ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/220662178

# An Experimental Study of Benchmarking Functions for Genetic Algorithms

**Article** · April 2002

DOI: 10.1109/ICSMC.2000.886604 · Source: DBLP

CITATIONS
183

READS
538

**2 authors:**

Jason Digalakis
National and Kapodistrian University of Athens
**39** PUBLICATIONS **495** CITATIONS

SEE PROFILE

Konstantinos G. Margaritis
University of Macedonia
**416** PUBLICATIONS **2,228** CITATIONS

SEE PROFILE

**Some of the authors of this publication are also working on these related projects:**




NORD - Neural Operations Research & Development View project

All content following this page was uploaded by Konstantinos G. Margaritis on 05 June 2014.

The user has requested enhancement of the downloaded file.

*Intern. J. Computer Math.*, 2002, Vol. 79(4), pp. 403–416

Taylor & Francis
Taylor & Francis Group

# AN EXPERIMENTAL STUDY OF BENCHMARKING FUNCTIONS FOR GENETIC ALGORITHMS

JASON G. DIGALAKIS and KONSTANTINOS G. MARGARITIS

*Department of Applied Informatics, University of Macedonia, Greece*

(*Received 3 March 2000; in final form 9 August 2000*)

This paper presents a review and experimental results on the major benchmarking functions used for performance control of Genetic Algorithms (GAs). Parameters considered include the eect of population size, crossover probability and pseudo-random number generators (PNGs). The general computational behavior of two basic GAs models, the Generational Replacement Model (GRM) and the Steady State Replacement Model (SSRM) is evaluated.

*Keywords:* Genetic Algorithms Performance, Generational Replacement Model, Steady-State Replacement Model. Population Size, Pseudo-random number generators, ISAAC PNG.

*C. R. Categories*: I 2.8, F 2.2

## 1. INTRODUCTION

Genetic algorithms (GAs) are search procedures based on principles derived from the dynamics of natural population genetics. These algorithms are based on the genetic processes of biological organisms. Many researchers of the field have frequently used various function groups in order to study the performance of GAs while a big part of the research involves the definition of control parameters and their potential adjustment. Up to now, no general conclusions are available concerning the optimum parameterization of operators.

In this paper we examine 14 different functions in order (a) to test specific parameter of GAs and (b) to evaluate the general computational behavior of GAs.

The authors are with the PDP Laboratory, Department of Applied Informatics, University of Macedonia, GR 54046, Thessaloniki, Greece (telephone: +3031891890, e-mail: jason@uom.gr, kmarg@uom.gr).

ISSN: 0020-7160 © 2002 Taylor and Francis Ltd DOI: 10.1080/00207160290017942

The available theoretical analysis on genetic algorithms does not offer a tool which could help in a generalized adjustment of control parameters, leaving the choice of the proper operators, parameters and mechanisms to depend on the problem's demands, and the experience and preferences of the researcher.

The structure of this paper as follows: Section 2 presents relevant papers of other researchers and Section 3 describes the functions used in this paper. In Section 4 we set out the methodology used and then we present the experimental results of our experiments. Finally we draw some general conclusions.

## 2. REVIEW OF BENCHMARKING FUNCTIONS

This section will briefly present the work of other researchers in order to establish the most frequently used functions in the performance analysis of various GA models and the suggested parameter sets.

De Jong [4] constructed a test environment of five problems in function minimization. His work translated Holland's theories into practical function optimization. De Jong used two performance measures for judging the genetic algorithms. First, he defined on-line performance as an average of all costs up to the present generation. It penalizes the algorithm for too many poor costs and rewards the algorithm for quickly finding where the lowest costs lie. Second, he defined off-line performance as the best cost found up to the present generation. The functions suggested by De Jong have certain characteristics, which depict many of the difficulties, found in optimization problems.

Following his study, a number of people suggested and tested various improvements to the basic genetic algorithm. Goldberg in [7] (pp. 106–120) nicely summarizes De Jong's work.

Grefenstette [8] used a metagenetic algorithm to optimize the on-line and off-line performance of genetic algorithms based on varying six parameters: population size $P_s$ crossover probability $C_p$, mutation probability $M_p$, generation gap $G$, scaling window, and whether elitism is used or not. The metagenetic algorithm used $P_s = 50$, $C_p = 0.6$, $M_p = 0.001$, $G = 1$, no scaling, and elitism. The best genetic algorithm for on-line performance had $P_s = 30$, $C_p = 0.95$, $M_p = 0.01$, $G = 1$, scaling of the cost function and elitism.

Scwefel's test set, consisting of 62 functions that cover an enormous diversity of different topologies is surely one of the most extensive function sets. ([18], pp. 319–354).

Extensive empirical studies concerning the performance of various crossover operators have been performed by Schaffer *et al.*, published in several papers which shed some light on the relation between performance, parameterization and

configuration of Genetic Algorithms ([3, 5, 17]). Schaer *et al.* used discrete sets of parameter values ($P_s = 10, 20, 30, 50, 100, 200$, $M_p = 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.10$, $C_p = 0.05$ to $0.95$ in increments of 0.10 and 1 or 2 crossover points) that had a total of 8400 possible combinations.

Baeck [1], emphasizes on two test functions (Sphere model, Rastrigin) for finding the total optimum and in the fast convergence. They had discrete sets of parameter values (probability crossover $= 0.6$, population size $= 50$, string length $= 32n$, mutation probability $= 0.001$, tournament selection and crossover two point).

Baeck in [2] formulates heuristic rules indicating that performance is decreasing both for large population size ($\mu > 200$) combined with large mutation probability $p_m > 0.05$ as well as for small population size ($\mu < 20$) combined with small mutation probability $p_m = 0, 002$. Finally, he describes several artificial test functions in Evolutionary Programming, including the sphere model, Rosenbrock's function, Ackley's function, the function after Fletcher and Powell and fractal function.

Patton *et al.* in [13] explore an alternative population-based form of adaptation for evolutionary computation, termed as Guided Gaussian Mutation (GGM), which is designed specifically as a localized search operator. In testing the potential eectiveness of the GGM operator, Patton selected a number of optimization test functions from the literature. Algorithms were tested with population sizes of 200 and 500. All test runs were tracked for 500 generations.

Michalewicz presented an empirical investigation of the performance of a hierarchy of evolution programs and indicated that generality grows at the expense of performance [11].

Salomon [15] analyzed the benefits of small mutation rates that are commonly used and focuses on the performance loss of typical GAs when applying a rotation to the coordinate system The theoretical analysis presented in his paper shows that most of the widely used test function have $n$ independent parameters and that, when optimizing such functions, many GAs scale with an O(nlnn) complexity.

Pohlheim [14] describes a number of test functions implemented for use with the Genetic and Evolutionary Algorithm Toolbox for Matlab. These functions are drawn from the literature on genetic algorithms' evolutionary strategies and global optimization.

In addition to the experiments mentioned so far, carried out by GA researchers, one can observe that there are also other suggestions as regards function sets from scientists of other relevant fields. We suggestively refer to one of the best know in the field of function optimization, by More, Garbow, Hillstrom [12] who suggested a total of 34 functions which has become a landmark for many researchers since then. More, Garbow, Hillstrom in [12] produced a relatively large collection of carefully coded test functions and designed very simple procedures for testing

the reliability and robustness of unconstrained optimization software. The functions they suggested are appropriate for comparative tests of various algorithmic methods used in optimization.

Finally, another important research carried out by Ford, Riley, and Freeman [6]. Involves an effort to evaluate the NAG parallel library. More specifically, they evaluate the solution improvement due the inclusion of parallelism offers to the finding of the total optimum, by providing the comparative results of the serial and parallel routines for solving the some problems. The functions they chose are also used in the work of More, Garbow and Hillstrom and are: (a) Extended Rosenbrock, (b) Watson.

Meysenburg in [10] examine to what degree the quality of the PRNG employed affects the performance of a simple GA including eleven test functions [20].

Finally, there are World Wide Web pages [20, 21], which contain information on several test functions that can be used to evaluate the performance of GAs. The test suite of functions [20], used to evaluate the performance of GAs during the IEEE International Conference on Evolutionary Computation in 1996, is particularly interesting. Many other test functions are used in global optimization, and a brief overview of some popular functions is also given in [19].

## 3. SET OF BENCHMARK FUNCTIONS

Taking the above under consideration and given the nature of our study, we concluded to a total of 14 optimization functions. Table I summarizes some of the

TABLE I Unimodal and multimodal functions (F1–F8)

| *Name* | *Function* | *Limits* |
|---|---|---|
| F1 | $f_1 = \sum_{i=1}^{2} x_i^2$ | $-5.12 \le x_i \le 5.12$ |
| F2 | $f_2 = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$ | $-2.048 \le x_i \le 2.048$ |
| F3 | $f_3 = \sum_{i-1}^{5} int.(x_i)$ | $-5.12 \le x_i \le 5.12$ |
| F4 | $f_4 = \sum_{i=1}^{30} (ix_i^4 + Gauss(0,1))$ | $-1.28 \le x_i \le 1.28$ |
| F5 | $f_5(x_i) = 0.002 + \sum_{j=1}^{25} \left(\frac{1}{j} + \sum_{i=1}^{2} (x_i - a_{ij})^6\right)$ | $-65536 \le x_i \le 65536$ |
| F6 | $f_6 = 10V + \sum_{i=1}^{10} (-x_i, \sin(\sqrt{\lvert x_i \rvert})), V = 4189.829101$ | $-500 \le x_i \le 500$ |
| F7 | $f_7 - 20A + \sum_{i=1}^{20} (x_i^2 - 10\cos(2\pi x_i)), A = 10$ | $-5.12 \le x_i \le 5.12$ |
| F8 | $f_8 = 1 + \sum_{i=1}^{10} \left(\frac{x_i^2}{4000}\right) - \prod_{i=1}^{10} \left(\cos\left(\frac{x_i}{\sqrt{i}}\right)\right)$ | $-600 \le x_i \le 600$ |

widely-used test functions. Next, we will try to make a short reference to these functions and to justify our selection of this specific set.

The first five test functions have been proposed by Dejong. All test functions reflect different degrees of complexity. Test functions F1–F4 are unimodal (*i.e.*, containing only one optimum), whereas the other test functions are multimodal (*i.e.* containing many local optima, but only one global optimum).

Sphere [F1] is smooth, unimodal, strongly convex, symmetric and it does not have any of the problems we have discussed so far.

Rosenbrock [F2] is considered to be difficult, because it has a very narrow ridge. The tip of the ridge is very sharp, and it runs around a parabola. Algorithms that are not able to discover good directions underperfom in this problem.

Step [F3] is the representative of the problem of flat surfaces. Function F3 is piecewise continuous step function. Flat surfaces are obstacles for optimization algorithms, because they do not give any information as to which direction is favorable. Unless an algorithm has variable step sizes, it can get stuck on one of the flat plateaus. The background idea of the step function is to make the search more difficult by introducing small plateaus to the topology of an underlying continuous function.

Quartic [F4] is a simple unimodal function padded with noise. The gaussian noise makes sure that the algorithm never gets the same value on the same point. Algorithms that do not do well on this test function will do poorly on noisy data.

Foxholes [F5] is an example of a function with many local optima. Many standard optimization algorithms get stuck in the first peak they find.

The Schwefel, Rastrigin, Griewangk [F6–F8] functions are typical examples of non-linear multimodal functions. Rastrigin's function [F7] is a fairly difficult problem for genetic algorithms due to the large search space and large number of local minima. Rastrigin has a complexity of $O(n\ln(n))$, where $n$ is the number of the function parameters. This function contains millions of local optima in the interval of consideration. The surface of the function is determined by the external variables $A$ and $\omega$, which control the amplitude and frequency of the modulation respectively. With $A = 10$ the selected domain is dominated by the modulation. The function 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.

Schwefel's function is somewhat easier than Rastrigin's function, and is characterized by a second-best minimum which is far away from the global optimum. In Schwefel function, $V$ is the negative of the global minimum, which is added to the function so as to move the global minimum to zero, for con-

venience. The exact value of $V$ depends on system precision; for our experiments $V = 4179.829101$.

Griewangk has a complexity $O(n\ln(n))$, where n is the number of the function's parameters. The terms of the summation produce a parabola, while the local optima are above parabola level. The dimensions of the search range increase on the basis of the product, which results in the decrease of the local minimums. The more we increase the search range, the flatter the function. Generally speaking, this is a good function for testing GA performance mainly because the product creates sub-populations strongly codependent to parallel GAs models.

Most algorithms have difficulties to converge close to the minimum of such functions, because the probability of making progress decreases rapidly as the minimum is approached. The rest of the functions (Table II) of the GA performance testing set chosen, are drawn from the field of mathematical programming and primarily cover the area of non-linear programming problems ([12, 9]).

One of the reasons we chose this set of problems is that one of the major developments in mathematical programming has to do with algorithms, a kind of programming which deals with the possibility to define the problem's solution in its various forms.

The possibility to solve such a problem is determined by two factors. Firstly, the establishment of an automatic solution method must be feasible, *i.e.* a method which solves all problems of the same category always following the same exact steps. Such a method, which also comprises genetic algorithms, is necessary in order for the method's programming and the problem's solving with the help of a computer to be possible. Further discussion of these test functions and their properties can be found in [15].

The standard functions of the available solving systems express in each case the optimization criterion. The dimensions of the systems are represented according to the constants or variables of the mathematical problem, while the limitations between the variables express the laws conditioning the system. The mathematical expression in many of our problems is such, that a big increase in the computational complexity of the mathematical system is entailed. The nature of each problem defines the feasibility and easiness with which the problem is solved from our computer. Genetic algorithms are also a systematic process, which is used for solving such problems, and it is thus used in our paper for a specific set of problems that tests their performance (see Tables I, II).

TABLE II Non linear squares problems

| *Name* | *Function definition* | *Dimensions* |
|---|---|---|
| F9 | $f_i = \sum_{j=2}^{n} (j-1)x_j t_i^{(j-2)} - \sum_{j=1}^{n} (x_j t_i^{(j-1)})^2 - 1$ | $2 \le n \le 31$, $m = 31$ |
| | where $t_2 = i/29, 1 \le i \le 29$ | |
| | $f_{30}(x) = x_1$, $f_{31}(x) = x_2 - x_1^2 - 1$ | |
| | *Standard starting point* : $x_0 - (0, \ldots, 0)$ | |
| F10 | $f_{2i} - 1(x) = 10(x_{2i} - x_{(i-1)}^2)$ | $n$ variable but even $m = n$ |
| | $f_{2i}(x) = 1 - x_{(2i-1)}$ | |
| | *Standard starting point:* $x_0 = (\xi_j)$ *where* | |
| | $\xi_{2j} - 1 = -1.2, \xi_{2j} = 1$ | |
| F11 | $f_1(x) = x_1 - 0.2$ | $n$ variable, $m = 2n$ |
| | $f_i(x) = \alpha^{[1/2]}\left(\exp\left[\frac{x_i}{10}\right] + \exp\left[\frac{x_{[i-1]}}{10}\right] - y_i\right), 2 \le i \le n$ | |
| | $f_i(x) = \alpha^{[1/2]}\left(\exp\left[\frac{x_{[i-n+1]}}{10}\right] - \exp\left[\frac{-1}{10}\right]\right), n \le i \le 2n$ | |
| | $f_{2n}(x) = \left(\sum_{j=1}^{n} (n-j+1)x_i^2\right) - 1$ where | |
| | $\alpha = 10^5, y_i = \exp\left[\frac{i}{10} + \exp\left[\frac{(i-1)}{10}\right]\right.$ | |
| | *Standard starting point:* $x_0 = (\frac{1}{2}, \ldots, \frac{1}{2})$ | |
| F12 | $f_1(x) = 10^4\, x_1 x_2 - 1$ | $n = 2, m = 2$ |
| | $f_2(x) = \exp[-x_1] + \exp[-x_2] - 1.0001$ | |
| | *Standard starting point:* $x_0 = (0, 1)$ | |
| F13 | $f_1(x) = \exp\left[\frac{-\lvert y_i mi\, x_2 \rvert^{(x_3)}}{x_i}\right] - t_i$ | $n = 3, n \le m \le 100$ |
| | $t_i = i/100$ | |
| | $y_i = 25 + (-50\ln(t_i))^{(2/3)}$ | |
| | *Standard starting point:* $x_0 = (5, 2.5, 0.15)$ | |
| F14 | $f_{(4i-3)}(x) = x_{(4i-3)} + 10x_{(4i-2)}$ | $n$ variable but a multiple of 4 |
| | $f_{(4i-3)}(x) = 5^{(1/2)}(x_{(4i-1)} - x_{4i})$ | $m = n$ |
| | $f_{(4i-1)}(x) = (x_{(4i-2)} - 2x_{(4i-1)})^2$ | |
| | $f_{(4i)}(x) = 10^{(1/2)}(x_{(4i-3)} - x_{(4i)})^2$ | |
| | *Standard starting point:* $x_0 = (\xi_i)$ | |
| | $\xi_{4j} - 3 = 3, \xi_{4j} - 2 = -1, \xi_{4j} - 1 = 0, \xi_{4j} = 1$ | |

## 4. METHODOLOGY

The main performance metric is the efficiency of the genetic algorithm, (*i.e.* the ability to reach an optimum is increased the number of populations we whether the number of GA iterations required to find a solution decreased). In order to facilitate an empirical comparison of the performance of GAs, a test environment for these algorithms must be provided in the form of several objective functions $f$.

We used the PGAPack [9] library and we tested both population replacement models available in the field of serial GAs. The first, the generational replacement genetic algorithm (GRGA), replaces the entire population each generation and is the traditional genetic algorithm. The second variant is the steady state genetic algorithm (SSGA) in which only a few structures are generated in each iteration ($\lambda = 1$ or 2) and they are inserted in the current populations ($Q$=$P(t)$).

The criterion we used for ending the algorithm stipulates the following:

> "The executions stop only if after a certain number of repetitions (in this paper 100 repetitions) the best individual is not substantially mutated."

The population size affects both the overall performance and the efficiency of GAs. Two population replacement schemes are used. The first, the generational replacement model (GRM), replaces the entire population each generation and is the traditional genetic algorithm. The second, the steady state replacement model (SSRM), typically replace as only a few strings each generation.

Two PNGs have been tested, *i.e.* (a) PGARUniform and (b) ISAAC (Indirection, Shift, Accumulate, Add, and Count). Each time PGAPack is run, unique sequence of random numbers is used. In the package, the initial population of individuals is created by selecting each bit uniformly at random. For each bit position of each individual in the population, a call is made to a PNG to obtain a real number in the range [0, 1).

We obtained the ISAAC PNG from the World Wide Web [22]. It is intended to be a PNG worthy of use in cryptographic applications. The claimed period for the PNG ranges from $2^{40}$ to $2^{8295}$, depending on the seed value. Then, we found the online value for each PNG and test function, and then compared this value with the true optimal value for each function.

The crossover in our experiments is applied with six probability values $p_c \in [0.6, 0.95]$ with step 0.05 [4, 8, 17], and its application or non-application is defined by a random number generator. When it is not applied, the ospring is considered as an exact copy of one of the parents with equal probability. Mutation on the other hand is applied with a $1/2l$ probability where $l$ is the alphanumerical string [2, 9].

## 5. EXPERIMENTAL RESULTS

Using the method described in the previous section we tried to determine the effect on the performance of a GA a) of the population size, b) of the population replacement model selection, c) of the crossover probability of the parents and d) of the pseudo-random generator. For our experiments we used a Silicon-Graphics (Origin 200) [MIPS RISC R10000 64BIT, 128 MB memory] while the implementation of our programs was carried out with the help of the PGAPack library.

More specifically, the total of our parameters was:

1. Population sizes: (50, 100, 150, 200, 250, 300, 350, 400).
2. Population replacement models: a) Generational Replacement Model, b) Steady State Replacement.
3. Crossover probability: [0.6–0.95] with step 0.05.
4. Mutation probability: $1/2l$ where $l$ the individual's alphanumerical string.
5. Crossover mechanism: uniform crossover.
6. Algorithm ending criteria: the executions stop only if after 100 repetitions the best individual has not substantially changed.
7. Pseudorandom generators: ISAAC, PGARuniform
8. Mutation mechanism: Gaussian.
9. Selection: tournament selection.
10. Fitness function: linear ranking.

We examined the average, best and worst values we took for each function in a total of 1792 executions of the GA ($8\times2\times8\times14$). The main performance metric was the efficiency of the genetic algorithm model we used. This was studied by choosing a large set of test problems and trying to characterize on different problems profile how well the GA performed. Table III shows that an SSGA

TABLE III The effect of population size

| *Ps* | *50* | *100* | *150* | *200* | *250* | *300* | *350* | *400* |
|---|---|---|---|---|---|---|---|---|
| F1 | 5.25 | 3.85 | 1.81 | 0.6 | 0.8 | 0.7 | 0.75 | 0.68 |
| F2 | 4.8 | 3.74 | 1.61 | 0.4 | 0.5 | 0.4 | 0.45 | 0.48 |
| F3 | 4 | 3.8 | 1.6 | 0.6 | 1.74 | 1.76 | 1.78 | 1.8 |
| F4 | 12.35 | 6.21 | 2 | 1.2 | 1.95 | 1.96 | 1.97 | 1.98 |
| F5 | 12 | 5 | 1.8 | 1 | 1.86 | 1.87 | 1.88 | 1.89 |
| F6 | 6900 | 5700 | 5380 | 4400 | 4316.25 | 4318.25 | 4320.25 | 4322.25 |
| F7 | 65 | 33.6 | 10 | 12.2 | 10.06 | 10.07 | 10.35 | 10.07 |
| F8 | 9.8 | 4.2 | 1.9 | 2 | 1.96 | 1.97 | 1.98 | 1.99 |
| F9 | 39.25 | 11.15 | 4.7 | 1.01 | 2.15 | 2.05 | 2.15 | 2.16 |
| F10 | 22.3 | 12.1 | 2.3 | 1.8 | 2.37 | 2.39 | 2.41 | 2.43 |
| F11 | 6 | 3.78 | 1.4 | 0.42 | 1.47 | 1.44 | 1.48 | 1.42 |
| F12 | 7.2 | 4.86 | 1.8 | 0.85 | 1.76 | 1.73 | 1.85 | 1.88 |
| F13 | 25.72 | 14.56 | 5.8 | 1.1 | 2.07 | 2.06 | 2.07 | 2.07 |
| F14 | 15 | 4 | 1.2 | 0.5 | 1.25 | 1.25 | 1.25 | 1.25 |

performance improves when the population size is changed. A small size provides an insufficient sample, which makes them perform poorly. A large population size will undoubtedly raise the probability of the algorithm performing an informed and directed search. However, a large population requires more evaluations per iteration or generation possibly resulting in the method developing redundant controllers and becoming very slow especially when implemented serially. We found a populations of $200 \sim 250$ to be optimal for our task, but in general we assert that the optimum population size depends on the complexity of the domain. When the population size is reduced to half, no significant performance different is observed as evident from the figures. This success with smaller population is because of the genetic variability which exists within each individual and in the population is sufficient to adapt in this environmental change. Of course, to achieve this level of performance the SSGA needed more memory space to keep redundant information as compared to the same size population.

Table IV shows the average number of generations taken be the GRM and SSRM on the occasions where it found the optimal value (within $\pm 0.001$) for the test function in question. These results apply to the Generational Replacement model and Steady-State Replacement model. It should be noted that the absolute number of generation is important performance.

In [2] it is argued to that crossover alone is sufficient to find the global optimum of separable, multimodal functions within O(nlnn) time. Table V shows the

TABLE IV The number of generations for convergence

| *Function* | *F1* | *F2* | *F3* | *F4* | *F5* | *F6* | *F7* | *F8* | *F9* | *F10* | *F11* | *F12* | *F13* | *F14* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SSRM | 25 | 232 | 25 | 223 | 20 | 35 | 450 | 100 | 350 | 300 | 310 | 172 | 430 | 456 |
| GGRM | 20 | 88 | 18 | 158 | 17 | 22 | 350 | 80 | 230 | 280 | 600 | 130 | 397 | 351 |

TABLE V Results for different crossover probabilities

| *Pc%* | *60* | *65* | *70* | *75* | *80* | *85* | *90* | *95* |
|---|---|---|---|---|---|---|---|---|
| F1 | 1.25 | 0.850 | 0.600 | 0.610 | 0.644 | 0.727 | 0.737 | 0.733 |
| F2 | 0.8 | 0.740 | 0.400 | 0.405 | 0.410 | 0.510 | 0.420 | 0.432 |
| F3 | 1.2 | 0.8 | 0.6 | 0.4 | 0.488 | 0.508 | 0.528 | 0.548 |
| F4 | 2.35 | 1.21 | 1.2 | 0.8 | 0.7 | 0.71 | 0.72 | 0.73 |
| F5 | 5.2 | 2.1 | 1.4 | 0.6 | 0.61 | 0.62 | 0.63 | 0.64 |
| F6 | 5100 | 4500 | 4400 | 4380 | 4315 | 4317 | 4319 | 4321 |
| F7 | 15 | 13.6 | 12.2 | 8.8 | 8.81 | 8.82 | 9.1 | 8.82 |
| F8 | 1.8 | 1.2 | 2 | 0.7 | 0.71 | 0.72 | 0.73 | 0.74 |
| F9 | 1.25 | 1.15 | 1.01 | 0.7 | 0.9 | 0.8 | 0.9 | 0.91 |
| F10 | 2.3 | 2.1 | 1.8 | 1.1 | 1.12 | 1.14 | 1.16 | 1.18 |
| F11 | 0.8 | 0.78 | 0.42 | 0.2 | 0.22 | 0.19 | 0.23 | 0.17 |
| F12 | 3.2 | 1.86 | 0.85 | 0.6 | 0.51 | 0.48 | 0.6 | 0.63 |
| F13 | 1.72 | 1.56 | 1.1 | 0.8 | 0.82 | 0.81 | 0.815 | 0.817 |
| F14 | 1.2 | 1.1 | 0.5 | 0.56 | 0.54 | 0.52 | 0.51 | 0.5 |

results of experiments we did to compare eight values of crossover probability. Crossover rates suggest that a high rate, which disrupts many strings selected for reproduction is important in a small population. The same performance difference does not appear, however, between tests using different crossover probabilities. A higher crossover rate tends to disrupt the structures selected for reproduction at a high rate, which is important in a small population, since high performance individuals are more likely to quickly dominate the population.

The population size affects both the ultimate performance and the efficiency of Gas. A large population requires more evaluations per generation, possibly resulting in an unacceptably slow rate of convergence. This emfasizes that population size, if increased sufficinetly, can offset differences in crossover propability. On the other hand, for small population size the choice of crossover operators plays a more dominant role than.

Using the method described in Section 4 we were able to determine to what extend the quality of PNG affects the performance of the SSRM and GRM. Figures 1 and 2 show, in picture form, the similarity in performance between the GA driven by the best PNG and the GA driven by one of the worst PNGs. Specifically, it compares the performance of the GA driven by the best PNG (ISAAC) versus the performance of the GA driven by the worst PNG (PGAR-Uniform). It plots the online performance measure value found, expressed as a percentage of the optimal value, achieved by the GA across all fourteen of the test functions.

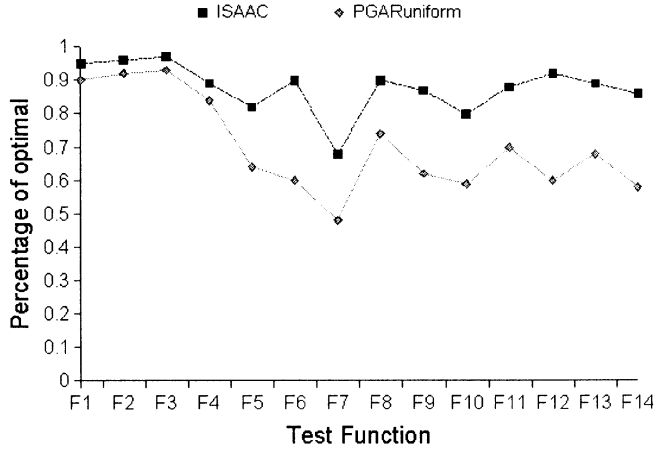


FIGURE 1 GA online percentage of optimal value, (GGRM model).

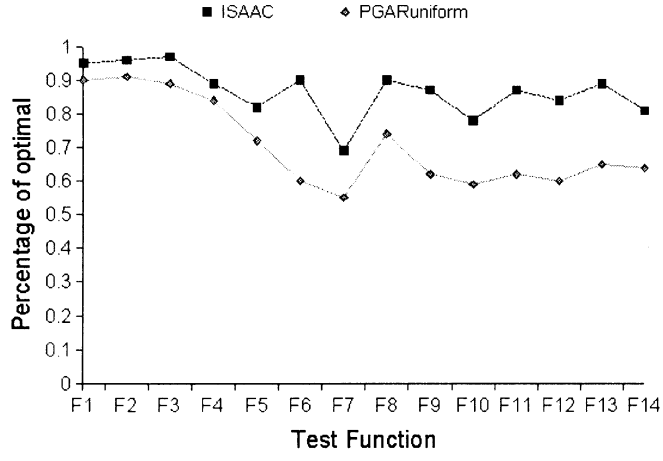


FIGURE 2 GA online percentage of optimal value, (SSRM model).

## 6. CONCLUSIONS

Genetic Algorithms, as have been discussed, provide a very good conceptional framework for optimization inspired by nature, but theoretical questions and algorithmic considerations discussed in this paper suggest that the set of functions on which GAs yield optimal performance converges to a set of benchmarking functions.

Experiments were performed to search for the optimal GAs for a given set of numerical optimization problems. Finding an appropriate and represented set of test problems is not easy task, since the particular combination of properties represented by any set of test functions does not allow for generalized performance statements. From the fourteen objective functions tested empirically and the results reported here we can give only a statement of limited generality about the efficiency of GA. Test functions that yield another ordering of the methods are likely to exist. The experimental data also suggest that, while it is possible to GA control parameters, very good performance can be obtained with a varying range of GA control parameter settings.

For a computationally bound GA properly sizing the population is important because both an oversized or undersized population reduces the final solution quality. Our results clearly show that the absence of crossover leads to poor results while the presence of even a small crossover rate immediately improves our results. Generally speaking, we can argue that a population size of $200 \sim 250$ and a crossover probability percentage of $70 \sim 75\%$ bring satisfactory results.

The use of large population size values does not necessarily entail in all cases better results and consequently increases the computational time.

By using the Generational Replacement Model we can be more certain that the best selected parents of the current generation will participate in the reproduction of offsprings compared to the Steady-State model. This does not mean, however, that the Steady-State Replacement Model does not make proper use of the parent selection process, but simply that the possibility for a greater number of parents with good characteristics to be chosen is greater with the generational replacement model. This entails a computational cost since in all 14 functions of our paper we found a relatively good solution with a greater number of generations in the generational replacement model compared to the steady state replacement.

An the issue we examine in this paper is to what degree the performance of the PNG employed effects the performance of the GA. For some test functions, ISAAC PNG drove the GA to its maximum value in fewer generations. In fact, when the GA did produce an individual with optimal value, there was no statistical difference between the number of generations required to produce the individual when the GA was driven by a good PNG (ISAAC) and when it was driven by PGARUniform. When optimizing multimodal functions, a PNG has even more severe consequences. Therefore, the experiments described above are important in that they identify approximately optimal parameter settings for the four performance measures considered.

## *References*

[1] Baeck, T. (1991). "Optimization by Means of Genetic Algorithms". In *Internationales Wissenschaftliches Kolloquium* (pp. 1–8). Germany: Technische Universit".

[2] Baeck T. (1996). *Evolutionary Algorithms in Theory and Practice*, New York: Oxford University Press, pp. 138–159.

[3] Caruana, R. A., Eshelman, L. J. and Schaer, J. D. (1989). "Representation and hidden bias II: Eliminating defining length bias in genetic search via shue crossover". In N. S. Sridharan, editor, *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 750–755. Morgan Kaufmann Publishers, San Mateo, CA.

[4] De Jong, K. A. (1975). "An analysis of the behaviour of a class of genetic adaptive systems", University of Michigan, Ann Arbor. (University Microfilms No. 76-9381).

[5] Eshelman, L. J., Caruana, R. A., and Schaer, J. D. (1989). "Biases in the crossover landscape", In *Proceedings of the 3rd International Conference on Genetic Algorithms and Their Applications*, Morgan Kaufmann Publishers, San Mateo, CA, pp. 10–19.

[6] Ford, R. W. and Riley, G. D. (1997). "The development of Parallel Optimization Routines for the NAG Parallel library", *High Performance Software for Nonlinear Optimisation Conference*, Naples.

[7] Goldberg, D. E. (1989). "*Genetic algorithms in search, optimization and machine learning*", New York: Addison-Wesley, pp. 40–45.

[8] Grefenstette, J. J. (1986). "Optimization of control parameters for genetic algorithms", in Sage, A. P. (Ed.), *IEEE Transactions on Systems, Man, and Cybernetics*, Volume SMC-16 (pp. 122–128). New York: IEEE

[9] Levine, D. (1996). "Users Guide to the PGAPack Parallel Genetic Algorithm Library", Argonne National Laboratory, Mathematics and Computer Science Division, pp. 19–28.
[10] Meysenburg, M. M. (1997). "The eect of the quality of Pseudo-Random Number Generator Quality on the Performance of a Simple Algorithm", College of Graduate Studies, University of Idaho.
[11] Michalewicz, Z. (1993). "A hierarchy of evolution programs: An experimental study". Evolutionary Computation, **1**(1) pp. 51–76.
[12] More, J. J., Garbow, B. S., and Hillstrom, K. E. (1981). "Testing Unconstrained Optimization Software, {ACM} Transactions on Mathematical Software, vol 7, pp. 17–41.
[13] Patton, A. L. , Dexter, T., Goodman, E. D. and Punch, W. F. (1998). "On the Application of Cohort-Driven Operators to Continuous Optimization Problems using Evolutionary Computation", in *Lecture Notes in Computer Science*, vol 1447, pp. 671–682, Springer - Verlag Inc.
[14] Pohlheim, H. (1997). "GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB", http://www.systemtechnik.tuilmenau.de /~pohlheim/GA_Toolbox.
[15] Salomon. R. (1995). "Reevaluating Genetic Algorithm Performance under Coordinate Rotation of Benchmark Functions", BioSystems vol. 39, pp. 263–278, Elsevier Science.
[16] Salomon, R. (1997). "Improving the Performance of Genetic Algorithms through Derandomization." In G. Pomberger (Ed.), *Software - Concepts and Tools*, vol 18, pp. 175–184, Springer-Verlag, Berlin.
[17] Schwefel, J. D., Caruna, R. A., Eschelman, L. J. and Das, R. (1989). "A study of control parameters aecting online performance of genetic algorithms for function optimization", in J. D. Schaer (Ed.), *Proceedings of the 3rd International Conference on Genetic Algorithms and Their Applications*, pp. 61–68, Morgan Kauman Publishers, San Mateo, CA.
[18] Schwefel, II. P. (1977). Numerische optimierung von Computer-Modellen mittels der Evolutionsstrategie, In Interdisciplinary Systems Research vol. 26, pp. 319–354, Birkhäuser, Basel
[19] Törn, A. and Zilinskas, A. (1991). Global Optimization, volume 350 of Lecture Notes in Computer Science, Springer, Berlin.
[20] First international contest on evolutionary optimization. http://iridia.ulb.ac.be/langerman/ICEO.html, 1997.
[21] Optimization test functions - L. Lazauskas.http://www.maths.adelaide. edu.au/Applied.llazausk/alife/realfopt.htm, 1997.
[22] Random number generators, http://ourworld.c ompuserve.com/home pages/bob_jenkins/; randomnu.htm, 1997.

View publication stats