Published in Spector, L., E. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors. 2001 Proceedings the Genetic and Evolutionary Computation Conference, GECCO-2001, pp.584-591, San Francisco, USA, 7-11 July, CA: Morgan Kaufmann Publishers, 2001.

On Biologically Inspired Genetic Operators: Transformation in the Standard Genetic Algorithm

Anabela Simões[1,2]
[1]Instituto Superior de Engenharia de Coimbra
Quinta da Nora
3030 Coimbra - Portugal
abs@isec.pt

Ernesto Costa[2]
[2]Centro de Informática e Sistemas da Universidade de Coimbra, Polo II - Pinhal de Marrocos
3030 Coimbra - Portugal
ernesto@dei.uc.pt

Abstract

In this paper, we introduce a biologically inspired recombination operator that occurs in the colonies of bacteria. The mechanism is called transformation and is responsible for the genetic variation and consequently the advantageous characteristics that some bacteria possess. We present an implementation of the transformation mechanism in the standard GA (SGA) and we compare its performance solving two different classes of problems using either transformation or the traditional crossover operators. The results show that the GA using transformation is always superior to the SGA. The good results obtained by transformation seem to be related to the great degree of diversity that the mechanism introduces in population.

1 INTRODUCTION

For a population to survive changes in its environment it must have sufficient genetic variety to adapt to the new conditions: less genetically diverse populations may be at greater risk. Known as genetic diversity, this great variation within species is what allows populations to adapt to changes in climate and other local environmental conditions.

Genetic Algorithms (GAs) are inspired by genetics and natural selection: a population evolves through a number of generations, where the fittest individuals are more likely to be selected to reproduce in each generation. This process allows the evolution of the population to the best solution (Holland, 1992; Goldberg, 1989). The population's diversity is introduced by the application of two main genetic operators: mutation and crossover. These operators produce changes in the individuals, creating evolutionary advantages in some of them.

Nature maintains genetic diversity by several mechanisms besides crossover and mutation. Some of those mechanisms are: inversion, transduction, transformation, conjugation, transposition and translocation (Gould and Keeton, 1996).

Some researchers in the field of Evolutionary Computation (EC) highlighted the importance of studying different biologically inspired genetic operators. (Mitchell and Forrest, 1994) and (Banzhaf et al., 1998) stress that it would be important to analyze if some of the mechanisms of rearranging genetic material present in the biological systems, when implemented and used in the Evolutionary Algorithms (EA), improve their performance.

Several authors have already used some biologically inspired mechanisms besides crossover and mutation in EA. For instance, inversion (Holland, 1992), conjugation (Harvey, 1996), translocation (De Falco et al., 2000), transduction (Nawa et al., 1999) and transposition (Simões and Costa, 1999; 2001a) were already used as the main genetic operators in the EA. As far as we know none implementation of the transformation mechanism was tested in EA.

Bacteria sometimes take up and incorporate fragments of DNA from the environment. This is called transformation (Clark and Russell, 1997).

In this paper, we propose a computational implementation of the transformation mechanism and we study the GA performance solving two different problems. The empirical analysis will focus the application of the traditional crossover operators and transformation, for different population's size. The two classes of problems used to study the GA performance were function optimization (Rastrigin, Griewangk, Schwefel and Ackley test functions) and a combinatorial optimization problem (0/1 knapsack problem).

This paper is organized in the following manner. First, in section 2, we describe the biological functioning of the transformation mechanism and we introduce our computational implementation for the proposed recombination mechanism. Section 3, details the characteristics of the experimental environment, including the selected problems to test the GA performance and the GA parameters. In section 4, we make an exhaustive

comparison of the results obtained with the proposed recombination operator and with the standard crossover operators (1-point, 2-point and uniform crossover). Finally, we present the relevant conclusions of the work.

2 TRANSFORMATION

In our work, we will propose a modified GA with the introduction of a new biologically inspired operator, called transformation. Next sections will describe this mechanism.

2.1 BIOLOGICAL TRANSFORMATION

Some bacteria readily take up outside DNA. If they have this ability, they are said to be competent. Competent bacteria can absorb fragments of DNA proceeding from dead bacteria and present in their environment.

Usually, transformation consists in the transfer of small pieces of extra cellular DNA between organisms. These strains of DNA, or gene segments, are extracted from the environment and added to recipient cells (Russell, 1998).

After that, there are two possibilities, failure or success, known technically as restriction and recombination. Restriction is the destruction of the incoming foreign DNA, since those bacteria assume that foreign DNA is more likely to come from an enemy, such as a virus. In this case, transformation fails. Recombination is the physical incorporation of some of the incoming DNA into the bacterial chromosome. If this happens, genes from the assimilated segment replace some of the host cell's genetic information and bacteria are permanently transformed. Once integrated in the chromosome, the DNA segment is able to survive.

2.2 COMPUTATIONAL TRANSFORMATION

The DNA fragments to incorporate in the individuals of the population are generated at the beginning of the process. This DNA fragments consist in binary strings of different lengths and will form the gene segment pool.

We will use the transformation mechanism as the main genetic operator in the GA. Therefore, transformation is applied every generation instead of the standard crossover operator. First, we select the individuals to be transformed using the roulette-wheel selection method and these individuals are changed with a fixed probability. Part of the gene segment pool is changed every generation, using genetic information of the individuals of the population.

This modified GA will be referred as Transformation-based GA (TGA) and is described in Figure 1.

The main aspects to consider in the implementation of transformation are the origin of the gene segments that will transform each individual and how the process of transformation will occur. These aspects will be detailed in the next sections.

1. Generate Initial Population
   Generate Initial Gene Segment Pool
2. DO
   2.1. Evaluate Population
   2.2. Select Individuals
   2.3. Transform Individuals
   2.4. Replace Population with New Individuals
   2.5. Create New Gene Segment Pool

WHILE (NOT Stop_Condition)

Figure 1: The GA Modified with Transformation

2.2.1 The Basic Functioning of the Transformation Mechanism

The GA starts with an initial population of individuals and an initial pool of gene segments, both created at random. In each generation, we select individuals to be transformed and we modify them using the gene segments in the segment pool. After that, the segment pool is changed, using the old population to create part of the new segments with the remaining being created at random (see Figure 2).

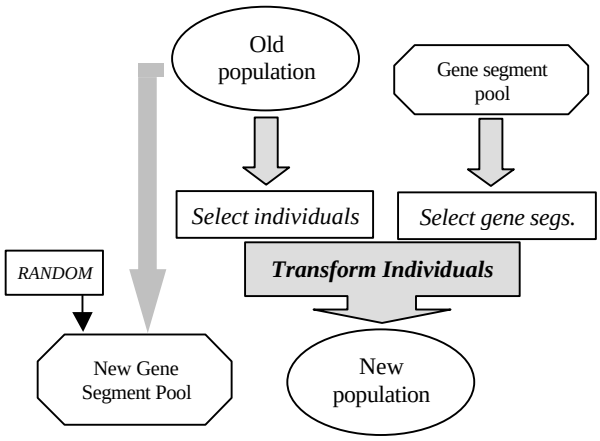

Figure 2: Computational Transformation

2.2.2 Origin of the Gene Segments

The segments that each individual will take up from the "surrounding environment" will proceed, mostly, from the individuals existing in the previous generation. In the used experimental setup, we changed the segment pool every generation. The modifications were made replacing 70% of the segments with new ones, created from the individuals of the old population. The remaining 30% were created at random. The size of the gene segments is also chosen in a random manner.

2.2.3 Transforming the Genetic Information of an Individual

After selecting individuals to a mating pool, we use the transformation mechanism to produce new individuals. In this case, there is no sexual reproduction among the individuals of the population. Each individual will generate a new one through the process of transformation. We can consider this process a form of asexual reproduction. Each individual will be transformed using a transformation probability.

The proposed mechanism can be described as follows: we select a segment from the segment pool and we randomly choose a point of transformation in the selected individual. The segment is incorporated in the genome of the individual, replacing the genes after the transformation point, previously selected. Obviously, the chromosome is seen as a circle. Proceeding this way the chromosome length is kept constant. This corresponds to the biological process where the gene segments, when integrated in the recipient's cell DNA, replace some genes in its chromosome. Figure 3 illustrates the process of transforming an individual.

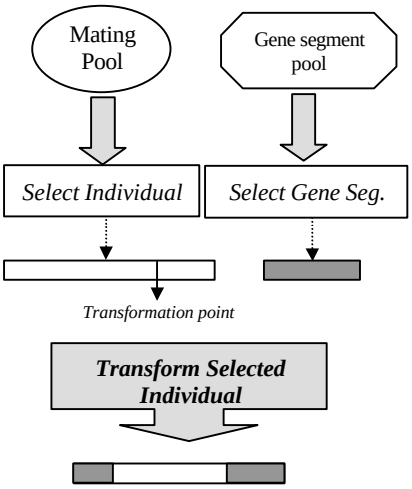

Figure 3: Transforming an Individual

3 EXPERIMENTAL SETTINGS

In order to investigate the performance of the TGA, we selected two different classes of problems: a combinatorial optimization problem (the 0/1 Knapsack problem (KP)) and the function optimization domain.

We selected these problems, since that, they are well known benchmarks to EA (Goldberg, 1989).

3.1 THE 0/1 KNAPSACK PROBLEM

The knapsack problem is a NP-complete problem, where we have to find the feasible combination of objects so that the total value of the objects put in the knapsack is maximized, subject to a capacity or weight constraint.

Formally, let C be the weight limitation (maximum permissible weight of the knapsack), let the integers 1,2,…,n denote n available types of objects, p_i and w_i, the value (or profit) and the weight of the ith object, respectively. A solution for the problem is represented by the binary vector **x** of length n. Each element of **x** can be zero or one: if x_i=1 then the item i was selected for the knapsack.

The knapsack problem can be expressed as

$$max \sum_{i=1}^{n} p_i.x_i \quad (1)$$

i.e., maximizing the profits, subject to the weight constraint

$$\sum_{i=1}^{n} x_i.w_i \le C, \quad (2)$$

where x_i is the selected object.

3.1.1 The Implemented Knapsack

We used several knapsack types (with 50, 100, 250 and 500 items). The evaluation of the solutions used a penalty function; the weights and profits vectors were created without any correlation and we used average capacity for the knapsack, as suggested in (Michalewicz, 1999).

The fitness f(x) for each binary string is determined as:

$$f(x) = \sum_{i=1}^{n} x_i.p_i - Pen(x) \quad (3)$$

with $Pen(x)$ the penalty function.

The penalty function is zero to all feasible solutions (those that don't exceed the knapsack capacity) and greater than zero otherwise. There are many possibilities for assigning the penalty value to the infeasible solutions. In our case, we considered a logarithmic penalty function defined by expression (4).

$$Pen(x) = log_2\left(1 + r.\left(\sum_{i=1}^{n} x_i.w_i - C\right)\right) \quad (4)$$

with $r = max_{i=1..n}\{p_i/w_i\}$

The generation of the vectors of profits (P[i]) and weights (W[i]) was made using the uncorrelated method, i.e.,

W[i]=(uniformly) random ([1..v])

P[i]=(uniformly) random ([1..v])

The value used for the parameter v was 10.

The capacity of the knapsack (average capacity) was calculated by:

$$C = 0.5\sum_{i=1}^{n} W[i] \quad (5)$$

3.2 TEST FUNCTIONS

We also evaluate the transformation mechanism by comparing its performance with the performance of the SGA (using three standard crossover operators) on several function optimization problems. To assess the quality of the algorithms we used the minimum function value found after a fixed number of function evaluations (50000, 100000 and 200000 in this case). The selected functions selected to analyze the GA performance were *Rastrigin*, *Schwefel*, *Griewangk*, and *Ackley* functions. All those functions are highly multimodal and have been used in other experimental comparisons of EA (Potter and De Jong, 1994; Gordon and Whitley, 1993).

The Rastrigin function is defined as:

$$f(x) = A*n + \sum_{i=1}^{n} x_i^{\,2} - A*\cos(2\Pi x_i) \qquad (6)$$

where n=20, A=10 and $-5.12 \le x_i \le 5.12$. The main characteristic of this function is the existence of many sub-optimal peaks whose values increase as the distance of the global optimum point increases

The Schwefel function is defined as:

$$f(x) = V*n + \sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right) \qquad (7)$$

where n=10, V=418.9829 and $-500 \le x_i \le 500$. The global minimum of the function is zero. The interesting aspect of this function is the existence of a second-best minimum far away from the global minimum, which can trap the optimization algorithms on a local optimum.

The Griewangk function is defined as:

$$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^{\,2}}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \qquad (8)$$

where n=10 and $-600 \le x_i \le 600$. This function has a product term, which introduces interdependency among the variables.

The Ackley function is defined as:

$$f(x) = 20 + e - 20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^{\,2}}\right) - \exp\left(\frac{1}{n}\cos(2\pi x_i)\right) \qquad (9)$$

where n=30 and $-30 \le x_i \le 30$. At a low dimension the landscape of this function is unimodal, however, the second exponential term covers the landscape with many small peaks and valleys.

3.3 THE PARAMETERS OF THE GENETIC ALGORITHM

The GA was first implemented with crossover (1-point. 2-point and uniform) and then with transformation. In the first problem, the 0/1 knapsack, we executed experiments to study the effect of the population size in the GA efficiency. Therefore, the population size varied between 20, 50, 100 and 200 individuals. In this problem, the GA evolved through 1000 generations.

For the function optimization domain, we fixed the maximum number of function evaluations equal to 200000.

In both classes of problems, we used binary representation to encode the problem, the roulette wheel selection and an elite size of two individuals. The mutation and crossover/transformation rate were 0.1% and 70%, respectively. The results reported in the next sections are the average computed over twenty-five runs.

3.4 EVALUATION MEASURE

We used the De Jong's off-line measure to compare GA efficiency when applied crossover or transformation (De Jong 1975). This measure is defined by:

$$X^*_e(g) = \frac{1}{T} * \sum_{t=1}^{T} f_e^{\,*}(t) \qquad (10)$$

where $f_e^{\,*}$ = *best* $\{f_e(1), f_e(2), ..., f_e(n)\}$ and **T** is the number of runs. This means that off-line measure is the average of the best individuals in each generation. Due to the 25 trials, the average of the 25 runs was evaluated.

4 EXPERIMENTAL RESULTS

Next sections show the averaged results obtained in the knapsack problem and in the selected test functions.

4.1 RESULTS OBTAINED IN THE KNAPSACK PROBLEM

The proposed mechanism allowed the GA to achieve better solutions than the SGA using one-point, two-point or uniform crossover. This observation can be generalized to all the tested instances of the KP, i.e., with 50, 100, 250 and 500 items. Table 1 summarizes all the results for the 0/1 KP using the SGA and the TGA with different population's sizes. The best solutions found for n=50, 100, 250 and 500 are marked in bold.

As we can see, the population size is an important parameter when using crossover. In fact, increasing the population size from 50 to 100 or 200 individuals, crossover's performance shows some improvements. Using transformation with smaller populations, the GA obtained better results than the SGA with larger populations. As we can see in the table, with only 20 individuals in the population the TGA achieves solutions superior to the ones achieved by the SGA with 200 individuals.

Table 1: Summary of the Obtained Results using the SGA and the TGA with Different Population's Size

		Genetic Operator												
		One-point Crossover			*Two-point Crossover*			*Uniform Crossover*			*Transformation*			
	P. Size®	*50*	*100*	*200*	*50*	*100*	*200*	*50*	*100*	*200*	*20*	*50*	*100*	*200*
Nº of Items	50	227,77	236,48	244,17	245,82	253,91	263,10	265,00	271,58	273,56	278,63	284,55	287,51	**300,76**
	100	358.71	454.66	466.24	439.65	491.63	514.46	490.97	511.70	520.96	528.21	551.28	576.93	**590.40**
	250	950.08	1074.39	1089.54	923.92	1120.51	1036.18	1037.94	1211.10	1173.67	1330.68	1361.94	1375.79	**1410.28**
	500	1734.66	1985.96	1959.66	1845.88	1972.49	1996.18	2001.11	2303.35	2183.51	2548.29	2630.82	2633.53	**2656.17**

In order to understand these results, it is important to see how the GA evolved through the 1000 generations. In Figure 4, we show a representative example for the KP with 100 items. The figure compares the GA performance using uniform crossover with 200 individuals and transformation with a population of 50 binary strings.

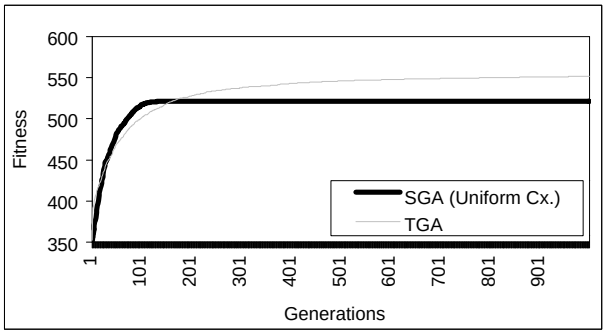

Figure 4: Comparing the SGA (200 inds.) and the TGA (50 inds.) Performances

As Figure 4 shows, uniform crossover only allow the SGA to improve in the first generations and after that the evolution stops. The TGA evolved during a long period, and was able to reach better results than crossover, even with a smaller population.

To analyze the influence of the population size in the GA's performance when using transformation we show, in Figure 5, the results obtained for the KP with 100 items. To the other instances, the results are quite similar. We can see that when using larger populations the maximum result obtained is superior.

Comparing the execution times spent by the four genetic operators solving the KP, we can see that transformation is the mechanism that consumes more time. Nevertheless, the differences are relatively small compared with the crossover operators. The time spent by the TGA is approximately 7% superior to the time spent by the operator that obtains the worst results (one-point crossover) and 3% superior to the best crossover operator (uniform crossover).

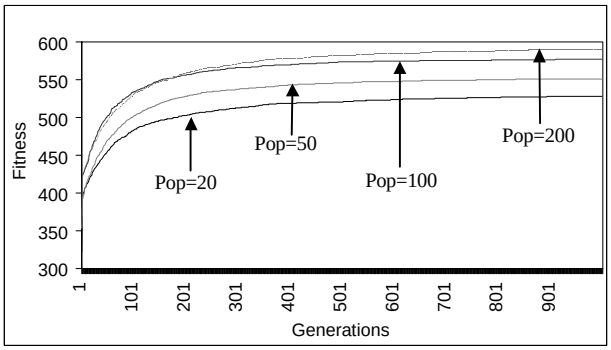

Figure 5: The TGA's Performance using Different Population's Size

Table 2 reports the results (in seconds) obtained running 25 trials of the SGA and the TGA with a population of 200 individuals, in a Pentium II with a 300 MHz processor.

Table 2: Time Spent to Solve the 0/1 Knapsack Problem

Nº items	**Cx1**	**Cx2**	**CxU**	**TGA**
50	2726	3276	3302	3702
100	6642	6757	6807	7095
250	15736	15842	16005	16656
500	30870	31761	32356	33382

Table 3: Function Optimization: Summary of the Results (minimization)

		Genetic Operator											
		One-point Crossover			*Two-point Crossover*			*Uniform Crossover*			*Transformation*		
	Nº evals	**50000**	**100000**	**200000**	**50000**	**100000**	**200000**	**50000**	**100000**	**200000**	**50000**	**100000**	**200000**
Function	Rastrigin	88.170	88.170	88.170	67.639	67.639	67.639	63.739	63.739	63.739	73.272	52.518	**36.682**
	Griewangk	0.323	0.323	0.323	0.259	0.259	0.259	0.244	0.244	0.244	0.074	0.026	**0.010**
	Schwefel	665.406	665.406	665.406	557.770	557.770	557.770	456.273	456.273	456.273	220.878	62.404	**8.695**
	Ackley	16.248	16.248	16.248	15.102	15.102	15.102	14.181	14.181	14.181	11.617	8.645	**5.941**

4.2 RESULTS OBTAINED IN THE FUNCTION OPTIMIZATION DOMAIN

The TGA obtained, in the entire set of test functions, the best solutions after 200000 function evaluations. Table 3 reports the achieved results. The results presented are those obtained after 50000, 100000 and 200000 function evaluations using the SGA and the TGA. The best solutions are marked in bold.

In this case, the GA using the transformation mechanism evolves very slowly to the achieved result. On the other hand, the SGA converges very rapidly to the obtained value, but is unable to continue evolving. Besides, just like in the KP, in the function optimization domain, TGA obtained better results than SGA with fewer number of function evaluations.

The graphical representation shown in Figure 6 illustrates the SGA and TGA performances minimizing the Ackley function, but we observed a similar behavior in all the test functions. Once the population converges to a certain value, SGA is incapable of continue exploring other zones of the search space. The TGA evolves slower, but can continue improving during the 2000 generations.

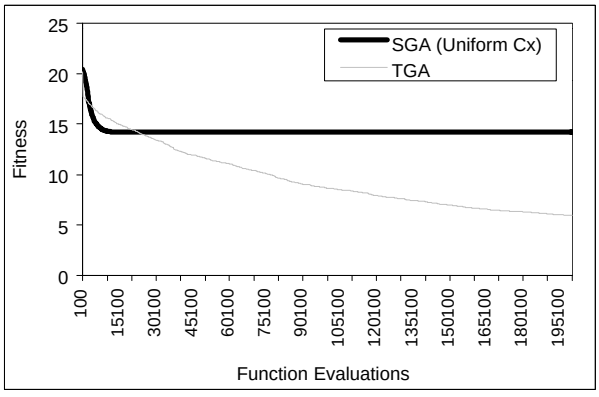

Figure 6: SGA and TGA evolution in 200000 function evaluations

Once again, these results appear to be a consequence of the loss of diversity in the population when using the crossover operators. TGA evolves during the entire simulation because the genetic variation of the individuals is kept in high levels. In the next section, we will focus the population's diversity measured in both problem domains.

Concerning the computational times, once again, TGA was the slower algorithm, but the differences to the times used by the crossover operators are quite small. TGA was approximately 7% slower than one-point crossover (the operator which obtained the worst results) and 4% slower than uniform crossover (which obtained the best performance among the crossover operators). Table 4 shows the times (in seconds) spent in the execution of the 25 trials for the minimization of the test functions.

Table 4: Time Spent to Minimize the Test Functions

Function	***Cx1***	***Cx2***	***CxU***	***TGA***
Rastrigin	7039	7059	7298	7698
Griewangk	5338	5360	5478	5686
Schwefel	4683	4722	4832	4989
Ackley	11320	11588	11667	12152

4.3 POPULATION'S DIVERSITY

The main reason for the good results obtained by the TGA seems to be the great diversity that the proposed mechanism introduces in the population. This can be the explanation for the fact of the TGA with 20 individuals outperforms the SGA with 200. To compare the diversity in the population we used a standard measure, which is the sum of the Hamming distances between all possible pairs in the population. This measure, when normalized, is defined as:

$$Div(Pop) = \frac{1}{LP(P-1)} \sum_{i=1}^{P} \sum_{j=1}^{P} HD(p_i, p_j) \qquad (11)$$

where L is the chromosome length, P is the population size; p_i is the i^{th} individual in the population and HD is the Hamming distance function.

Figure 7 shows the variation of the population's diversity for the KP. The results were obtained by the GA solving the KP problem with 100 items and compare the diversity

maintained by uniform crossover and transformation. To the other instances of the KP, the results were very similar.

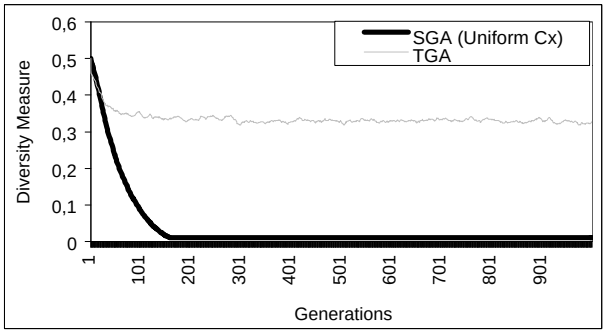

Figure 7: Population's Diversity in the KP

As we can see, the diversity of the population is higher when using transformation, indicating that the individuals are covering more areas of the search space. When applying uniform crossover, the population's diversity decreases to values near to zero avoiding the GA to continue evolving. In the Figure 4 we observed that the SGA stops evolving about generation 130. As Figure 7 indicates, the diversity of the population achieves the lower levels about generation 130.

In the domain of function optimization, the results were very similar. Figure 8 shows the diversity measure in the minimization of the Ackley function. Once again, there is a correspondence between the point where the diversity reaches low values and the point where the SGA stops evolving (20000 function evaluations in Figure 6).

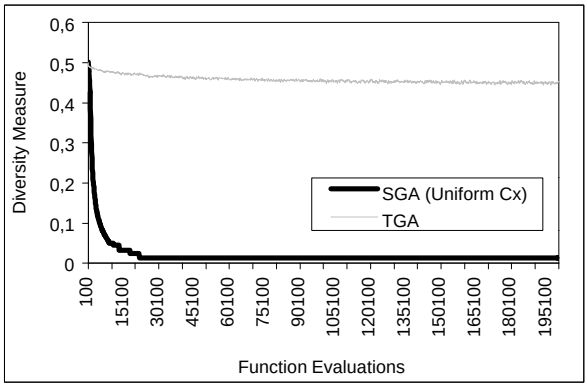

Figure 8: Population's Diversity minimizing Ackley Function

5 CONCLUSIONS

In this paper, we introduced a new genetic operator inspired in bacterial genetics, called transformation. We used this operator as an alternative to crossover and we studied the GA performance solving two different classes of problems. The results showed that the transformation mechanism is clearly superior to the SGA. Besides, with few individuals in population (or fewer function evaluations) transformation can achieve better solutions than crossover with larger populations.

Observing the population's diversity, we can see that transformation preserves a high degree of genetic variation among the individuals of the population.

We are currently using this genetic operator in a classical dynamic optimization problem and the preliminary results show that the TGA is able to adapt to the new solution when a change occurs (Simões and Costa, 2001b).

In order to enhance the GA performance when using this mechanism we are also implementing some modifications concerning some issues, namely, the assessment of the best transformation rate, the influence of the gene segment length and the generation of the gene segment pool.

Acknowledgements

This paper was partially supported by the Portuguese Ministry of Science and Technology under the program POSI.

References

W. Banzhaf, P. Nordin, R. E. Keller and F. D. Francone (1998). *Genetic Programming - An Introduction - On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann

D. Clark and L. Russell (1997). *Molecular Biology Made Simple and Fun.* Cache River Press.

I. De Falco, A. Iazzetta, E. Tarantino and A. Della Cioppa (2000). *On Biologically Inspired Mutations: The Translocation*. In Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference (GECCO'00), 70-77, Las Vegas, USA, 8-12 July 2000.

K. A. De Jong (1975). *Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. Dissertation, Department of Computer and Communication Science, University of Michigan.

D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc.

V. S. Gordon and D. Whitley (1993). *Serial and Parallel Genetic Algorithms as Function Optimizers*. In S. Forrest (ed.), Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA5), 177-183, Morgan Kaufmann.

J. L. Gould and W. T. Keeton (1996). *Biological Science.* W. W. Norton & Company.

I. Harvey (1996). The Microbial Genetic Algorithm. Submitted to *Evolutionary Computation.* MIT Press.

J. H. Holland (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* 1st MIT Press edition, MIT Press.

Z. Michalewicz (1999). *Genetic Algorithms + Data Structures = Evolution Programs.* 3rd Edition Springer-Verlag.

M. Mitchell and S. Forrest (1994). Genetic Algorithms and Artificial Life. *Artificial Life* **1**(3):267-289.

N. Nawa, T. Furuhashi, T. Hashiyama and Y. Uchikawa (1999). *A Study of the Discovery of Relevant Fuzzy Rules Using Pseudo-Bacterial Genetic Algorithm.* IEEE Transactions on Industrial Electronics.

M. A. Potter and K. De Jong (1994). *A Cooperative Coevolutionary Approach to Function Optimization*. In the Proceedings of the Third Parallel Problem Solving from Nature (PPSN3), Jerusalem, Israel, 249-257, Springer-Verlag.

P. J. Russell (1998). *Genetics*. 5th edition, Addison-Wesley.

A. Simões and E. Costa (1999). *Transposition versus Crossover: An Empirical Study*. Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E. (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99), 612-619, Orlando, Florida USA, CA: Morgan Kaufmann.

A. Simões and E. Costa (2001a). *An Evolutionary Approach to the Zero/One Knapsack Problem: Testing Ideas from Biology.* In the Proceedings of the Fifth International Conference on Neural Networks and Genetic Algorithms (ICANNGA' 2001), 22-25 April, Prague, Czech Republic, Springer-Verlag.

A. Simões and E. Costa (2001b). *Using Biological Inspiration to Deal with Dynamic Environments*. Submitted to the 7th International Conference on Soft Computing (MENDEL'2001).