Parallel Computing 17 (1991) 619–632
North-Holland

The parallel genetic algorithm as function optimizer

H. Mühlenbein [a], M. Schomisch [a] and J. Born [b]

[a] *Gesellschaft für Mathematik und Datenverarbeitung mbH, Postfach 1240, D-5205 Sankt Augustin 1, Germany*
[b] *Institut für Informatik und Rechentechnik, Rudower Chaussee 5, D-1199 Berlin-Adlershof, Germany*

Received January 1991

Abstract

Mühlenbein, H., M. Schomisch and J. Born, The parallel genetic algorithm as function optimizer, Parallel Computing 17 (1991) 619–632.

In this paper, the parallel genetic algorithm PGA is applied to the optimization of continuous functions. The PGA uses a mixed strategy. Subpopulations try to locate good local minima. If a subpopulation does not progress after a number of generations, hillclimbing is done. Good local minima of a subpopulation are diffused to neighboring subpopulations. Many simulation results are given with popular test functions. The PGA is at least as good as other genetic algorithms on simple problems. A comparison with mathematical optimization methods is done for very large problems. Here a breakthrough can be reported. The PGA is able to find the global minimum of Rastrigin's function of dimension 400 on a 64 processor system! Furthermore, we give an example of a superlinear speedup.

Keywords. Search methods; optimization methods; parallel genetic algorithm; performance evaluation; speedup results.

1. Introduction

The parallel genetic algorithm PGA was introduced in 1987 [9,10]. It is an adaptation of the genetic algorithm GA invented by Holland [6] to parallel computers. The PGA is totally asynchronous with distributed control. This has been achieved by the introduction of a spatial population structure and active individuals.

We have successfully applied the PGA to a number of problems, including function optimization and combinatorial optimization. In this paper we summarize the results for function optimization. The results in combinatorial optimization have been published elsewhere [4,12,13].

In all these applications we have used the same algorithm with only slight modifications. We have taken the largest published problem instances known to us. The PGA found solutions, which are comparable or even better than any other solution found by other heuristics. This is proof of its power by experiment.

We hope that future researchers will start were we have left off. The evaluation of heuristics for difficult optimization problems has to be done with large problems. Results on small toy problems cannot be extrapolated to large problems.

The most difficult part of a random search method is to explain why and when it will work. We will make a comparison with other random search methods and explain the more tricky parts of the algorithm. These are the spatial structure of the population, the selection schedule and the genetic operators.

Throughout the paper we will use biological terms. We believe that this is a source of inspiration and helps one to understand the PGA intuitively.

2. Parallel search and optimization

The PGA has been designed to solve the following general problem:

OPT 1 *P*: *Given a function $F: X \mapsto R$, where X is some metric space. Let S be a subspace of X. We seek a point x in S which optimizes F on S or at least yields an acceptable approximation of the supremum of F on S.*

Many optimization methods have been proposed for the solution of this problem. We are interested in parallel optimization methods. A parallel optimization method of parallelism N is characterized by N different search trajectories, which are performed in parallel. It can be described as follows

$$x_i^{t+1} = G_i\left(x_1^t,..x_N^t,\ F(x_1^t),..F(x_N^t)\right) i = 1,\dots, N. \qquad (1)$$

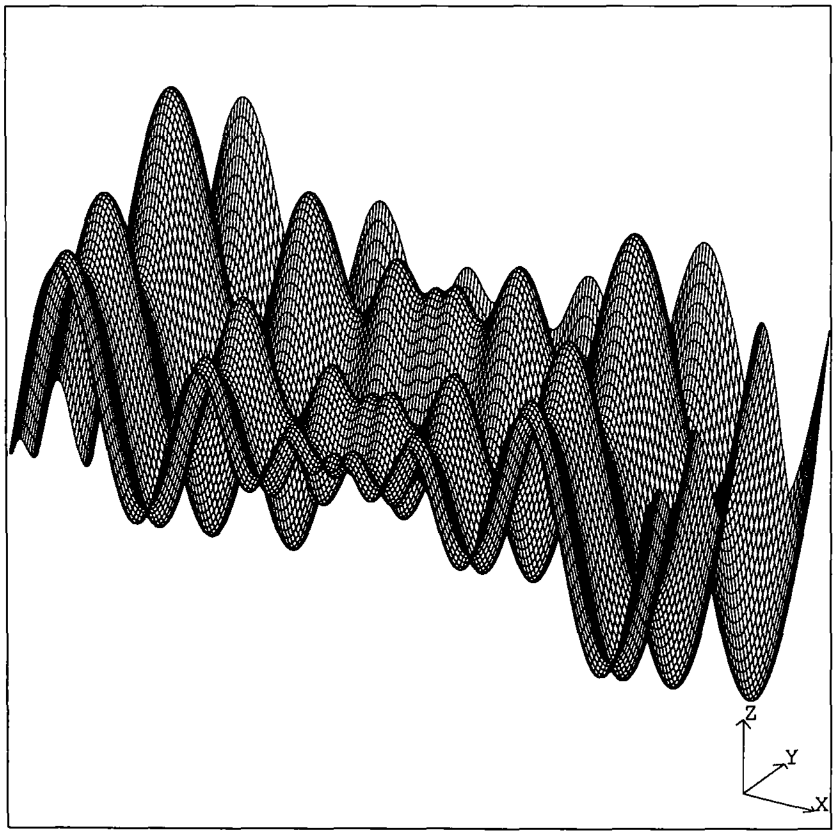

Fig. 1. 3D-version of test-function F7.

The mapping $G = (G_1, \ldots G_N)$ describes the linkage or information exchange between the parallel searches. If the N searches are independent of each other we just have

$$x_i^{t+1} = G\big(x_i^t, F(x_i^t)\big). \tag{2}$$

A parallel search method which combines the information of two searches can be described as follows

$$x_i^{t+1} = G_i\big(x_{i-1}^t, x_i^t, F(x_{i-1}^t), F(x_i^t)\big) i = 1, \ldots, N. \tag{3}$$

The basic questions of parallel search methods can now be stated:

- Are N parallel searches of time complexity t as efficient as a single search of time complexity $N * t$?
- Are N linked searches more efficient than N independent searches?
- How should the linkage be done?

In order to understand these questions intuitively, we leave the abstract mathematical description and turn to a natural search metaphor. The advantage of using a metaphor is that it leads to a qualitative understanding of the problem and the algorithm.

In this paper we use the following simple metaphor: The search is done by N active individuals, x_i describes the position of individual i and $F(x_i)$ its current value, representing the height in an unknown landscape. How should the individuals search the unknown landscape? A typical landscape is shown in *Fig. 1*.

We will investigate search algorithms which mimic evolutionary adaptation found in nature. Each individual is identified with an animal, which searches for food and produces offspring. In evolutionary algorithms, $F(x_i)$ is called the fitness of individual i, x_i^{t+1} is an offspring of x_i^t, and G is called the selection schedule.

The offspring are created by genetic operators. The genetic operators work on a genetic representation. In the simplest case the genetic representation is just a bitstring of length n, the *chromosome*. The positions of the strings are called *loci* of the chromosome. The variable at a locus is called a *gene*, its value *allele*. The set of chromosomes is called the *genotype*, which defines a *phenotype* (the individual) with a certain fitness.

3. Formal description of the PGA

The PGA can be informally described as follows. Subgroups of individuals live on a ladder-like 2-D world. New offspring are created by genetic operators within a subpopulation. Every k generations, the best individual of a subpopulation is sent to its neighbors. If the subpopulation does not progress for a number of generations, one individual will try local hillclimbing.

In the terminology of Section 2, we can describe the PGA as a parallel search with a linkage of two searches. The linkage is done probabilistically. It is biased by the spatial distribution of the individuals doing the search. The information exchange within the total population is a diffusion process.

We will now describe the PGA more formally. The PGA is a black-box solver, which can be applied to the class of problems

$$\min\{f(x) \mid x \in X\}, \text{ where } X \subseteq R^n,\ f\colon R^n \mapsto R.$$

$$X = \{x \in R^n \mid a_i \leqslant x_i \leqslant b_i,\ i = 1, \ldots, n\}, \text{ where } a_i < b_i,\ i = 1, \ldots, n.$$

Note that f is not required to be convex, differentiable, continuous or unimodal.

The PGA operates with a set (a population) of vectors (individuals). The *phenotype* of an individual is given by a real vector x with $x \in X$. The *genotype* of an individual is given by the bit representation y of x according to the floating point format of the computer.

A *gene locus* is a single bit position. The bits of the floating point representation define a *chromosome*. The genetic representation of an individual consists of a *set of n chromosomes*. This is an important difference to other GA's, which combine the floating point representations into a single chromosome.

The PGA can be described as an eight-tuple
$\mathrm{PGA} = (\mathbf{P}^0, \lambda, \mu, \delta, \tau, \mathrm{GA}, \Lambda, t)$
where
$\mathbf{P}^0$ – initial population
λ – number of subpopulations
μ – number of individuals of a subpopulation
δ – number of neighbors in a neighborhood relation among subpopulations
τ: – isolation time in generations
GA – Genetic Algorithm applied to a subpopulation
Λ – local optimizer
$t: \{0, 1\}^{\lambda} \mapsto \{0, 1\}$ – termination criterion.

In the following we call the application of the Genetic Algorithm GA to a subpopulation P_i^0, $i \in \{1, \dots, \lambda\}$ an evolution process $\mathrm{GA}(P_i^k)$, $k = 1, 2, \dots$.

Each subpopulation is mapped onto a different processor. The above definition includes the special case $\mu = 1$. Then the subpopulation consists of one member only. This case has been used extensively for combinatorical problems (see [13]).

There are many neighborhood relations possible. We have used for all our applications a neighborhood relation which can be called a ladder. An example is shown in *Fig. 2*. It is outside the scope of this paper to describe why such a population structure is promising (see [13] for details).

The PGA allows the independent evolution of the subpopulations for a certain time. This

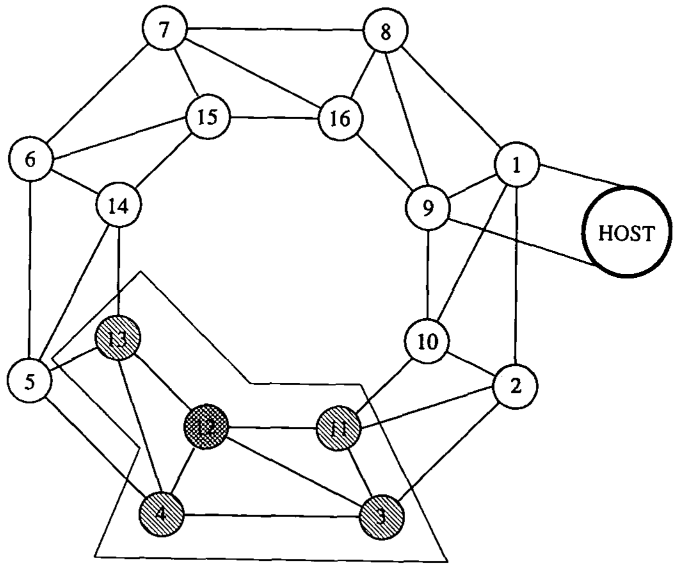

Fig. 2. 16-node version of the ladder-population. Every node represents a subpopulation P_i, $i = 1, \dots, \lambda$. Population P_{12} has four populations in its neighboorhood.

'isolation' time is given in generations. Migration to the neighbors takes place at generation $\tau, 2\tau, 3\tau, \ldots$. In the PGA only the best individuals migrate.

The termination criterion t operates on a set of termination flags generated by the evolution processes GA(P_i^k). The PGA will be terminated if all λ flags are on.

The Genetic Algorithm GA can be described as a nine-tuple:

$$\mathrm{GA} = \left(P^0, \mu, \Omega, \Gamma, \Delta, \Theta, l, \Lambda, t\right),$$

where P^0 is the initial population, μ is the population size, Ω is the selection operator, Γ the crossing-over operator, Δ the mutation operator, and Θ the recombination operator. Λ is the local hillclimbing method, l the criterion when to start hillclimbing and t is the termination flag. Ω, Γ, Δ, Λ use probability distributions.

In the following we will describe the GA in more detail. It has to be mentioned that our intention is a robust algorithm, which can be applied to a large variety of problems without adjustment of the parameters.

3.1. Selection

The selection operator works as follows. At each stage of the algorithm the population $P^k = (y_1^k, \ldots, y_\mu^k)$ is sorted in increasing order by $f(y_i^k)$.

For individuals with an odd index $i = 1, 3, 5, \ldots$ a mate is selected with a ranking scheme proposed by Whitley [18]. This scheme uses a nonlinear transformation function to prefer better individuals.

A selection index i_s is generated by:

$$i_s = \frac{\mu}{2(a-1)}\left(a - \sqrt{a^2 - 4(a-1) * \mathrm{rnd}(0,1)}\right).^{1}$$

Whitley has used the bias $a = 1.5$. We realize a higher selection pressure by using $a = 2.5$ as default.

The main purpose of this scheme is to select as severely as possible without destroying the diversity of the population too much. Each couple produces two offspring, therefore only $\mu/2$ couples are selected.

In addition, the GA uses the elitist strategy. The elitist strategy guaranties that the best individual of P^k survives to P^{k+1}.

3.2. Mutation, crossing-over and recombination

After the selection step has been done, the genetic operators Δ, Γ and Θ are applied to the two selected individuals y'^k and y''^k. All genetic operators, which the PGA uses, work in the space of the genotypes. In order to understand the effect of these operators, it is necessary to describe the genetic representation, in our case the ANSI-IEEE floating point representation.

In this representation the genotype g has the following form

S	EXP	FRAC

The value of g, the phenotype p, is given by

$$p = (-1)^s \times 1.\mathrm{frac} \times 2^{\mathrm{exp}-\mathrm{bias}}.$$

In single precision, g is 32 bits long, s is coded by 1 bit, exp by 8 bits and frac by 23 bits. Bias is equal to 127 in decimal.

[1] rnd(0, 1) denotes a random variable uniformly distributed on $\langle 0, 1 \rangle$.

The PGA mutation scheme works as follows. First, with probability p_m a chromosome is mutated at all Second, one bit is chosen randomly in the frac part and flipped. The mutated phenotype y' is given by

$$y' - y = (-1)^a 2^{-k} y,$$

where $a = 1$ if bit k is on and $a = 0$ otherwise.

The mutation operator samples the neighborhood of y exponentially more frequently than regions a larger distance from y. If the feasible set is symmetric around the origin, we also flip the sign bit s.

The crossing-over operator Γ is the traditional one point crossover applied to every homologous chromosome in the mutated individuals y'^k and y''^k with probability p_c.

The recombination operator Θ exchanges homologous chromosomes with probability p_r.

The genetic operators are explained in *Fig. 3*.

In the PGA the most important operator is the recombination operator. Recombination between $y'^k = (y_1'^k, \dots, y_n'^k)$ and $y''^k = (y_1''^k, \dots, y_n''^k)$ gives new phenotypes at the corners of the parallelepiped given by y'^k and y''^k.

Figure 4 shows the effect of recombination in the case of two dimensions. The square denotes the area, where new points are sampled by the genetic operators.

3.3. *Local hillclimbing*

The feature 'local hillclimbing' has been very important in the application domain of combinatorial optimization (see [4,13]). There are two reasons for this. First, hillclimbing is

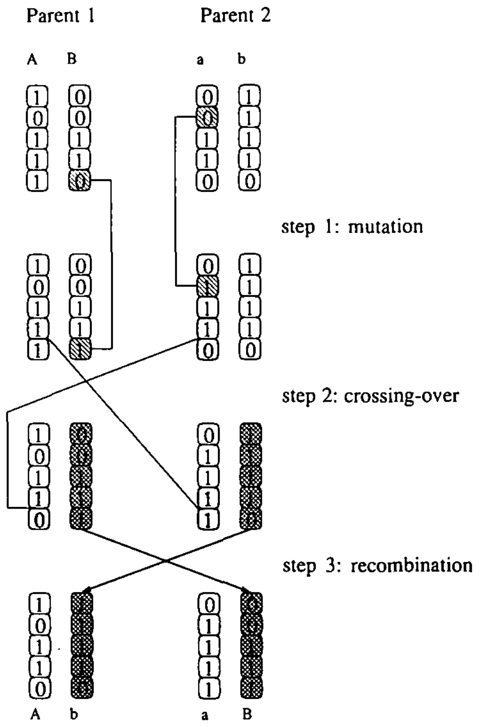

Fig. 3. How GA produces offspring. The mutation operator is a simple bit-flip. Crossing over can be described as an exchange of bit-sequences between homologous chromosomes, while recombination exchanges pairs of homologous chromosomes whithout destroying their binary structure.

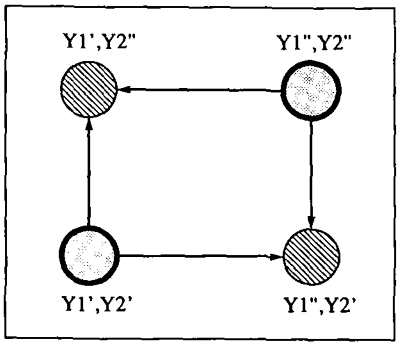

Fig. 4. Recombination in phenotype space.

often much faster than applying genetic operators and second, the hills often have a 'building block feature', i.e. each local hill has some information in common with the global minimum.

The situation is different in the application domain of function optimization. First, the evaluation of a new sample point is the same for a local hillclimbing method or a genetic operator. Furthermore, hillclimbing in the early stages of the search process does not pay off because no information about the global minimum is obtained. Therefore we decided to start the hillclimbing in the last stages of the algorithm (when be believe that the subpopulation has reached a region of attraction of a good minimum).

The decision to do local hillclimbing is done dynamically. An individual of a subpopulation is selected for local hillclimbing if one of the following criteria are fulfilled:

$$f_1^k - f_1^{k-\Delta k_1} = 0$$

$$\left\| x_1^k - x_\mu^k \right\| \leqslant a \| b - a \|.$$

f_i^k is the fitness value of an individual with rank i and phenotype x_i^k, $i \in \{1, \ldots, \mu\}$.

The subpopulations may use different strategies for hillclimbing. In this manner we want to make the algorithm robust with regard to assumptions about (unknown) regions of attraction of the global minimum.

One of the implemented local hillclimbing methods is a random search technique with adaptive step size control proposed by Solis and Wets [16] (We use 'Algorithm 1' with normally distributed steps.)

We have chosen this strategy because it is very robust and does not use derivatives of the fitness function.

The vector of start step sizes σ^0 is determined by:

$$\sigma_i^0 = \frac{1}{\sqrt{n}} \min\left\{ \alpha \| b - a \|, \max\left\{ \left\| x_1^k - x_2^k \right\|, \max\left\{ 10^{-3}, 10^{-3} \left\| x_1^k \right\| \right\} \right\} \right\}, \; i = 1, \ldots, n.$$

(x_1^k, x_2^k are the individuals with the best, respective the second best fitness value in P^k.) The control parameters of this hillclimbing method are explained in [16].

3.4. Termination

The PGA is a totally distributed algorithm. In distributed algorithms the problem of termination is difficult. In the PGA, the host program decides about the global termination (see *Fig. 2*). Every process GA(P_i^k) sets a termination flag which is used by the termination criterion of the host program. The termination flag of a subpopulation is set, if the following criterion is fulfilled:

$$|f_1^k - f_i^{k-\Delta k_2}| \leqslant \epsilon |f_1^k|.$$

This criterium is tested every 0 mod Δk_2 generations.

4. Performance evaluation

Performance evaluation of probabilistic search algorithms is a difficult task by itself. To do the evaluation, at least a performance measure and a representative suite of test functions are needed. We will use as performance measure the average number of function evaluations to compute the optimum as well as the time needed for computation. The first number measures the 'intelligence' of the algorithm, the second the speed.

The difficulties in constructing well justified test functions are similar to those in constructing mathematical models of 'typical' real life functions. Since no mathematical models for constructing test functions are known, they are constructed heuristically. Normally they consist of a number of combinations of elementary functions whose local minima are known.

In broad terms the complexity of a function optimization problem depends on:
- the number of local minima;
- distribution of the local minima;
- the distribution of the function values of the local minima;
- the domain of attraction for local minima.

We will present performance data from eight functions, which vary the above variables (see *Table 1*).

Functions F1–F5 have been proposed by De Jong [7]. They are used extensively in the

Table 1
Eight function test bed

Function number	Function	Limits
1	$f_1(x) = \sum_1^3 x_i^2$	$-5.12 \leqslant x_i \leqslant 5.12$
2	$f_2(x) = 100(x_1^2 - x_2^2)^2 + (1 - x_1)^2$	$-2.048 \leqslant x_i \leqslant 2.048$
3	$f_3(x) = \sum_1^5 \text{integer}(x_i)$	$-5.12 \leqslant x_i \leqslant 5.12$
4	$f_4(x) = \sum_1^{30} i x_i^4 + \text{Gauss}(0, 1)$	$-1.28 \leqslant x_i \leqslant 1.28$
5	$f_5(x) = 0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$	$-65.536 \leqslant x_i \leqslant 65.536$
6	$f_6(x) = nA + \sum_1^{20} x_i^2 - \mathrm{A}\cos(2\pi x_i)$	$-5.12 \leqslant x_i \leqslant 5.12$
7	$f_7(x) = \sum_1^{10} -x_i \sin(\sqrt{\lvert x_i \rvert})$	$-500 \leqslant x_i \leqslant 500$
8	$f_8(x) = \sum_1^{10} x_i^2/4000 - \prod_1^{10} \cos(x_i/\sqrt{i}) + 1$	$-600 \leqslant x_i \leqslant 600$

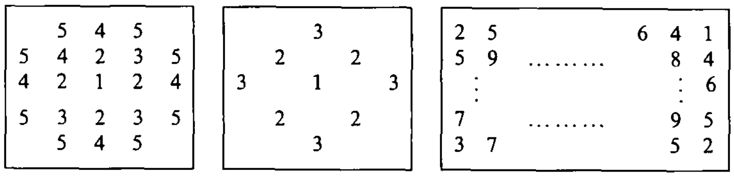

Fig. 5. Distribution of the minima for F6 (left), F7 (right) and F8 (middle) for $n = 2$. The numbers characterize the order of the minima.

Genetic Algorithms community. The test environment includes function which are discontinuous (F3), nonconvex (F2), multimodal (F5) and stochastic (F4).

Functions F6–F8 are used in 'mainstream' optimization. Each function tests the performance of a global search method on a specific aspect of the optimization problem.

F6 was proposed by Rastrigin [17]. It is highly multimodal. The local minima are located at a rectangular grid with size 1. The global minimum is at $x_i = 0$, $i = 1,\ldots, n$, giving $f(x) = 0$. Grid points with $x_i = 0$ except one coordinate, where $x_i = 1$, give $f = 1$, the second best minimum. With increasing distance to the global minimum the fitness values of local minima become larger. (See *Fig. 5*).

Function F7 was proposed by Schwefel [14]. Here we consider the function in ten and more dimensions. The global minimum is at $x_i = 420.9687$, $i = 1,\ldots, n$. The local minima are located at the points $x_k \approx (\pi(0.5 + k))^2$, $k = 0, 2, 4, 6$ for positive directions of the coordinate axis and $x_k \approx (\pi(0.5 + k))^2$, $k = 1, 3, 5$. Points with $x_i = 420.9687$, $i = 1,..n$, $i \neq j$, $x_j = -302.5232$, give the second best minimum – far away from the global minimum. Therefore the search algorithm may be trapped in the wrong region. (See *Fig. 5*.)

Function F8 has been suggested by Griewangk [3]. The function has its global minimum $f = 0$ at $x_i = 0$, $i = 1,\ldots, n$. The local minima are located approximately at $x_k \approx \pm k * \pi\sqrt{i}$, $i = 1,\ldots, n$, $k = 0, 1, 2, 3,\ldots$.

In ten dimensions there are four suboptimal minima $f(x) \approx 0.0074$ at $x \approx (\pm\pi, \pm \pi\sqrt{2}, 0,\ldots,0)$. (See *Fig. 5*.)

5. Numerical results

The PGA has many internal control parameters, which have been derived from the evolution metaphor. It is not necessary to tune all these parameters for a specific function. The following numerical results have been obtained with a single set of parameters. This demonstrates the robustness of this search method. The parameters, which have been used for these runs, are given in *Table 2*.

Numerical results are shown in *Table 3* for the case $\lambda = 4$, $\mu = 20$ and in *Table 4* for the case $\lambda = 8$, $\mu = 20$.

Table 2
Control parameter setting

Number of subpopulations $\lambda = 4, 8$
Number of individuals of a subpopulation $\mu = 20$
Isolation time $\tau = \mu/2$
Mutation probability $p_m = 1/n$ ($= 3/n$ in case of F8)
Crossing over probability $p_c = 0.65$
Recombination probability $p_r = 0.5$

Table 3
Results for $\lambda = 4$

Function	n	Best		Worst		Average	
		Time	Feval	Time	Feval	Time	Feval
F1	3	0.48	340	2.47	2020	1.44	1170
F2	2	0.69	580	2.50	2180	1.48	1235
F3	5	1.75	1360	7.26	8780	3.44	3481
F4	30	7.52	2000	29.15	7600	12.08	3194
F5	2	1.25	400	10.91	4580	3.33	1256

F5: The global optimum was not found in 3 of the 50 runs.

The average is based on 50 runs. In all runs the global minimum has been found to at least three digits of accuracy. The time needed for a run is the time at the host between starting the computation and arriving of the minimum function value at the host. The number of function evaluations is not as accurate because of the termination problem mentioned earlier. If the minimal function value arrives at the host computer, it sends a termination command to all processors. Before terminating, these processors send the number of function evaluations done so far to the host which sums this number up.

It is instructive to compare our results with the results of Eshelman et al. [1], which are shown in *Table 5*. G is the average number of function evaluations with optimal parameter setting by Grefenstette [2] and E is the average number of function evaluations with optimal parameter setting by Eshelman (We present the results for the best version: 'eight-point traditional crossover').

The non-optimized PGA for F1 and F2 needs less function evaluations than these optimized Genetic Algorithms which use binary coding and a simple panmictic population. The most dramatic improvements is with F2. The discontinuous function F3 is solved better by the standard GA. This may indicate that the PGA is tuned more to continuous functions than to discontinuous ones. The results for F4 are in the same order for $\lambda = 4$. Shekel's foxholes problem F5 is also solved better by the PGA. We found these results encouraging, because the problems are too small for the PGA to work ideally.

We turn now to the more complex functions. F6 has been used by Hoffmeister et al. [5] to compare Genetic Algorithms with Evolution Strategies. Their methods did not find the global minimum within 250 generations. In a population of size 50 this gives 12500 function evaluations. After 250 generations they arrived at a minimum value of $f = 10$. The PGA found the optimum in all runs with an average of 9900 evaluations.

Table 4
Results for $\lambda = 8$

Function	n	Best		Worst		Average	
		Time	Feval	Time	Feval	Time	Feval
F1	3	0.69	780	3.75	3180	1.57	1526
F2	2	0.70	920	2.74	2480	1.68	1671
F3	5	1.43	1480	6.14	7200	3.43	3634
F4	30	6.84	3600	12.79	6720	9.92	5243
F5	2	1.09	740	4.6	3580	2.79	2076
F6	20	7.55	6980	12.29	16320	9.41	9900
F7	10	4.38	3780	11.83	25140	6.61	8699
F8	10	11.24	39240	45.47	123080	16.84	59520

F7: The global optimum was not found in 4 of the 50 runs. F8: $\lambda = 16$, $\mu = 40$; $p_m = 0.3$

Table 5
Results of Grefenstette and of Eshelman

Function	*n*	*G* feval	*E* feval
Fl	3	2210	1538
F2	2	14229	9477
F3	5	2259	1740
F4	30	3070	4137
F5	2	4334	3004

A typical run is shown *Fig. 6*. In the beginning there is very good progress until a fitness value of about 20 is reached. In the last part of the algorithm the subpopulations get similar and concentrate around suboptimal minima. The global minimum is found by recombination only.

There are no results available for function F7 in higher dimensions. Griewangk's function F8 is considered to be one of the most difficult test functions. The following results for F8 have been compiled in Törn and Zielinkas [17].

Griewangk's algorithm [3] needed 6600 function evaluations (average number), but only one of the four suboptimal minima was found. Snyman and Fatti [15] found the global minimum in 23399 function evaluations. Rinnooy Kan and Timmer [8] also tried function F8. They wrote: "For the two-dimensional problem, our method never really got started. Only one minimum was found, after which the method terminated. For the ten-dimensional problem, the global minimum was found in six of the ten runs. The method terminated at an early stage using 700 function evaluations on the average. In the other four runs the method was stopped after 4000 points were sampled. At this time 17000 function evaluations had been used."

A performance evaluation based on time is not possible because of lack of data. In mainstream optimization a normalization of the computing time is done in order to compare the results for different machines. The normalized time unit is 1000 computations of Shekel's

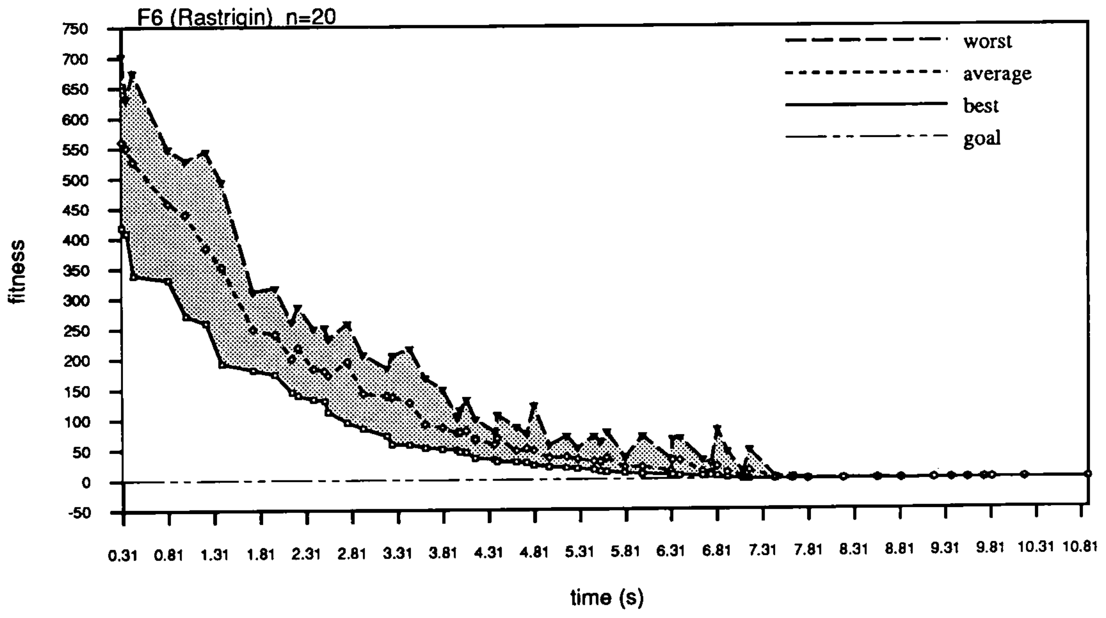

Fig. 6. Average run for test-function F6.

functions [17]. For our machine, the MEGAFRAME HyperCluster [11], the time unit is 1.7640 on one processor.

We will now turn to very large problems, which have not been dealt with before. The PGA is an asynchronous parallel algorithm and therefore ideally suited to parallel computers. The behavior of the PGA on parallel processors will be investigated in the next section.

6. Superlinear speedup

Measurements of the speedup of the PGA on parallel processors is very difficult. First, each individual run is different because of the probabilistic nature of the algorithm. Second, a different number of processors needs a different parameter λ, which changes the search strategy.

The data in *Tables 3* and *4* show a small speedup for the functions F4 and F5 only. The global minimum of F1–F3 is computed faster on four processors than on eight processors. This situation is typical for small problems. Parallel search pays off only, if the search space is large and complex.

The largest speedup is to be expected with function F7. Here the suboptima are far away from the optimmum, so that a parallel search with subpopulations seems to be promising. *Table 6* shows the result.

In the table a superlinear speedup can be observed when going from four to eight processors. The speedup is 2.6. This can be explained easily. Eight processors need less function evaluations than four processors to locate the minimum. The speedup between one processor and four processors would have been still more. Unfortunately we cannot make a comparison, because a single panmictic population (we tried up to $\mu = 640$) was not able to find the minimum. This shows the advantage of a spatial population structure. Similar results have been found with combinatorial optimization problems [13].

All optimization problems so far have been solved by a small number of processors. We claim that the PGA is a breakthrough in parallel search methods. We hope to demonstrate this with the next examples. In these examples we will try to solve very large problem instances with the maximum number of processors we have available. The number is 64 [11].

The number of function evaluations for F6 increases approximately like $n * \sqrt{n}$ in the range from $n = 20$ until $n = 200$ (*Table 7*). The same is true for F7 in the range from $n = 10$ until $n = 50$. Then the number of function evaluations increases rapidly. It is an open question, if a much larger number of processors could reduce the above numbers substantially. F6 – 400 and F7 – 150 are single runs only.

7. The PGA and mathematical optimization

There exists no unique optimal search method. Most of the proposed search methods for global optimizations use heuristics to generate the sample points as well as possible, given the information at hand. The search strategy of the PGA can be explained as follows.

Table 6
Superlinear speedup F7 ($n = 30$)

λ	μ	Best		Worst		Average	
		Time	Feval	Time	Feval	Time	Feval
4	80	68.7	37280	114.2	79280	84.6	50720
8	40	27.4	28640	35.6	43560	32.3	37573
16	20	15.1	31680	20.1	52980	17.2	39773

Table 7
Benchmarks for F6 and F7 (5 runs)

Funktion	n	λ	μ	Average	
				Time	Feval
F6	50	8	20	57.40	42753
	100	16	20	129.69	109072
	200	32	40	404.26	390768
	400	64	40	4849.458	7964400
F7	50	32	20	34.96	119316
	100	64	20	213.93	1262228
	150	64	40	1635.56	7041440

λ random samples of μ points are placed uniformly within the feasible set. Within each subgroup additional points are tested by recombination. These points are drawn probabilistically out of the n-dimensional grid defined by the μ sample points. Selection concentrates the sample points at promising regions. Therefore the size of the grid, i.e. the region tested, gets smaller. If the points are too close to each other, local hill-climbing will be done. So each subgroup ends up at one or more local minima.

By migration to the neighboring subgroups, the information of good points is spread. In the last stage of the algorithm the grid defined by the local minima found is also tested by recombination. This is the reason why subpopulations are so important in the PGA. The subpopulations do not need to converge to the global minimum, this can be found later by recombination.

It is instructive to compare the PGA heuristic with more standard optimization techniques. The most similar heuristic is used by clustering methods. For grouping points around minima, two strategies have been used. The first strategy tries to form groups around *some* of the most promising local minima, the second tries to produce groups at *all* local minima. In both cases a cluster analysis is applied to prevent redetermination of already known local minima.

The second strategy cannot be applied to large optimization problems. Rastrigin's function F6 has n^{11} local minima, where n is the dimension! We have to use strategies which concentrate the point in promising areas. In standard optimization three pure strategies and some combinations between these are used [17].

CON1: *Retain a pre-specified portion of the best points.*
CON2: *Displace the points by some steps of a local optimizer.*
CON3: *Replace the worst point with a better point anywhere in the region.*
CON4: CON1 *followed by* CON2.
CON5: CON2 *followed by* CON1.

The similarity to strategies used by the PGA is obvious. The major difference is the cluster analysis to prevent redetermination of already known local minima.

8. Conclusion

The PGA is no miracle, it is a very simple parallel search which concentrates implicitly the search in promising regions. It suffers from the fact that in the last stage of the algorithm the same function evaluations are done over and over again. Nevertheless, the PGA is able to solve complex search problems in very high dimensions which have not been solved before.

It has to be proven, that search methods based on *rational* arguments are able to solve such large problems. We hope that researchers in mathematical optimization methods take up our challenge!

An objection often mentioned against genetic algorithm is, that there is no convergence theorem. But any algorithm which tries to estimate the probability to find the global minimum will suffer from the *curse of dimensionality*. This can be easily be demonstrated. In n dimensions, a hypercube of size a has 2^n hypercubes of size $1/2a$. With statistical information only, we need an exponential number of sample points to be sure that the global minimum is in one of the subcubes!

The most important feature of the PGA is that it is a totally asynchronous parallel algorithm. It runs with 100% efficiency on any number of processors. Many extensions of the PGA are straightforward. We only want to mention the following: Subpopulations could do different search strategies, the population structure could change during the run, the population could consist of different species etc. These extensions will be implemented in the course of applying the PGA to more and more challenging applications.

References

[1] L.J. Eshelman, R.A. Caruana and J.D. Schaffer, Biases in the crossover landscape, in: J.D. Schaffer, ed., *Proc. Third Internat. Conf. on Genetic Algorithms* (1989) 10–19.

[2] J.J. Grefenstette, Optimization of control parameters for genetic algorithms, *IEEE Trans. Systems, Man Cybernet.*, 16 (1986) 122–128.

[3] A.O. Griewangk, Generalized descent for global optimization, *JOTA* 34 (1981) 11–39.

[4] M. Gorges-Schleuter, Genetic algorithms and population structure–a massively parallel algorithm, PhD thesis, University of Dortmund, 1990.

[5] F. Hoffmeister and Th. Bäck, Genetic algorithms and evolution strategies: similarities and differences, in: H.-P. Schwefel, ed., *PPSN – First Internat. Workshop on Parallel Problem Solving from Nature*, Dortmund, FRG (October 1–3, 1990) Preprints, 1990.

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

[7] K.A. De Jong, An analysis of the behavior of a class of genetic adaptive systems, PhD thesis, University of Michigan, Dissertation Abstracts International 36(10) 5140B (University Microfilms No. 76-9381), 1975.

[8] A.H.G. Rinnooy Kan and G.T. Timmer, Stochastic global optimization methods, I and II, *Mathemat. Programming* 39 (1987) 27–26, 57–78.

[9] H. Mühlenbein, M. Gorges-Schleuter and O. Krämer, New solutions to the mapping problem of parallel systems – the evolution approach, *Parallel Comput.* 4 (1987) 269–279.

[10] H. Mühlenbein, M. Gorges-Schleuter and O. Krämer, Evolution algorithm in combinatorial optimization, *Parallel Comput.* 7 (1988) 65–85.

[11] H. Mühlenbein, O. Kräemer, G. Peise and R. Rinn, The MEGAFRAME HyperCluster – a reconfigurable architecture for massively parallel computers, in: *PARCELLA 90* (Akademie Verlag, Berlin, 1990) 143–156.

[12] H. Mühlenbein, Parallel genetic algorithms, population genetics and combinatorial optimization, in: J.D. Schaffer, ed., *Proc. Third Internat. Conf. on Genetic Algorithms* San Mateo, CA (1989) (Morgan Kaufman, Los Altos, CA, 1989) 416–421.

[13] H. Mühlenbein, Evolution in time and space–the parallel genetic algorithm, in: G. Rawlins, ed., *Foundations of Genetic Algorithms* (Morgan Kaufman, Los Altos, CA, 1991).

[14] H.P Schwefel, Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie, Vol. 26 of *Interdisciplinary System Research* (Birkhäuser, Basel, 1977).

[15] J.A. Snyman and L.P. Fatti, A multi-start global minimization algorithm with dynamic search trajectories, *JOTA* 54 (1987) 121–141.

[16] F.J. Solis and R.J-B. Wets, Minimization by random search techniques, *Mathemat. Operat. Res.* 6 (1981) 19–30.

[17] A. Törn and A. Zilinskas, Global optimization, Vol. 350 of *Lecture Notes in Computer Sciences* (Springer, Berlin, 1989).

[18] D. Whitley, The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best, in: J.D. Schaffer, ed., *Proc. Third Internat. Conf. on Genetic Algorithms*, San Mateo, CA (1989) (Morgan Kaufmann, Los Altos, CA, 1989) 116–121.