A Statistical Genetic Algorithm

Angel Kuri M.
akm@pollux.cic.ipn.mx
Centro de Investigación en Computación
Instituto Politécnico Nacional
Zacatenco
México 07738, D.F.

Abstract

A Genetic Algorithm which works by statistically mimicking the genetic operations of a traditional genetic algorithm is presented. This Statistical Genetic Algorithm (STA) is tested vs. a set of 8 "difficult" functions and is shown to compare favorably with its more traditional non-statistical counterparts. A qualitative description of the algorithm is presented which conceptually explains its behavior.

Keywords
Genetic Algorithm, genetic operators, statistical, optimization.

1. Introduction
In the past few years the evolutionary computation community and, in particular, the Genetic Algorithm (GA) community, has paid much attention to the problem of determining the best ways to achieve optimization of complex numerical functions. Special attention has been given to those functions which seem to be difficult for the GA to optimize (GA-hard) and the reasons why this is so (see, for instance, [1], [2], [3]). In this paper we propose and analyze yet another variation of a GA. This particular form of a GA has been suggested by Enrique Díaz Infante [4] and takes advantage of the fact that the average behavior of traditional Simple Genetic Algorithm (SGA) may be achieved without actually applying the genetic typical operators (selection, crossover and mutation) but, rather, statistically simulating their behavior. In order to test the past assertion we defined four variations of the proposed STA: 1) A statistical GA with 1-point crossover (the counterpart of the SGA) which we denote by STA1P; 2) A statistical GA with uniform crossover (the counterpart of the SGA with uniform crossover) which we denote by STAUN; 3) An elitist statistical GA with 1-point crossover (the counterpart of an elitist SGA) which we denote by STE1P; 4) An elitist statistical GA with uniform crossover (the counterpart of an elitist SGA with uniform crossover) which we denote by STEUN. In part 2 of the paper we define what we mean by these 4 variations. In part 3 we describe 8 functions we have chosen to test the behavior of the 4 variations. In part 4 we describe the results of a set of runs designed to establish the performance of the four variations relative to their more traditional counterparts. Finally, in part 5 we offer some conclusions and point to further lines of research.

2. A Statistical GA

2.1. Conventions
In what follows, the following applies:

i) Each run consists of G generations.

ii) Each population is assumed to consist of N individuals.

iii) Each of any individual's genome is assumed binary.

iv) The length of the genome is l.

v) Each of the genomes represents an m-dimensional real vector $\vec{x} = (x_1,\ldots,x_m)$.

vi) Each real number x_i has a fixed point format of the form *S/I/D* where *S* is the sign bit, *I* is a set of bits representing the integer part of the number and *D* is a set of bits representing the decimal part of the number. For instance, if $|I| = 2$ and $|D| = 5$ the weighted binary string *11101000* represents the number *-3.25*. A genome is a concatenated collection of m such numbers.

vii) Every number x_i is encoded in Gray [5].

2.2. Algorithm STAUN
a) Set $i \leftarrow 0$.

b) Generate the i-th population Π_i.

c) Obtain the fitness (F_j) of each of the individuals ($1 \le j \le N$).

F_j is calculated from:

$$F_j = f_j + \mathrm{K} \qquad (2.a)$$

where

f_j is the original function evaluated for the *j-th* genome and $\mathrm{K} = \frac{1}{N}\sum_j |f_j| + |\min_j(f_j)|$.

d) Obtain the relative fitness $\Phi_j = \frac{F_j}{\sum_j F_j}$.

e) Every bit *k* of the population's genome is assigned a probability P_k as follows:

$$P_k = \sum_j \Phi_j b_{jk} \quad k = 1, \ldots, l \qquad (2.b)$$

where b_{jk} denotes the *k-th* bit of the *j-th* individual. Notice that P_k actually represents the weighted expected number of times that bit *k* will take the value *1* as a function of the fitness of the *i-th* population.

f) Repeat *N* times:

i) Set $j \leftarrow 1$.

ii) Generate a uniformly distributed number $0 < \rho_j, \rho_k \leq 1$.

iii) If $\rho_j > 0.005$

$$b_j = \begin{cases} 0 \ \textit{if} \ \rho_k > P_j \\ 1 \ \textit{if} \ \rho_k \leq P_j \end{cases}$$

otherwise

$$b_j = \begin{cases} 0 \ \textit{if} \ \rho_k \leq P_j \\ 1 \ \textit{if} \ \rho_k > P_j \end{cases}$$

iv) Set $j \leftarrow j + 1$.

v) If $j < l$ go to (ii)

vi) An individual's genome is composed by concatenating the *l* bits from the procedure above.

g) Set $i \leftarrow i + 1$.

h) If $i > G$ end algorithm; otherwise proceed with step (c).

□

2.3. Qualitative Description

Algorithm STAUN starts by generating Π_0 population's individuals randomly. That is, each of the *l* bits in every individual's genome is assigned the value *0* or *1* randomly with *P(0) = P(1) = 0.5*. The fitness of the *j-th* individual is calculated from the objective function and normalized as per (2.a) to ensure a positive fitness. Given this, it is easy to determine *j*'s relative fitness which, immediately, induces a partial ordering in the population according to the value of Φ_j. Once this is done, we are able to determine what may be aptly called a *probabilistic genome* (PG). In this genome, the probability that the *j-th* bit of the genome attains a value of *1* is derived from equation (2.b). In actuality what we are doing is defining a set of probability distribution functions (pdfs), 1 for each of the *l* bits in the genome. These pdfs are Bernoulli distributed and, initially, may have rather large variances (σ^2). Given the above, every new population is generated by sampling from the *j-th* distribution to compose its new *N* individuals. The *i-th* population consists of individuals that respond to the average behavior of the $(i-1)-th$. However, because of statistical fluctuations, neighboring regions of the problem's landscape are constantly being explored. Since the *i-th* SG is derived from the new fitnesses, the *(i+1)-st* population is closer to the solution. A new set of *l* probability distribution functions replaces the older one. Every new population is also Bernoulli distributed but with an increasingly small σ although this process is, in general, non-monotonic. Eventually (one hopes) the pdfs of the final population will have a Bernoulli distribution with $\sigma \approx 0$, implying convergence. In a strict sense, the STA avoids the need to include explicit mutation provisions. However, preliminary tests showed that premature convergence is avoided if such provisions are made. From the point of view of encoding, the whole process may be seen as a search for a crisp encoding of the solution with a set of fuzzy bits. The bits of the initial generations are quite fuzzy and each bit is progressively de-fuzzyfied in consecutive generations. In the case of an STA the problem of GA-hardness translates to the problem of not targeting to intermediate pdfs with small σs.

2.3.1. Statistical Operators

Although the STAUN algorithm does not explicitly include any of the so-called genetic operators, there is an equivalence between the proposed algorithm's workings and such operators. In fact, we have defined 4 variations (to be discussed in the sequel) which depend on this equivalence.

2.3.1.1. Selection

In an SGA individuals are selected proportionally according to the quotient $\frac{f_i}{\bar{f}}$. In the STAUN proportional selection stems from the way the distributions for the bits of the SG are calculated. Proportionality is tacit in the fact that better individuals collaborate with higher weight in the bits' distributions of the SG.

2.3.1.2. Crossover

In an SGA crossover is of the "1-point" sort, meaning that 1 point in the individual's genome is selected randomly and the two segments defined by this locus are interchanged. Uniform crossover (where a set of *l/2* bits are randomly selected to be interchanged) yields a different behavior of the algorithm and the STAUN algorithm does follow this behavior rather than that of the 1-point sort (hence the nomenclature: STA*UN*). This may be clearly understood since every bit's statistical distribution is sampled with the same probability.

2.3.1.3. Mutation

In an SGA the exploration new areas of the problem's landscape is achieved via the purposeful obliteration of randomly selected bits. This happens with, typically, low probability. In STAUN the strategy adopted (outlined in step (f) of the algorithm) to achieve this effect is to adopt the exact dual of the SG's distribution with a very low probability. We have chosen $P_m = 0.005$.

2.3.2 Variations

In order to test the operation of the STA we considered 4 variations of a traditional GA: a) The SGA as defined by Holland [6] (denoted as SGA1P); b) An SGA where 1-

point crossover is replaced by uniform crossover (denoted as SGAUN); c) An SGA where the best individual in the *i-th* generation is preserved. This *elitist* GA is denoted as SGE1P; d) An SGA with elitism and uniform crossover (denoted as SGEUN). We also considered their statistical counterparts which we denote as a) STA1P, b) STAUN, c) STE1P and d) STEUN. The STA variations work as described above with the obvious preservation of the best individual in STE1P and STEUN. The 1-point crossover operator was implemented in STA1P and STE1P by assigning a weight to the *j-th* bit in the SG according to $w_j=(l-j+1)/l$. This is consistent with the fact that in 1-point crossover the bits are more likely to be interchanged if they are close to the rightmost bit in the genome.

3. Objective Functions

8 problems, along with their objective functions were defined to test the behavior of all 8 variations described above. In what follows we briefly describe the problems and make some light remarks.

Problem 1:

Maximize

$$f = x_1 + |\sin(32x_1)| - |\cos(11x_2)|$$

Subject to:

$$0 \le x_1 \le \pi$$
$$0 \le x_2 \le \pi$$

Solution:

$$f(x_1,x_2) \approx 4.0930$$
$$x_1 \approx 3.0935$$
$$x_2 \approx 2.1420$$

Problem 2:

Maximize

$$f(x_1,x_2) = -(x_1-5)^2-(x_2-3)^2$$

Solution:

$$f(x_1,x_2) = 0$$
$$x_1 = 5.0$$
$$x_2 = 3.0$$

Problem 3:

Minimize

$$f(x_1,x_2) = x_1^2 + x_2^2 + x_1x_2 + 2x_1 + 2x_2 - 13$$

Subject to:

$$r_1(x_1,x_2) = x_1^2 + x_2^2 + x_1 + x_2 - 8 \ge 0$$
$$r_2(x_1,x_2) = x_1x_2 + x_1 + x_2 - 5 \ge 0$$

Solution:

$$f(x_1,x_2) \approx 0.0009879$$
$$x_1 \approx 1.0005490$$
$$x_2 \approx 1.9996705$$

and

$$r_1(1.0005490, 1.9996705) \approx 10^{-8}$$
$$r_2(1.0005490, 1.9996705) \approx 10^{-4}$$

Problem 4:

Minimize

$$f(x_1,x_2) = 3x_1^2 + 2\sin(x_1)\cos(x_2) + 2x_2^2 + \cos(x_1) + \sin(2x_2) - 27.5$$

Subject to:

$$r_1(x_1,x_2) = 3x_1^2 + 2\sin(x_1)\cos(x_2) - 10.2 \ge 0$$
$$r_2(x_1,x_2) = 2x_2^2 + \cos(x_1)\sin(2x_2) - 17.3 \ge 0$$

Solution:

$$f(x_1,x_2) \approx -1.0037418244$$
$$x_1 \approx 1.9981788993$$
$$x_2 \approx 2.9264232963$$

and

$$r_1(1.9981788993, 2.9264232963) \approx 10^{-5}$$
$$r_2(1.9981788993, 2.9264232963) \approx 10^{-5}$$

Remarks. Problems 3 and 4 are an attempt to find a general (evolutionary) algorithm for the solution of simultaneous non-linear equations. The method is simple. We obtain a linear combination of the n equations to be solved (in this case $n = 2$). Here we simply added the equations. Then we establish the need to minimize the composite function thusly obtained and impose the need to satisfy the requirement that the original equations are ≥ 0. In trying to minimize de composite function without violating the constraints, such constraints are also minimized and the original problem is solved. Notice that we could have equally well defined the problem in terms of maximization. Then we would have imposed the condition that the constraints remain ≤ 0.

Problem 5:

Minimize

$$f(x_1,x_2) = 100(x_2 - x_1^2)^2 + (1-x_1)^2$$

Subject to:

$$x_1 + x_2^2 \ge 0$$
$$x_1^2 + x_2 \ge 0$$
$$-\frac{1}{2} \le x_1 \le +\frac{1}{2}$$
$$-1 \le x_2 \le +1$$

Solution:

$$f(x_1,x_2) \approx 0.2500000$$
$$x_1 \approx 0.5000049$$
$$x_2 \approx 0.2500051$$

Problem 6:

Minimize

$$f(x_1,x_2) = -x_1 - x_2$$

Subject to:

$$x_2 \le 2x_1^4 - 8x_1^3 + 8x_1^2 + 2$$
$$x_2 \le 4x_1^4 - 32x_1^3 + 88x_1^2 - 96x_1 + 36$$
$$0 \le x_1 \le 3$$
$$0 \le x_2 \le 4$$

Solution:

$$f(x_1,x_2) \approx 5.466036$$
$$x_1 \approx 2.339889$$
$$x_2 \approx 3.126146$$

Problem 7:

Minimize

$$f(x_1,x_2) = (x_1 - 10)^3 + (x_2 - 20)^3$$

Subject to:

$$(x_1 - 5)^2 + (x_2 - 5)^2 - 100 \ge 0$$
$$-(x_1 - 6)^2 - (x_2 - 5)^2 + 82.81 \ge 0$$
$$13 \le x_1 \le 100$$
$$0 \le x_2 \le 100$$

Solution:

$$f(x_1,x_2) \approx -6961.668579$$
$$x_1 \approx 14.095059$$
$$x_2 \approx 0.843090$$

Problem 8:

Minimize

$$f(x_1) = \left(e^{x_1} - 1\right)^2 + \left(\frac{1}{\sqrt{1 + x_1^2}} - 1\right)^2$$

Solution:

$$f(x_1) \approx 10^{-6}$$
$$x_1 \approx -3 \times 10^{-8}$$

4. Experiments

We conducted a set of experiments as follows:

a) Every one of the 8 variations was run 5 times with different seeds for the random number generator for each of the 8 objective functions. Thus, a total of 320 simulations were run.

b) In all cases we set $N = 50$ and $G = 50$.

c) In problems 1, 3, 4, 5, 6 and 7 (which are constrained) we used a death penalty for those individuals not complying with the constraints. Fitness of an individual is determined from

$$f_i(\vec{x}) = \begin{cases} f_i(\vec{x}) \ \textit{if feasible} \\ K - \sum_{i=1}^{s} \left(\frac{K}{m}\right) \textit{otherwise} \end{cases} \quad (4.a)$$

where $K = 10^9$, s = number of satisfied constraints.

d) The behavior of the algorithms was registered for each of the 50 generations. We calculated the relative error (ε_R) for the *i-th* generation from

$$(\varepsilon_R)_i = \frac{\max(|best_i|, |Best|) - \min(|best_i|, |Best|)}{|Best|} \quad (4.b)$$

where $best_i$ is the best individual in generation i and *Best* is the known solution.

e) The value of *Best* was determined by running an *eclectic* [7] GA for an appropriate number of generations.

4.1. Results

In the following figures we show the results of the conducted experiments. The $(\varepsilon_R)_i$ have been plotted every 5 generations. The vertical axis corresponds to the error while the horizontal axis corresponds to the generation. F_Gxx denotes the *xx−th* generation. It should be noted that in these experiments rarely was the best value of the objective function reached. This is of no importance because it was not our intent to test the speed of convergence but, rather, the relative performance between the different variations. Also, this behavior is to be expected when the number of allotted generations is as small as in here ($G = 50$).

In figure 1 we show ε_R for SGE1P and STE1P algorithms in problem 1. Both GAs reach an acceptable ε_R but the standard GA outperforms the statistical GA in the final stages of the algorithm.

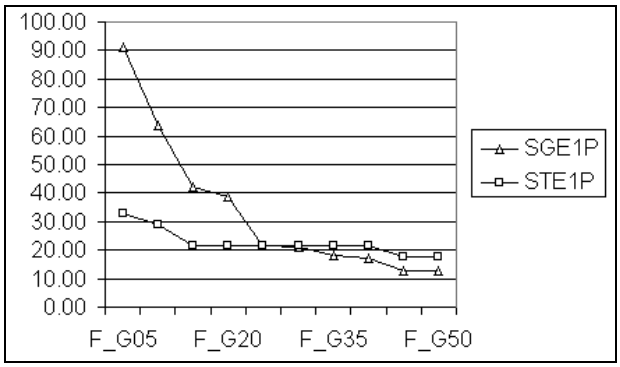

Figure 1. Elitist 1-point Crossover for Problem 1.

In figure 2 we show the RE for SGEUN and STEUN. Notice that SGEUN's performance is worse than SGE1P's whereas STEUN is markedly better than STE1P.

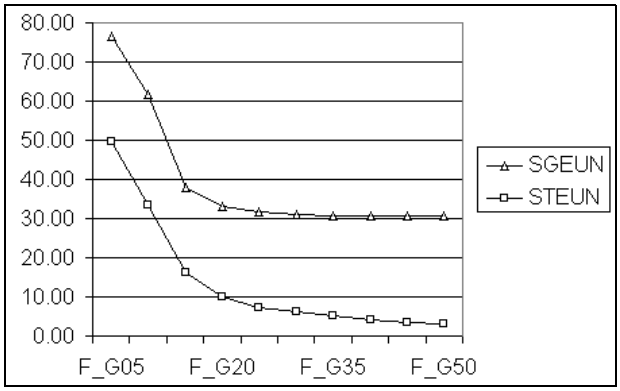

Figure 2. Elitist Uniform Crossover for Problem 1.

In this case, the fact that ε_R starts from 30+% and 90% in the statistical and standard cases respectively (in figure 1)

and from 50% and 70_{+}% (in figure 2) denotes the fact that there was, at least, one individual which did comply with the constraints in the first 5 generations.

In figure 3 we show the RE for SGA1P and STA1P in problem 7. Here ε_R for SGA1P starts at 100% meaning that no feasible individuals were present during the initial generations of the algorithm.

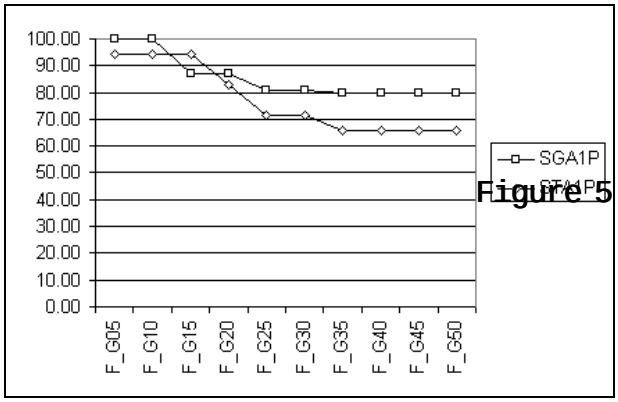

Figure 3. Non-elitist 1-point Crossover for Problem 7.

In figure 4 ε_R for SGAUN and STAUN for problem 7 is shown. In both figures 3 and 4 the statistical variations outperform their traditional counterparts.

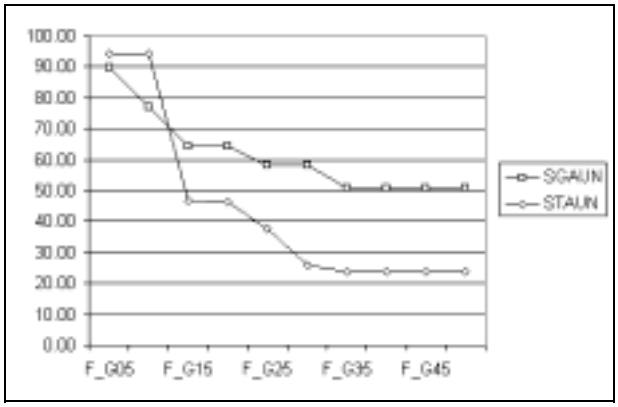

Figure 4. Non-elitist Uniform Crossover for Problem 7.

In order to have a more complete picture of the actual behavior of the 8 variations we averaged the results for all eight problems. In the next 4 figures (5, 6, 7 and 8) we show the averaged relative error grouped by variation: a) Non-elitist/1-point, b) Non-elitist/uniform, c) Elitist/1-point and d) Elitist/uniform.

With the exception of the elitist algorithms with 1-point crossover, all the statistical variations yielded better behaviors than their counterparts.

Finally, in figure 9 we show the average ε_R for all 8 variations we explored.

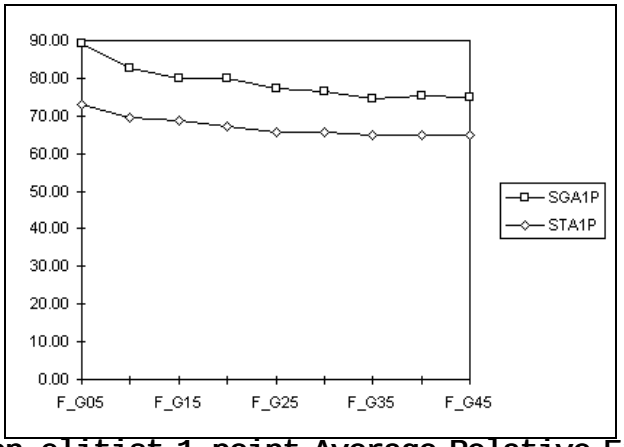

Figure 5. Non-elitist 1-point Average Relative Error.

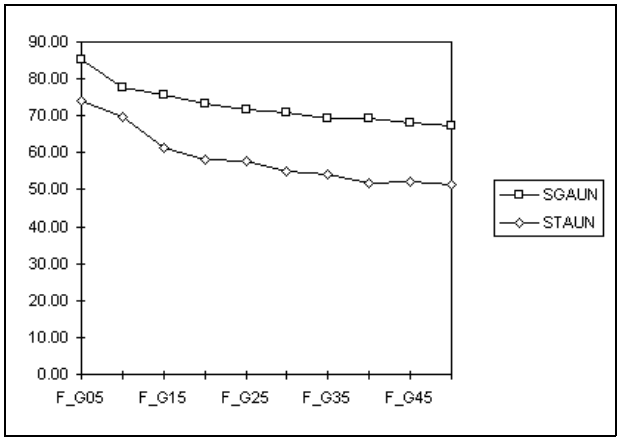

Figure 6. Non-elitist Uniform Relative Error.

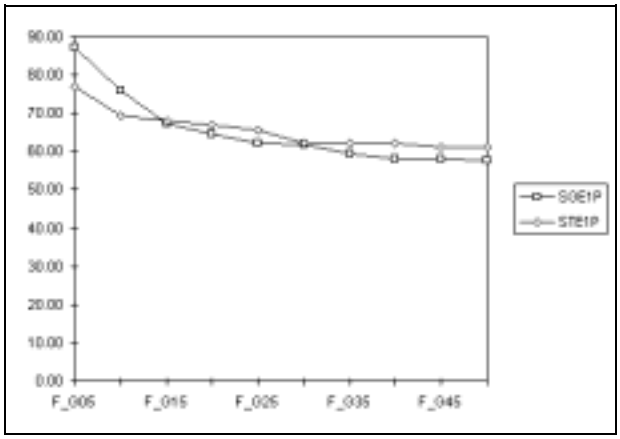

Figure 7. Elitist 1-point Relative Error.

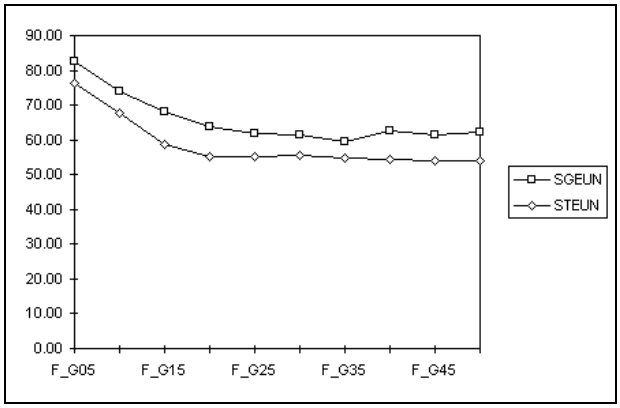

Figure 8. Elitist Uniform Average Error

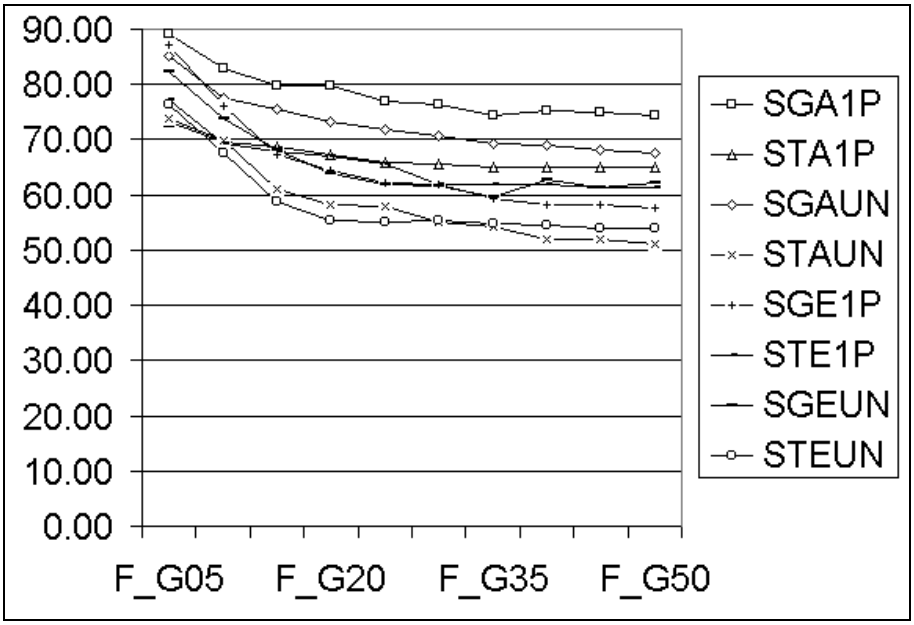

Figure 9. Average Error for All Problems.

The variations, ordered by their performance from worst to best were as follows: 1) SGA1P, 2) SGAUN, 3) STA1P, 4) SGEUN, 5) STE1P, 6) SGE1P, 7) STEUN, 8) STAUN.

5. Conclusions

As expected from previous theoretical results [8], uniform crossover is superior to 1-point crossover in general. Also expected [9] was the fact that, in all cases except for the best overall performer, elitist variations were superior to their non-elitist counterparts. Interestingly, however, although our initial intuition was that standard variations would have similar behavior to their statistical counterparts this turned out not to be so. Only in the case of elitist 1-point crossover was the statistical variation worse. However, even this may be attributed to random fluctuations and to the short number of generations.

Why did the statistical GAs turn out to be superior? From the qualitative analysis it would seem that the STA does not easily get trapped in undesired pdfs with small σ. Additionally one may speculate that the STA is less sensitive to misleading schemas which are known to be responsible of deceptive problems because of the probabilistic genome's "fuzzy" nature. A similar consideration may lead us to believe that the STA will also be impervious to spurious correlation, another well known problem [10] with traditional GAs.

Further research should be devoted to testing these issues plus the one related with the rate of convergence for larger *G*s. If the above considerations turn out to be confirmed then STAs have another advantage over traditional GAs: they are more efficient in terms of computational demands. This is obvious from the fact that actual selection and crossover are never performed. In the actual simulations STAs were at least twice as fast as their standard counterparts.

References

[1] Mitchell, M., and Forrest, S. "What makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation", Machine Learning, **13**:285-319, 1993.

[2] Goldberg, D.E., "Simple Genetic Algorithms and the Minimal Deceptive Problem". In L.D. Davis (Ed.), Genetic Algorithms and Simulated Annealing. Research Notes in Artificial Intelligence. Los Altos, CA; Morgan Kaufmann, 1987.

[3] Mitchell, M., Forrest, S., and Holland, J. "The Royal Road for genetic algorithms: Fitness Landscapes and GA Performance". In F.J. Varela and P. Bourgine, eds., *Toward a practice of autonomous Systems: Proceedings of the First European Conference on Artificial Life*, MIT Press, 1992.

[4] Enrique Díaz Infante, personal communication, March 11, 1999.

[5] Hamming, R., "Introduction to Coding and Information Theory", Prentice-Hall, 1980.

[6] Holland, J. "Adaptation in Natural and Artificial Systems", Ann Arbor, MI; University of Michigan Press, 1975.

[7] Kuri, A., "A Comprehensive Approach to Genetic Algorithms in Optimization and Learning", Ed. Politécnica, 1999.

[8] Spears, W., De Jong, K., "An Analysis of Multi-Point Crossover". In Rawlins, G., editor, *Foundations of Genetic Algorithms*, pp. 301-315, Morgan Kauffman, 1991.

[9] Rudolph, G., "Convergence Analysis of Canonical Genetic Algorithms", IEEE Transactions on Neural Networks, **5**(1):96-101, January, 1994.

[10] Schaffer, J., Eshelman J., and Offut, D., "Spurious Correlation and Premature Convergence in Genetic Algorithms". In Rawlins, G., editor, *Foundations of Genetic Algorithms*, pp. 102-112, Morgan Kauffman, 1991.