^2(\alpha, H, p, \xi)}\}. \tag{6}$$

This metric measures the maximum potential for a parallel system to deliver cost effective performance involving the parameters of a particular application.

5. Performance of a Coarse-grained Implementation

In the second part of this survey we discuss the performance of PGAs in a specific context. However, the discussion can be extended to general implementations. A coarse-grained parallel version of the algorithm presented in Rivera (1999) is used for the discussion. The implementation has been tested on a SGI Origin 2000 with 64 processors. In our discussion the problem size $\mathcal{W}$ is the required percentage of good solutions, p the number of processors, α the migration rate, and β the connectivity of the topology. Thus, if $\alpha = 0.3$ and $\beta = 2$, then 30% of the population in a deme migrates to two neighbors processors. Table **??** shows the effectiveness vales (normalized by a factor of 10^5) that are obtained for a total population of 10.000 individuals. The parallel time t_p is measured in seconds, and optimal effectiveness values for each combination of parameters are highlighted. The maximum effectiveness gives us the best combination of parameters that delivers cost-effective results for the specific application.

Next, as shown in Figure **??**, we compare two different combinations of parameters: $(A) = \{\alpha = 0.7, \beta = 8\}$ and $(B) = \{\alpha = 0.3, \beta = 16\}$. It

can be observed that for small values of $\mathcal{W}$ combination (B) performs better than combination (A), and vice versa for large values of $\mathcal{W}$. Table **??** shows estimates of k_Γ and q such that

$$\Gamma_{opt} = k_\Gamma \mathcal{W}^q. \tag{7}$$

Thus, a power law approximation can be used to characterize the behavior of the PGA. In this particular case, the combination (A) with $q \approx -0.3$ is found to be more scalable than the combination (B) when a high degree of precision is required. In addition, it can be noted that the optimal number of processors can be also modeled by a power law of the form

$$p_{opt} = k_p \mathcal{W}^r. \tag{8}$$

For example, for the combination (A) we obtain $k_p \approx 35.00$ and $r \approx 0.7$. Figure **??** outlines the behavior of the PGA for the combination of parameters (A). As pointed before, a similar analysis can be carried out for general PGA implementations.

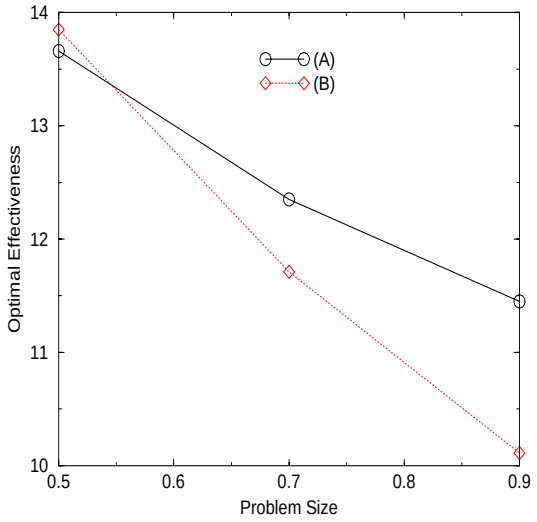

Figure 4. Scaling of $\mathcal{W}$

6. Conclusions and Further Research

Relevant issues concerning PGAs have been discussed in an effort to review the current state-of-the-art. Parallelization strategies have been depicted and some of the most important implementations currently available have been pointed out.

Table II. Effectiveness values

$\mathcal{W}$	α	β	$p = 4$	$p = 8$	$p = 16$	$p = 32$
50%	0.3	4	9.71	10.05	9.36	8.91
		8		12.01	10.31	10.35
		16			**13.85**	11.91
	0.5	4	9.51	11.01	9.76	8.71
		8		**12.05**	11.35	10.51
		16			9.81	9.36
	0.7	4	8.31	9.81	11.01	10.12
		8		10.01	**13.66**	11.03
		16			10.91	9.01
70%	0.3	4	6.91	8.72	7.51	6.91
		8		10.02	9.30	8.00
		16			**11.75**	10.61
	0.5	4	8.61	7.62	6.91	5.96
		8		**8.35**	7.62	6.87
		16			8.31	7.93
	0.7	4	6.32	8.35	10.31	9.15
		8		9.71	**12.35**	10.91
		16			9.81	10.15
90%	0.3	4	8.51	9.35	8.90	8.98
		8		9.60	9.83	**10.11**
		16			8.91	9.15
	0.5	4	7.15	7.32	6.41	6.95
		8		8.31	7.96	**8.96**
		16			7.61	8.15
	0.7	4	5.41	7.61	7.65	10.98
		8		8.71	10.01	**11.45**
		16			8.31	9.36

In this paper, we have discussed how to configure PGAs that will deliver cost-efficiency performance. It has been shown that the best combination of control parameters, which allows for an effective use of an increasing number of processors, can be identified by using the maximum effectiveness values. In addition, our experiments showed that a power law approximation can be used to characterize the behavior of a particular PGA. This power law can also be used to derive

Table III. Estimates of k_Γ and q

Parameters	k_Γ	q
(A)	$11.01e-0.5$	-0.3
(B)	$9.80e-0.5$	-0.7

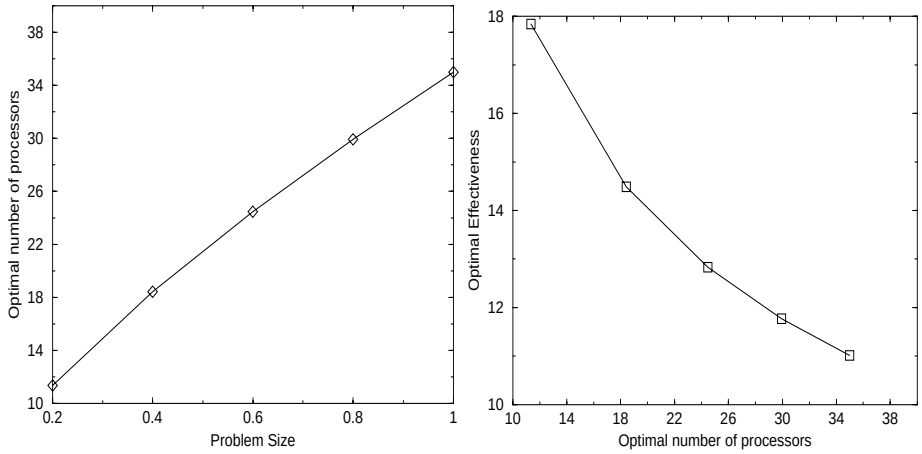

Figure 5. Scaling for (A)=$\{\alpha = 0.7, \beta = 8\}$

simple heuristics for identifying crossover points where a particular set of parameters becomes more effective than another.

Scalability analysis is a valuable tool for designing PGAs that combine different topologies, migration events, and hybridization. Currently, we are incorporating scalability functions into an object-oriented environment that will provide a framework of classes for a typical PGA. Our work in progress indicates that this approach enables easy development of high performance PGAs with the feature of choosing among several alternatives depending on the problem and machine parameters.

References

Alander J.T. (1999). Indexed Bibliograhy of Distributed Genetic Algorithms. Report 94-1-PARA, University of Vaasa, Department of Information Technology and Production Economics. Available via *ftp://ftp.uwasa.fi/cs/report94-1/gaPARAbib.ps.Z*

Amdahl G.M. (1967). The Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In Proceedings of *The AFIPS Conference,* 483-485.

Belding T.C. (1995). The Distributed Genetic Algorithm Revisited. In Proceedings of *The Sixth International Conference on Genetic Algorithms,* 114-121, San Mateo, CA: Morgan Kaufmann.

Bianchini R., Brown C.M., Cierniak M., & Meira W. (1995). Combining Distributed Populations and Periodic Centralized Selections in Coarse-grain Parallel Genetic Algorithms. *Artificial Neural Nets and Genetic Algorithms,* 483-486, New York: Springer-Verlag.

Braun H.C. (1990). On Solving Travelling Salesman Problems by Genetic Algorithms. In Schwefel H & Manner R. (Eds.) *Parallel Problem Solving from Nature,* 129-133. Berlin: Springer-Verlag.

Caldwell C. & Johnston V.S. (1991). Tracking a Criminal Suspect Through "Face-Space" with a Genetic Algorithm. In Proceedings of *The Fourth International Conference on Genetic Algorithms,* 416-421.

Cantú-Paz E. (1998). *A Markov Chain Analysis of Parallel Genetic Algorithms with Arbitrary Topologies and Migration Rates.* IlliGAL Report No. 98010, University of illinois at Urbana-Champaign, Urbana, IL.

Cohoon J.P., Martin W.N., & Richards D.S. (1987). Punctuated Equilibria: A Parallel Genetic Algorithm. In Proceedings of *The Second International Conference on Genetic Algorithms,* 148-154. Hillsdale, NJ: Lawrence Erlbaum Associates.

Davis L.D. (1991). *The Handbook of Genetic Algorithms.* Van Nostrand Reinhold: New York, NY.

East I.R. & Rowe J. (1996). Effects of Isolation in a Distributed Population Genetic Algorithm. *Parallel Problem Solving from Nature,* 408-419, Berlin: Springer-Verlag.

Fourman M.P. (1985). Compaction of Symbolic Layout Using Genetic Algorithms. In Proceedings of *the First International Conference on Genetic Algorithms,* 141-153.

Goldberg D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley: New York, NY.

Goodman E.D.(1996). *An Introduction to GALOPPS v3.2,* TR 96-07-01, GARAGe, I. S. Lab., Michigan State University. Available at *ftp:// isl.msu.edu/pub/GA*

Gordon V.S., Whitley D., & Bö hm A. (1992). Dataflow parallelism in genetic algorithms. In Manner R. & Manderick D. (Eds.), *Parallel Problem Solving from nature, 2,* 533-542. Amsterdam: Elsevier Science.

Gordon V.S. & Whitley D. (1993). Serial and Parallel Genetic Algorithms as Function Optimizers. In proceedings of *The Fifth International Conference on Genetic Algorithms.* 177-183, San Mateo, CA: Morgan Kaufmann.

Gordon V.S. (1994). Locality in Genetic Algorithms. In proceedings of The First IEEE Conference on Evolutionary Computation, 428-432, Piscataway, NJ: IEEE Service Center.

Gorges-Schleuter M. (1989). ASPARAGOS an Asynchronous Parallel Genetic Optimization Strategy. In Proceedings of *The Third International Conference on Genetic Algorithms,* 422-427.

Gorges-Schleuter M. (1991). Explicit Parallelism of Genetic Algorithms Trough Population Structures. In Schwefel H & Manner R. (Eds.) *Parallel Problem Solving from Nature,* 150-159. New York: Springer-Verlag.

Gorges-Schleuter M. (1997). Asparagos96 and the Traveling Salesman Problem. In Proceedings of *The Fourth International Conference on Evolutionary Computation,* 171-174, IEEE Press.

Grama A., Gupta A., & Kumar V. (1993). Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures. *IEEE Parallel and Distributed Technology,* **1**(3): 12-21.

Grefenstette J.J. (1990).*A User Guide to GENESIS Version 5.0,* Navy Center for Applied Research in Artificial Intelligence, Washington D.C.

Grosso P.B. (1985). *Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model,* Ph.D. diss., University of Michigan, Ann Arbor, MI.

Gupta A. & Kumar V. (1993). Performance Properties of Large Scale Parallel Systems. *Journal of Parallel and Distributed Computing,* **19**: 234-244.

Hart W., Baden S., Belew R.K., & Kohn S. (1996). Analysis of the Numerical Effects of Parallelism on a Parallel Genetic Algorithm. In Proceedings of *The tenth International Parallel Processing Symposium,* 606-612.

Holland J.H. (1975). *Adaptation of Natural and Artificial Systems.* University of Michigan Press: Ann Arbor, MI.

Jin A.Y., Leung F.Y., & Weaver D.F. (1997). Development of a Novel Genetic Algorithm Search Method (GAP 1.0) for Exploring Peptide Conformational Space. *Journal of Computational Chemistry,* **18**(16): 1971-1984.

Levine D. (1995). *Users Guide to the PGAPack Parallel Genetic Algorithm Library,* ANL-95-18. Available at *http://www.mcs.anl.gov/pgapack.html.*

Lin S.C., Punch W.F., &. Goodman (1994). Coarse-grained Parallel Genetic Algorithms: Categorization and New Approach. In Proceedings of *The Sixth IEEE Symposium on Parallel and Distributed Processing,* Los Alamitos, CA: IEEE Computer Society Press.

Luke E.A., Banicescu I., & Li J. (1997). *The Optimal Effectiveness Metric for Parallel Application Analysis,* Technical Report MSU-EIRS-ERC-97-6.

Maini H., Mehrotra K., Mohan C., & Ranka S. (1994). Genetic algorithms for Graph Partitioning and Incremental Graph Partitioning. In Proceedings of *The IEEE Supercomputing Conference.*

Mejía-Olivera M. & Cantú-Paz E. (1994). DGENESIS-Software for the Execution of Distributed Genetic Algorithms. In Proceedings of the XX Conferencia Latinoamericana de Informática, 935-946, Available at *ftp://ftp.aic.nrl.navy.mil*

Mitchell M. (1997). *An Introduction to Genetic Algorithms.* MIT Press: Cambridge, MA.

Muhammad A., Bargiela A., & King G. (1997). Fine-grained Parallel Genetic Algorithm: A Stochastic Optimisation Method. In Proceedings of *The First World Congress on Systems Simulation,* 199-203.

Rebaudengo M. & Reorda M.S. (1993). An Experimental Analysis of the Effects of Migration in Parallel Genetic Algorithms. In proceedings of *The Euromicro Workshop on Parallel and Distributed Processing,* 232-238, Los Alamitos, CA: IEEE Computer Society Press.

Rivera-Gallego W. (1998). A Genetic Algorithm for Circulant Euclidean Distance Matrices. *Journal of Applied Mathematics and Computing,* **97**(2-3): 197-208.

Rivera-Gallego W. (1999). A Genetic Algorithm for Solving the Euclidean Distance Matrices Completion Problem. In Proceedings of *The ACM Symposium on Applied Computing,* 286-290.

Sarma J., & Jong K.D. (1996). An Analysis of the Effects of Neighborhood Size and Shape on Local Selection Algorithms. In *Parallel Problem Solving from Nature IV,* 236-244. Berling: Springer-Verlag.

Schraudolp N.N. & Grefenstette J.J. (1991). *A User's Guide to GAUCSD 1.2*, Technical Report Computer Science and Engineering Department, University of California, San Diego, CA.

Shapiro B., & Navetta J.(1994). A Massively Parallel Genetic Algorithm for RNA Secondary Structure Prediction. *Journal of Supercomputer,* **8**, 195-207.

Singh J.P., Hennessy J.L., & Gupta A. (1993). Scaling Parallel Programs for Multiprocessors: Methodology and Examples. *IEEE Computer,* **26**(7): 42-50.

Starkweather T., Whitley D., & Mathias K. (1991). Optimization Using Distributed Genetic Algorithms. In Schwefel H & Manner R. (Eds.) *Parallel Problem Solving from Nature,* 176-185. New York: Springer-Verlag.

Suzuki J. (1993). A Markov Chain Analysis on a Genetic Algorithm, In Proceedings of *The International Conference on Genetic Algorithms and Applications.*

Tanese R. (1989). Distributed Genetic Algorithms. In Proceedings of *The Second International Conference on genetic Algorithms,* 434-439.

Whitley D. & Starkweather T. (1990). GENITOR II: A Distributed Genetic Algorithm. *Journal of Experimental, Theoretical and Artificial Intelligence,* **2**, 189-214.