I Krajowa Konferencja
Algorytmy Ewolucyjne
Murzasichle, 12-15.06.1996

Self-adapting Parallel Genetic Algorithm with the Dynamic Mutation Probability, Crossover Rate and Population Size.

Joanna Lis
Institute of Biocybernetics and Biomedical Engineering
Polish Academy of Sciences
Trojdena 4, Warsaw, Poland, 02-109
e-mail: joanna.lis@ibib.waw.pl

Mirosław Lis
FRACTAL Software Studio
Obroncow Warszawy 6/5
Plock, Poland, 09-402

Abstract:

In this paper the parallel genetic algorithm with dynamic mutation, crossover rates and population size is proposed. This algorithm is based on the farming model of parallel computing. The basic idea of the dynamic establishing of control parameters is presented. Several experiments on function maximization problems have been performed to study the effects of varying the settings for the parallel model.

1. INTRODUCTION.

Genetic Algorithms (GAs) belong to the important mainstream forms of Evolutionary Algorithms (EAs), which are search and optimization techniques based on the principles of natural evolution (De Jong, 1975). GAs have been widely studied as function optimization methods. In many cases the standard GA turned out to be inefficient in the multimodal optimization problems. This was the reason for the introduction of a couple of advanced techniques based on improved selection schemes, the recombination operators and the adaptation of the algorithm's strategy parameters (Michalewicz, 1992).

EAs are controlled by several strategy parameters. In the case of GAs these are the population size (s), crossover probability (p_c), mutation rate (p_m), the length of individuals, generation gap, crowding factor, or the maximum expected value in case of ranking. However, it is clear that different GA parameter settings will be optimal for different objective functions.

The settings most commonly used up till now were determined empirically in a large number of experiments (De Jong, 1975), (Grefenstette, 1986), (Schaffer, 1989).

source	population size	crossover probability	mutation probability
[DeJo75]	50-100	0.6	0.001
[Gref86]	30	0.95	0.01
[Scha89]	20-30	0.75-0.95	0.005-0.01

Table 1. The optimal values of the parameters of the GA using a binary version of solution.

A number of researchers proposed including the GA control parameters directly into the genetic code (De Jong, 1975), (Shaefers, 1987). Another attempt of adaptively changing the GA configuration is also the "punctuated crossover" operator (Schwefel, Maenner, 1991).

Among the most promising methods of speeding up the convergence and improving robustness of evolutionary optimizations there are the adaptive techniques, which modify EAs parameter settings adaptively during the search process. According to the experimental and theoretical arguments an optimal control, exogenous parameters for all possible topologies of the objective function does not exist. Therefore, the optimal settings of the GA must be

determined for every application separately. Hence, the investigations of the adaptive techniques seems to be very promising from the point of view of a much broader variety of optimization algorithms, for which the general principle self-adapting mechanisms might be applicable.

This paper is organized as follows. The next sections present the background of GA and PGA adaptive extensions. In section 3 the parallel model of a GA with dynamic control parameters is presented. Section 4 describes the experiments and the results. The last section contains concluding remarks and hints for further research.

2. ADAPTIVE EXTENSIONS OF GAs

The idea of adapting the crossover probability, mutation rate and population size to improve the GAs performance has been employed previously. In the crossover mechanism discussed by Schaffer the distribution of the crossover points is adapted according to the performance of the generated offspring. The distribution information is encoded into each string using additional bits (Schaffer, 1987). Fogarty has studied the effects of varying the mutation rate over generations and integer encodings; the mutation rate decreased exponentially with generations (Fogarty, 1989). Davies discusses an effective method of adapting operator probabilities based on the performance of the operators. The mutation probability determined in advance was distributed among several competing genetic operators according to the number of the descendant chromosomes which acquired values of the fitness function better than any previous one (Davis, 1989). In (Whitley, 1989) the mutation probability value was conditioned by the mean Hamilton distance of solutions represented by the chromosomes in the context of a steady state GA. The method proposed in (Lis, 1995) consists in modifying the mutation probability (the mutation probability is determined for each individual as a function of its fitness, similar to variance of fitness) when the GA is being generated; it is based on transitory test results and allows to reach a nearly optimal mutation probability in a short time span. The approach described by Bäck in (Bäck, 1992) is different from the above mechanisms and the principle of his self-adapting method has been directly taken from Evolutionary Strategies. The adaptive mutation rates are handled as temporary and individually differing parameters. The mutation rates are initialized randomly and encoded as bit strings. They are also subject to mutation and selection. Very interesting self-adapting mechanism in EAs is described in (Fogel, 1995). Goldberg et al. investigated the effect of population size on GA behavior (Goldberg, 1992). A population sizing equation was derived that involves variance of building-block, fitness, noise of the genetic operators, and noise in the objective function. The authors found that with small populations, the GA performance is determined by chance, by mutation, or by another mechanism that serially injects diversity. Among several extensions to this work an on-line population sizing technique was proposed. The on-line sizing of the population could be based on information about the problem size, population variance, minimum signal and order of deception. Smith in (Smith, 1993) introduced an algorithm which adjusts the population size to the probability of selection error. Arabas et al devised an adaptive method for maintaining variable population size, which grows and shrinks according to some characteristic of the search (Arabas, 1994).

3. THE EFFECTS OF MUTATION, CROSSOVER AND POPULATION SIZING FOR PARALLEL GENETIC ALGORITHM

3.1. THE CHOICE OF THE CONTROL PARAMETERS IN PARALLEL IMPLEMENTATION OF GA

Good results obtained with the adaptive techniques suggest further investigations of this idea, with special emphasis on the parameters: p_m, p_c and s. On the other hand, due to

increasing demands for the application of genetic algorithms to large and complex search spaces with costly evaluation functions, and using large population sizes to reduce the problem of premature convergence, there is an ever growing need for fast implementations which allow quick and flexible experimentation. Parallel processing is the natural route to explore. Genetic algorithms are well suited for parallel execution (Fogarty, 1991), (Stender, 1994). In the case of parallel computer architecture design for GA new possibilities of automatic calculation of GA control parameters emerge.

The idea of adopting control parameters to improve the performance of parallel GA has been utilized previously (Mühlenbein, 1987). Very interesting ways of differentiating the subpopulations consist (Cohoon, 1991) in using for the evolution of each subpopulation GAs with different control parameters, such as crossover and mutation rates or population size. Tanese presented an algorithm (Tanese, 1989), in which the individuals sent by each subpopulation to the neighboring ones were chosen randomly from among those whose fitness was at least equal to the average fitness of the subpopulation. Tanese's experiments were performed with parameter settings (mutation and crossover rates) different from those recommended by De Jong, and with different settings for each subpopulation (Tanese, 1989). Whitley in (Starkweather and Whitley, 1991) suggests that "adaptive mutation" may be an important factor in obtaining superior results using PGA. Whitley's adaptive mutation establishing the mutation rate for the offspring according to the similarity of the two parents.

In this paper a new method of the dynamic establishing of the PGA mutation rate, crossover probability and population size is proposed. This method is based on a farming model of parallel GA and probability of mutation is established during execution of the algorithm.

3.2. Parallel Genetic Algorithm with the Dynamic Mutation Probability

A. Model.

The farming model (Marin, 1995) of parallelizing the problem of establishing mutation probability during the algorithm execution is used. The transputer net is organized in a master-slave fashion. The interconnection topology is logically a star, with the master in the center. The master processor allows the user to define parameters for a run and view the current result of the run.

The basic idea of the approach proposed in this paper is as follows. At the beginning of a run the master processor randomly generates an initial population, which it distributes among the slave processors and then supervises the activities of these processors. Each of the slaves executes the sequential GA. The population is evolving independently during a certain number of iterations called epoch or migration period. During the run, the slave processors periodically (after each epoch) report their best partial results (the individuals with the best fitness function values) to the master. The master assembles these results and re-sends them to the slaves as their new populations (the same for each slave processor).

B. Establishing the mutation probability.

A set of different mutation probability values (levels) is determined. At the beginning of the first epoch, three consecutive values of mutation probability are chosen. Next, each of the slave processors executes the sequential genetic algorithm with one of these three values of the mutation rate. Assume that during a given epoch, the best result (in the sense of the fitness function value) is obtained by the process with the highest mutation probability (from among three several consecutive values). Then, the probability of mutation for each process is shifted one level up. In the opposite case the mutation probability is shifted towards lower values. If

Run serial Genetic Algorithm on each process for number_of_generations

Receive the best individuals from processes

Form a new population

If the best individual was obtained by process with max mutation probability

Increase mutation rate for all processes to higher levels

If the best individual was obtained by process with min mutation probability

Decrease mutation rate for all processes to lower levels

Endfor.

Where: num_of_epoch and number_of_generations are the constant external parameters.

3.3. EXTENSION TO PARALLEL GENETIC ALGORITHM WITH SEVERAL DYNAMIC PARAMETERS

A. The method of dynamic establishing of the mutation probability executes a number of generations for three different values (levels) of mutation rate. For each of these values the fitness function is evaluated. The best fitness function value points at the best parameter value. The above method can be extended to dynamically establish more than one control parameter. This extension consists in constructing the best set of parameters (in the sense of the fitness function value).

The most intuitive route to proceed study those control parameters behavior would be to vary the parameters of interest in a full experimental design, that is to try all possible combinations of low, medium and high values of these parameters. This would work fine, except that the number of necessary runs would increase geometrically. For example, if we want to study 3 control parameters (p_m,p_c,s), the necessary number of runs would become $3^3 = 27$.

In order to eliminate the time-consuming computation, the statistic experiment planning methods - the Latin Squares designs - can be useful. These methods are typically used to identify the optimum settings for the different factors that affect the experimental process. The Latin Squares designs are used when the features of interest have more than two levels, and the interactions between factors are not interesting.

The example parameter sets for 3 parameters with 3 values (low L, medium M and high H) are showed below in Table 2.

experiment number	1	2	3	4	5	6	7	8	9
parameter 1	L	L	L	M	M	M	H	H	H
parameter 2	L	M	H	L	M	H	L	M	H
parameter 3	L	M	H	M	H	L	H	L	M

Table 2. Latin Squares experiment design.

After applying the Latin Squares designs the number of GA runs was significantly decreased (9 runs instead of 27). Notice, that for each particular parameter value, there are experiments with all the possible values of other parameters. Hence, this method allows an unbiased estimation of the particular parameter influence on the experiment result.

Suppose that the value of one of the parameters, say p_m, is fixed at level i. The performance of the GA may than be measured by the average of the best fitness function values obtained during all runs in which all the remaining parameters (p_c, s) were varied. The level i giving the best performance is then chosen and used to determine p_m employed by all the processes

the best fitness function value corresponds to the medium mutation probability is left unchanged.

The procedure is based on the assumption, that for a given environment (e.g. optimized function, GA version) and for a given generation there is one optimal value of mutation probability, and that the GA performance dependence on mutation probability is unimodal. Of course, the performance is affected by noise, but the experiments indicate, after the noise component is smoothed, the above assumption are well justified (e.g. (Lis, 1994)).

The following figure (Fig.1) illustrates how the method works. The vertical axis show what fitness function value can be obtained after some generation, horizontal axis show the mutation probability.

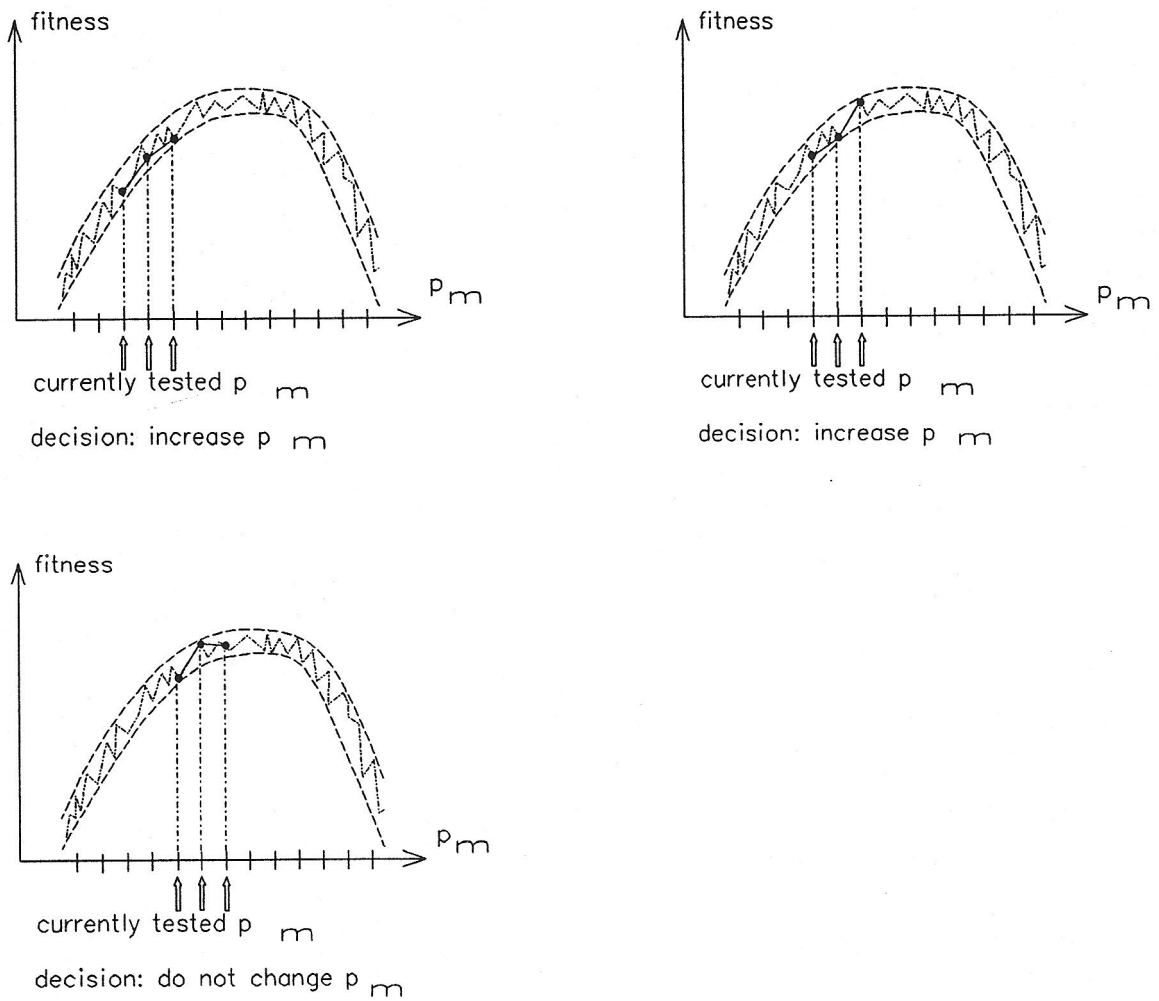

Fig.1.a,b,c. Mutation probability modification.

C. The scheme of PGA with dynamic mutation probability.

```
Determine a set of possible mutation probability levels
Choose three initial mutation probability levels from the set of mutation levels for processes
Randomly generate an initial population
For epoch = 1  to num_of_epoch do
    Send the population to processes
    Send the control parameters to processes
```

(farms) during several next generations. These p_m values may be shifted one level up (if i = 3), left unchanged (if i = 2), or shifted one level down (i = 1). Optimal values of all the other control parameters are determined using the same strategy.

B. After extension to more than one dynamic parameter, the scheme of PGA is as follows:

```
Determine a set of possible parameters levels
Choose initial parameters levels from the set of allowed levels for processes
Randomly generate an initial population
For epoch = 1  to num_of_epoch do
    Send the population to processes
    Send the control parameters to processes according to experiment plan
    Run serial Genetic Algorithm on each process for
                    average_number_of_generations * average_size / population_size
    Receive the best individuals from processes
    Form a new population
    For each parameter do
        For each of 3 control parameter levels do
            Average best individuals fitness for all processes using this level
        Endfor
        If the best average was obtained by process with max parameter level
            Increase parameter for all processes to  higher levels
        If the best average was obtained by process with min parameter level
            Decrease parameter for all processes to lower levels
    Endfor
Endfor.
```

To give equal chances to the competing processes with different population sizes, the product of population size and number of generations in epoch are equal. The average number of generations and average size are defined in advance. Then, at the beginning of each epoch the number of generations for each process is calculated, according to the population size parameter. It means that every process has the same time to show its usefulness.

4. EXPERIMENTS AND RESULTS.

The presented method was applied to dynamic establishing of the mutation probability, the crossover probability and population size. The PGA with dynamic control parameters was tested on a set of functions G_1, G_2, G_3, G_4 (Arabas et al, 1994).

The test functions are as follows:

G_1: $-x\sin(10\pi)+1$ $\quad -2.0 \le x \le 1.0$

G_2: $\text{integer}(8x)/8$ $\quad 0.0 \le x \le 1.0$

G_3: $x \cdot \text{sgn}(x)$ $\quad -1.0 \le x \le 2.0$

$$G_4: 0.5+\frac{\sin^2\sqrt{x^2+y^2}-0.5}{\left(1+0.001\left(x^2+y^2\right)\right)^2} \qquad -100.0 \le x,\ y \le 100.0$$

Performance of the PGA with dynamic control parameters has been compared to the performance of the Goldberg's Simple Genetic Algorithm (SGA) (Goldberg, 1989) and the GAVaPS (Genetic Algorithm with Varying Population Size) (Arabas, 1994).

The following assumptions were made for these experiments:

The problem coding methods as well as genetic operators were identical for SGA, GAVaPS and PGA (a simple binary coding has been used and two genetic operators: mutation and one point crossover). The initial parameter settings were initialized at random. In order to mimic the same conditions for the compared experiments, the stop criterion was changed. Instead of executing GA for certain number of epochs, the algorithm was stopped after 20 iterations, in which the best individual fitness function value did not improve. As in the reference experiments, the population was initialized randomly and 20 independent runs were performed. Then, the best individual fitness function values were averaged over these 20 runs. In Table 3 this average value is compared with the results reported in (Arabas et al, 1994).

Type of the algorithm	Functions			
	G_1	G_2	G_3	G_4
PGA with Dynamic Mutation, Crossover and Population Size	2.919	0.875	2.000	0.997
SGA	2.814	0.875	1.996	0.959
GAVaPS (1)	2.831	0.875	1.999	0.969
GAVaPS (2)	2.841	0.875	1.999	0.970
GAVaPS (3)	2.813	0.875	1.999	0.972

Table 3. Comparison of different Genetic Algorithms.

As it is showed in Table 3, the PGA presented here outperformed the other algorithms considered for every test function.

The parallel genetic algorithm with the dynamic mutation, crossover and population sizing was implemented in OCCAM 2 on INMOS transputer board. The board contained four T 800 transputers, each with 1 MB of memory. The transputers were cycled by 25 MHz clock. The OCCAM programing language is a high level language, designed to express concurrent algorithms and their implementation on a network of processing components. OCCAM enables an application to be described as a collection of processes, where each process executes concurrently, and communicates with other processes through channels.

5. Conclusion and Further Work

The behavior of GA's control parameters is complex and it has been the subject of intensive research. For example, it is known that the performance of a GA can range from that of random search to hill climbing, depending on the GA parameters settings. Designing a GA that meets the resource constraints of a given application may require a substantial amount of GA control parameter tuning. The adapting strategy introduced here, allowing to establish

several control parameters dynamically can be used in many applications. For example, one could imagine using several different mutation operators, each with its own probability rate. Because of the dynamic establishing of these rates the congruency of the operators can occur. Investigating the possible dominance of the operators in different stages of a GA run can be very interesting. This is also the way to introduce new operators, useful only in case of specific situations, e.g. at the beginning of a GA run. Once such an operator completes its task, its probability should decrease.

The goal of the technique presented here is to design a "self-learning" GA that can be repeatedly applied to a particular class of problems.

6. REFERENCES.

(Arabas, 1994) Arabas J., Z. Michalewicz, J. Mulawka, "GAVaPS - a Genetic Algorithm with Varying Population Size" in Proc. 1st IEEE Int. Conf. on Evolutionary Conference, Orlando, USA, pp. 73-78, 1994.

(Bäck, 1992) Bäck T., F. Hoffmeister, "Genetic Algorithms and Evolutionary Strategies. Similarities and Differences", Report of the Systems Analysis Research Group SYS/92, University of Dortmund, Department of Computer Science, 1992.

(Cohoon, 1991) Cohoon J.P., Martin W.N., Richards D.S., " A Multi-population Genetic Algorithm for Solving the K-Partition Problem on Hyper-cubes", in Proc. of the 4-th Int. Conf. on GAs, 1991

(Davis L. 1989). "Adapting operator probabilities in genetic algorithms", in Proc. 3rd Int. Conf. Genetic Algorithms, Morgan Kaufman, San Mateo, 1989.

(De Jong, 1975) De Jong K. A.,"An analysis of the behavior of a class of genetic adaptive systems", Ph.D. thesis, Univ. of Michigan, Ann Arbor MI,1975

(Fogarty, 1989) Fogarty T. C., "Varying the probability of mutation in the genetic algorithm", in Proc. 3rd Int. Conf. GAs, pp.104-109, J. Schaffer (ed) Morgan Kaufman, San Mateo, 1989.

(Fogarty, 1991) Fogarty T., "Implementing the Genetic Algorithm on Transputer Based Parallel Processing Systems", Parallel Problem Solving from Nature 1, 1991.

(Fogel, 1995) Fogel D. B, "Evolutionary Computation. Toward a New Philosophy of Machine Intelligence", IEEE Press, New York, 1995.

(Goldberg, 1992) Goldberg D.E., K. Deb and J.H. Clark, "Genetic Algorithms, Noise and the Sizing of Populations", Complex Systems, Vol.6, No.4, pp. 333-362, 1992.

(Grefenstette, 1981) Grefenstette J., "Parallel Adaptive Algorithms for Function Optimization", Vanderbilt University, Technical Report CS-81-19, 1981.

(Grefenstette, 1986) Grefenstette J., "Optimization of control parameters for Genetic Algorithms", IEEE Transactions on Systems, Man and Cybernetics, Vol.16, No.1, pp.122-128, 1986.

(Lis, 1994) Lis J.," Algorithms of classifiers design based on neural networks", doctoral thesis. Institute of Biocybernetics and Biomedical Engineering, PAS, Warsaw, 1994.

(Lis, 1995) Lis J., "Genetic Algorithm with the Dynamic Probability of Mutation in the Classification Problem", Pattern Recognition Letters, vol.16, pp.1311-1321, 1995.

(Marin, 1995) Marin F., Trelles-Salazar O., Sandoval F., "Genetic Algorithms on LAN-message Passing Architectures using PVM: Application to the Routing Problem",pp.1995.

(Michalewicz, 1992) Michalewicz Z. "Genetic algorithms + data structures = evolution programs", Springer-Verlag, 1992

(Muehlenbein, 1987) Muehlenbein H., Georges-Schleuter M., Kraemer O. "New solutions to the mapping problem of parallel systems: The evolutionary approach", Parallel Computing 4, pp.269-279, 1987.

(Schaffer, 1987) SchafferJ.D.,Morishima A., "An adaptive crossover distribution mechanism for GAs"in Proc. 2nd Int. Conf. on Genetic Algorithms, pp.36-40, MIT, Cambridge, MA, 1987.

(Schaffer, 1989)- Schaffer J. D., R. A. Caruana, L. J. Eshelman and R. Das, "A study of control parameters affecting on-line performance of genetic algorithms for function optimization.", In Proc 3rd. Int. Conf. Genetic Algorithms, pp.51-60, J.Schaffer (ed.), Morgan Kaufman, San Mateo, 1989.

(Smith, 1993) Smith R. E., "Adaptively Resizing Populations: An Algorithm and Analysis", in Proc. 5th Int. Conf. Genetic Algorithm, Forrest S. ed., pp. 653, Morgan Kaufmann Publishers, Los Altos, CA, 1993

(Starkweather, 1991) Starkweather T., Whitley D., Mathias K.," Optimization Using Distributed Genetics Algorithms", in R. Moenner, B. Manderick, ed., Parallel Problem Solving from Nature,1,pp.176-185, Elsevier Science, 1991.

(Stender, 1994), Stender ed. "Parallel Genetic algorithms", IOS Press, 1994.

(Tanese, 1989) Tanese R., "Distributed Genetic Algorithm". in Proc. 3rd Int. Conf. Genetic Algorithms,pp.434-439, J. Schaffer (ed.), George Mason University, Morgan Kaufman, 1989.