Self Adaptation of Operator Rates in Evolutionary Algorithms

Jonatan Gomez

Universidad Nacional de Colombia and The University of Memphis
jgomezpe@unal.edu.co, jgomez@memphis.edu

Abstract. This work introduces a new evolutionary algorithm that adapts the operator probabilities (rates) while evolves the solution of the problem. Each individual encodes its genetic rates. In every generation, each individual is modified by only one operator that is selected according to the encoded rates. Such rates are updated according to the performance achieved by the offspring (compared to its parent) and a random learning rate. The proposed approach is augmented with a simple transposition operator and tested on a number of benchmark functions.

1 Introduction

Evolutionary algorithms (**EA**) are optimization techniques based on the principles of natural evolution [1]. Although EAs have been used successfully in solving optimization problems, the performance of this technique (measure in time consumed and solution quality) depends on the selection of the EA parameters. Moreover, the process of setting such parameters is considered a time-consuming task [2]. Several research works have tried to deal with this problem [3]. Some approaches tried to determine the appropriated parameter values by experimenting over a set of well-defined functions [4,5], or by theoretical analysis [6,7,8]. Another set of approaches, called Parameter Adaptation (**PA**), tried to eliminate the parameter setting process by adapting parameters through the algorithm's execution [9,10,11,8,12]. PA techniques can be roughly divided into centralized control techniques (central learning rule), decentralized control techniques, and hybrid control techniques.

In the centralized learning rule approach, genetic operator rates (such as mutation rate, crossover rate, etc) are adapted according to a global learning rule that takes into account the operator productivities through generations (iterations) [3]. Generally, only one operator is applied per generation, and it is selected based on its productivity. The productivity of an operator is measured in terms of good individuals produced by the operator. A good individual is one that improves the fitness measure of the current population. If an operator generates a higher number of good individuals than other operators then its probability is rewarded. Two well known centralized learning rule mechanism are the adaptive mechanism of Davis [9], and Julstrom [10]. In Davis's approach, the operators and parents are stored for each offspring. If an offspring is better than the current best individual in the population, the individual and the genetic operator

K. Deb et al. (Eds.): GECCO 2004, LNCS 3102, pp. 1162–1173, 2004.

used for generating it are rewarded. Some portion of the reward is given recursively to the parents, grand parents, and so on. After certain number of iterations the operator probabilities are updated according to the reward reached. In the approach of Julstrom, a similar mechanism to Davis is developed, but only information about recently generated chromosomes is maintained for each individual. Although applied with relative success, a centralized technique has two main disadvantages: First, it requires extra memory for storing information on the effect of each genetic operator applied to an individual, parents, grandparents, etc. The amount of memory required grows exponentially on the number of generations used. For example, if the operator productivity is defined using the latest n generations and a single byte is used per productivity information, the extra memory required will be approximated 2^n bytes per individual. Second, a centralized technique requires an algorithm that uses such information for calculating the operator productivity in a global sense; it cannot be defined at the individual level, only at the population level. Therefore, the time complexity of the operator productivity grows linearly on the number of generations used and is increased by the size of the population.

In decentralized control strategies, genetic operator rates are encoded in the individual and are subject to the evolutionary process [3,13]. Accordingly, genetic operator rates can be encoded as an array of real values in the semi-open interval [0.0,1.0), with the constraint that the sum of these values must be equal to one [11]. Since the operator rates are encoded as real numbers, special genetic operators, meta-operators, are applied to adapt or evolve them. Although decentralized approaches have been applied to many problems with relative success, these approaches have two main disadvantages: First, a set of meta genetic operators used for evolving the probabilities must be defined. It is not easy to determine which operators to use and how these operators will affect the evolution of the solution. Second, rates for such meta operators have to be given. Although these techniques are not very sensitive to the setting of the meta operator rates, not every set of values works well.

In this paper, a "hybrid" technique for parameter adaptation is proposed. Specifically, each individual encodes its own operator rates and uses a randomized version of a learning rule mechanism for updating them. Such a randomized learning rule mechanism is defined locally (per individual) and uses the productivity of the genetic operator applied and a "random" generated learning rate. If a non-unary operator is selected, the additional parents are chosen using a selection strategy. The operator rates are adapted according to the performance achieved by the offspring compared to its parent, and the random learning rate generated. Although the population size is an EA parameter that can be adapted, and each genetic operator has its own parameters that can be adapted, this work is only devoted to the adaptation of genetic operator rates.

This paper is divided in five sections. Section 2 presents the proposed hybrid adaptive evolutionary algorithm. Section 3 reports some experimental results on binary encoding functions. Section 4 reports some results on real encoding optimization problems. Section 5, draws some conclusions.

Algorithm 1 Hybrid Adaptive Evolutionary Algorithm (HAEA)

```
HaEA( λ, terminationCondition )
1.  t_0 = 0
2.  P_0 = initPopulation( λ ) ,
3.  while( terminationCondition( t, P_t ) is false ) do
4.    P_{t+1} ={}
5.    for each ind ∈ P_t do
6.      rates = extract_rates( ind )
7.      δ = random(0,1)      // learning rate
8.      oper = Op_Select( operators, rates )
9.      parents = ParentSelection(P_t, ind )
10.     offspring = apply( oper, parents )
11.     child = Best( offspring, ind )
12.       if( fitness( child ) ¿ fitness( ind ) ) then
13.         rates[oper] = (1.0 + δ)*rates[oper]   //reward
14.     else
15.       rates[oper] = (1.0 - δ)*rates[oper]   //punish
16.     normalize_rates( rates )
17.     set_rates( child, rates )
18.     P_{t+1} = P_{t+1}∪ {child}
19.   t = t + 1
```

2 Hybrid Adaptive Control

Algorithm 1 presents the proposed Hybrid Adaptive Evolutionary Algorithm (**HaEa**). This algorithm is a mixture of ideas borrowed from Evolutionary Strategies (**ES**), decentralized control adaptation, and central control adaptation.

2.1 Selection Mechanism

In HaEa, each individual is "independently" evolved from the other individuals of the population, as in evolutionary strategies [14]. In each generation, every individual selects only one operator from the set of possible operators (line 8). Such operator is selected according to the operator rates encoded into the individual. When a non-unary operator is applied, additional parents (the individual being evolved is considered a parent) are chosen according to any selection strategy, see line 9. As can be noticed, HaEa does not generate a parent population from which the next generation is totally produced. Among the offspring produced by the genetic operator, only one individual is chosen as child (line 11), and will take the place of its parent in the next population (line 17). In order to be able to preserve good individuals through evolution, HaEa compares the parent individual against the offspring generated by the operator. The Best selection mechanism will determine the individual (parent or offspring) that has the highest fitness (line 11). Therefore, an individual is preserved through evolution if it is better than all the possible individuals generated by applying the genetic operator.

2.2 Encoding of Genetic Operator Rates

The genetic operator rates are encoded into the individual in the same way as decentralized control adaptation techniques, see figure 1. These probabilities are initialized (into the *initPopulation* method) with values following a uniform distribution $U[0, 1]$. We used a roullete selection scheme for selecting the operator to be applied (line 8). To do this, we normalize the operator rates in such a way that their summation is equal to one (line 16).

SOLUTION	**OPER**$_1$	**...**	**OPER**$_n$
100101011..01	0.3	...	0.1

Fig. 1. Encoding of the operator probabilities in the chromosome

2.3 Adapting the Probabilities

The performance of the child is compared against its parent performance in order to determine the productivity of the operator (lines 12-15). The operator is rewarded if the child is better than the parent and punished if it is worst. The magnitude of reward/punishment is defined by a learning rate that is randomly generated (line 7). Finally, operator rates are recomputed, normalized, and assigned to the individual that will be copied to the next population (lines 16-17). We opted by generating the learning rate in a random fashion instead of setting it to a specific value for two main reasons. First, there is not a clear indication of the correct value that should be given for the learning rate; it can depend on the problem being solved. Second, several experiments encoding the learning rate into the chromosome showed that the behavior of the learning rate can be simulated with a random variable with uniform distribution.

2.4 Properties

Contrary to other adaptation techniques, HAEA does not try to determine and maintain an optimal rate for each operator. Instead, HAEA tries to determine the appropiate operator rate at each instance in time according to the concrete conditions of the individuals. If the optimal solution is reached by an individual in some generation, then the rates of the individual will converge to the same value in subsequent generations. This is true because no operator is able to improve the optimal solution. HAEA uses the same amount of extra information as a decentralized adaptive control; HAEA requires a matrix of $n * M$ doubles, where n is the number of different genetic operators and M is the population size. Thus, the space complexity of HAEA is linear with respect to the number of operators (the population size is considered a constant). Also, the time expended in calculating and normalizing the operator rates is linear with respect to the number of operators $n * M$ (lines 8 and 12-16). HAEA does not require special

operators or additional parameter settings. Well known genetic operators can be used without modifications. Different schemes can be used in encoding the solution: binary, real, trees, programs, etc. The average fitness of the population grows monotonically each generation since an individual is replaced by other with equal or higher fitness.

3 Experiments Using Binary Encoding

We conducted experiments using binary encoding in order to determine the applicability of the proposed approach. In the binary encoding, the solution part of the problem is encoded as a string of binary symbols [1].

3.1 Test Functions

We used four well known binary functions: The 100 bits MaxOnes function where the fitness of an individual is defined as the number of its bits that are set to 1, the ten deceptive order-3 and ten bounded deceptive order-4 functions were developed by Golberg in 1989 [15] and the $(8 * 8)$ royal road function developed by Forrest and Mitchell [16].

3.2 Experimental Settings

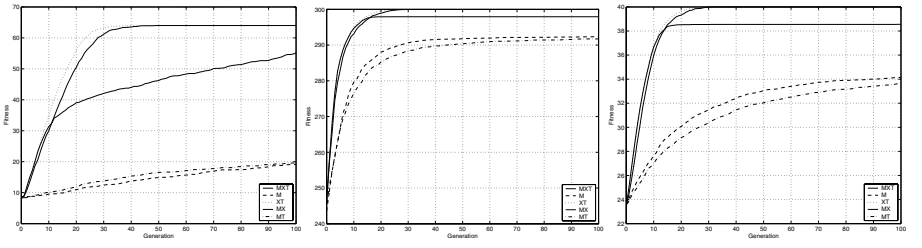
For each function, a population size of 100 was used and the HAEA algorithm was executed a maximum of 10000 fitness function evaluations (100 generations). A tournament of size 4 was applied to determine the additional parent of crossover. The reported results are the average over 100 different runs. We compared the performance reached by HAEA using different combinations of three well known genetic operators: single point mutation, single point crossover, and a simple transposition. In the single bit mutation, one bit of the solution is randomly selected (with uniform distribution) and flipped. This operator always modifies the genome by changing only one single bit. In single point crossover, a cutting point in the solution is randomly selected. Parents are divided in two parts (left and right) using such cutting point. The left part of one parent is combined with the right part of the other. In the simple transposition operator, two points in the solution are randomly selected and the genes between such points are transposed [17]. Five different combinations of genetic operators were tested: Only mutation (M); mutation and crossover (MX); mutation and transposition (MT); crossover and transposition (XT); and mutation, crossover and transposition (MXT).

3.3 Results

Table 1 shows the performance reached by HaEa on the binary functions. A value $a \pm b\,[c]$ indicates that a maximum average performance of a, with standard deviation of b, was reached by the HaEa using c fitness evaluations in all the runs.

Table 1. Performance reached by HAEA on binary functions using different operators.

	MaxOnes	Royal Road	Deceptive-3	Deceptive-4
M	90.00±1.32 [10000]	19.12±4.37 [10000]	292.34±1.44 [9900]	34.14±0.81 [9900]
MX	100.00±0.00 [5000]	55.12±7.83 [10000]	297.92±1.98 [2400]	38.56±1.04 [9900]
MT	84.56±1.69 [10000]	19.60±4.44 [10000]	291.72±1.44 [9600]	33.64±0.79 [10000]
XT	100.00±0.00 [3800]	64.00±0.00 [4000]	300.00±0.00 [2600]	40.00±0.00 [3100]
MXT	100.00±0.00 [3900]	64.00±0.00 [4900]	300.00±0.00 [3000]	40.00±0.00 [3100]



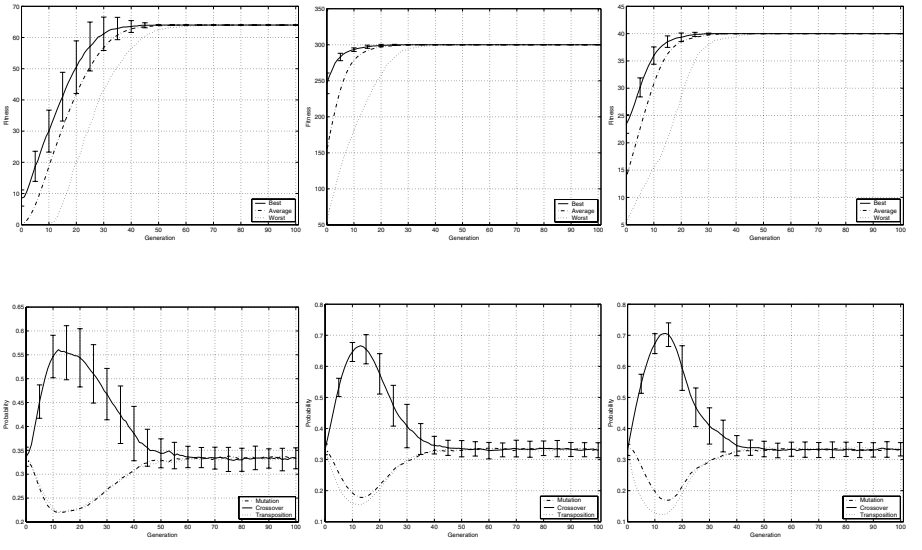
(a) (b) (c)

Fig. 2. Performance evolution using HAEA on binary functions. (a) Royal Road, (b) Deceptive-3 and (c) Deceptive-4.

Genetic Operators Analysis. Figure 2 shows the evolution of the fitness value for each variation of HAEA tested.

As expected, the performance of HAEA is highly affected by the set of operators used. While the performance reached by HAEA is low when using only mutation or mutation and transposition (M and MT), the performance is high and the optimal solution is found when all of the operators are used (MXT). These results indicate that HAEA is able to recognize the usefulness of each genetic operator. When crossover and transposition were used (MXT and XT), HAEA was able to find the optimal solution of each function. A possible explanation of this behavior is that transposition exploits the repetitive and symmetric structure definition of these test functions - any of these functions can be seen as the concatenation of small symmetric binary functions. Transposition can modify two of these small functions, the outer limits in the string being transposed, while maintaining other small functions, the inner part being transposed. It is not surprising that performance and convergence rate reached by HAEA when using crossover are higher than without using crossover. When crossover is not included (M and MT), HAEA is not able to find the optimal solution of any of the four functions. These results indicate that HAEA identifies the main role of crossover in the optimization process and allows it to exploit "good" regions of the search space. Figure 3 presents the HAEA(MXT) evolution of both solution and genetic operator rates for the tested functions. As shown,

the crossover rate increases rapidly in early stages of the evolutionary process. This means that crossover produced better individuals in early stages than other operators. When, crossover was not enough for locating the optimal solution, its rate decreases allowing HAEA to try mutation and/or transposition. Notice that HAEA is not trying to determine and maintain an optimal probability for each genetic operator chromosome as other adaptation techniques. HAEA has temporal learning of operator rates. This adaptive property allows HAEA to have a higher chance to locate an optimal solution than another evolutionary algorithm. When the optimal solution was reached for all individuals in the population, the genetic operators probabilities converge to the same value. This behavior is expected since no genetic operator can produce a fitter individual than the optimal solution.



(a) (b) (c)

Fig. 3. Performance (first row) and rates evolution (second row) using HAEA(MXT) on binary functions. (a) Royal Road, (b) Deceptive-3 and (c) Deceptive-4.

Comparing HAEA against other Evolutionary Algorithms and Previous Reported Results. Two standard GA were implemented in order to compare the performance of the proposed approach (HAEA): a Generational Genetic Algorithm (GGA) and a Steady State Genetic Algorithm (SSGA). The GGA used a tournament selection of size 4 for selecting each individual of the parent population, a standard bit mutation and single point crossover as gene-

tic operators. The mutation rate of each bit in the chromosome was fixed to $\frac{1}{l}$ where l is the length of the chromosome while the crossover rate was set to 0.7. These parameters have been considered good for solving binary encoding problems [14,2,11,8]. For the SSGA we used a tournament selection of size 4 for selecting the parent in the crossover operator, standard bit mutation and single point crossover as genetic operators, and *kill-worst* replacement policy. The crossover rate was fixed to 1.0 while the mutation rate of each bit was fixed to $\frac{1}{l}$ where l is the length of the chromosome. Table 2 summarizes the average performance reached by HAEA(MX), HAEA(MXT), our implementation of the GGA and SSGA, and compares them against the performance of some evolutionary algorithms reported in the literature.

Table 2. Comparison of HAEA performance on binary functions.

	MaxOnes	**Royal Road**	**Deceptive-3**	**Deceptive-4**
HAEA(MXT)	100.00±0.0[3900]	64.00±0.00[4900]	300.00±0.00[4800]	40.00±0.00[5000]
HAEA(MX)	100.00±0.0[5000]	55.12±7.83[10000]	297.92±1.98[2400]	38.56±1.04[9900]
GGA	100.00±0.0[4800]	49.52±9.24[10000]	293.52±3.01[10000]	37.08±1.29[2900]
SSGA	100.00±0.0[2800]	48.24±9.01[10000]	288.56±3.08[2000]	35.00±1.38[1100]
T-GGA [11]	99.96±0.20[7714]	35.52±6.02[7804]	289.68±2.41[5960]	-
T-SSGA[11]	100.00±0.0[2172]	40.64±7.65[3786]	289.12±3.08[3506]	-
T-D-GGA[11]	99.52±0.64[8438]	31.36±6.16[6086]	289.12±2.83[5306]	-
T-D-SSGA[11]	100.00±0.0[2791]	29.76±8.32[2428]	289.32±2.61[2555]	-
GA [8]	100.00±0.0[2500]	-	-	14.00 [10000]
PL-GA [8]	100.00±0.0[7400]	-	-	28.00 [10000]

Results in rows 5-8 are reported by Tuson and Ross in [11][1]. Results in rows 9 and 10 are reported by Lobo in [8][2]. HAEA(MXT) and HAEA(MX) outperformed other EA approaches in the three hard binary functions: Royal Road, Deceptive-3 and Deceptive-4[3]. In the simple MaxOnes problem, HAEA found the global optimal solution, but it took around twice the number of evaluations required by SSGA based approaches and the regular GA used by Lobo.

[1] These results are the average over 50 different runs. Row 5 presents the results obtained by Tuson and Ross using a Generational Genetic Algorithm (GGA) with fixed operator probabilities, row 6 using a Steady State Genetic Algorithm (SSGA), row 7 a GGA using distributed control parameter adaptation and row 8 a SSGA using distributed control. The population size used by Tuson and Ross was fixed to 100 individuals and the maximum number of iterations was set to 10000.

[2] These results are the average over 20 different runs. Row 9 presents the results obtained by Lobo using a regular GA with optimal parameters setting, population size of 100 individuals for the MaxOnes problem and 50 individuals for the deceptive-4 problem, , while row 10 presents the results using the Parameter-less GA proposed by Lobo. Results for the deceptive 4 function were approximated from [8], figure 4.4.

[3] The Parameter-less GA was able to find the solution after 4e+6 evaluations by increasing the population size

4 Experiments Using Real Encoding

We conducted experiments using real encoding in order to determine the applicability of the proposed approach. In the real encoding, the solution part of the problem is encoded as a vector of real values [14].

4.1 Experimental Settings

We used the real functions shown in table 3 as our testbed. In the last three functions, we fixed the dimension of the problem to $n = 10$. HAEA was executed a maximum of 20000 fitness evaluations (2000 iterations) with a population size of 100 individuals. The reported results are the average over 100 different runs. We implemented three different genetic operators: Gaussian mutation, Uniform mutation, Single real point crossover. The Gaussian mutation operator adds to one component of the individual a random number Δ that follows a Gaussian distribution $G(0, \sigma)$. We used $\sigma = \frac{max-min}{100}$, where max and min are the maximum and minumum values that can take each component. The uniform mutation operator replaces one of the component values of the individual with a random number following a uniform distribution $U(min, max)$. The single point real crossover generates two individuals by randomly selecting one component k and exchanging components $k, k+1, .., n$ between the parent. Four different combinations of operators were used by HAEA: Gaussian and uniform mutations (GU); Crossover and Gaussian mutation (XG); Crossover and uniform mutation (XM); and Crossover, uniform and Gaussian mutations (XUG).

Table 3. Real functions tested

Name	Function	Feasible region
Rosenbrock	$f(x) = 100 * \left(x_1^2 - x_2\right)^2 + (1 - x_1)^2$	$-2.048 \leq x_i \leq 2.048$
Schwefel	$f(x) = 418.9829 * n + \sum_{i=1}^{n} \left[-x_i * sin\left(\sqrt{\lvert x_i \rvert}\right)\right]$	$-512 \leq x_i \leq 512$
Rastrigin	$f(x) = n * A + \sum_{i=1}^{n} \left[x_i^2 - A * cos\left(2\pi x_i\right)\right]$	$-5.12 \leq x_i \leq 5.12$
Griewangk	$f(x) = 1 + \sum_{i=1}^{n} \left[\frac{x_i^2}{4000}\right] - \prod_{i=1}^{n} \left[cos\left(\frac{x_i}{\sqrt{i}}\right)\right]$	$-600 \leq x_i \leq 600$

4.2 Analysis of Results

Table 4 summarizes the results obtained by the proposed approach (HAEA) with different set of genetic operators after 20000 fitness evaluations. As expected, XUG has the best performance among the variations of HAEA tested (GU performs better than XUG only for the Rosenbrock function while XG performs

better than XUG only for the Griewangk function). Figure 4 shows the fitness and operator rates evolution using HaEa(XUG) variation. The behavior of the operator rates is similar to the behavior observed for binary functions; HaEa applies with high probability the best operator in early stages of the evolution while at the end operators are applied with similar probabilities. Clearly, crossover is very useful in locating the optimal solution of the Rastrigin, Schwefel, and Griewangk functions, but it is not so good for the Rosenbrock saddle function. This behavior can be due to the fact that crossover generates good offspring that are located in the narrow local solution valley of the Rosenbrock saddle function. Due to the adaptation mechanism, an individual can leave or move faster from such narrow optimal solution by trying other genetic operators.

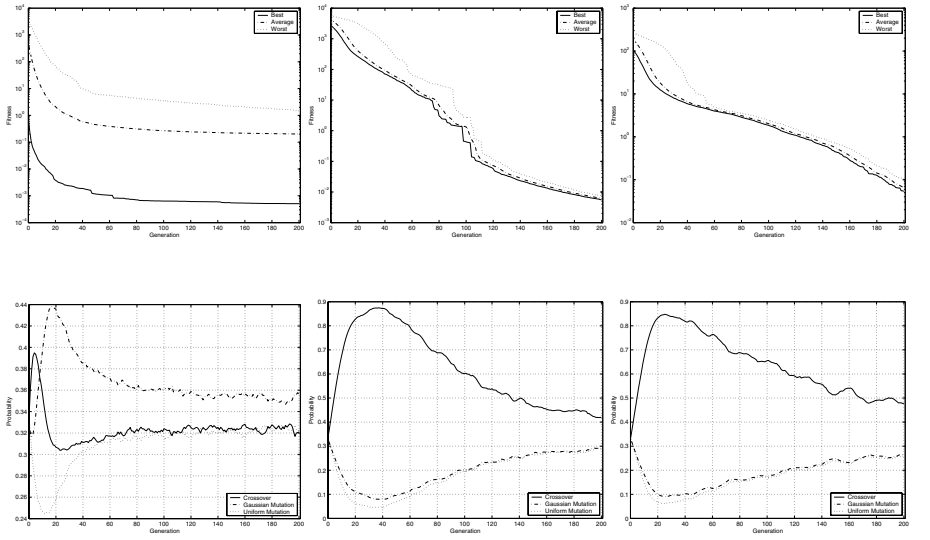
We implemented two versions of a generational algorithm, GGA(XG) and GGA(XU), and two versions of steady state algorithm, SSGA(XG) and SSGA(XU), in order to compare the performance of HaEa. A tournament of size 4 was used as parent selection mechanims, with single point real crossover and Gaussian (Uniform) mutation as genetic operators. For the steady-state approaches, we replaced the worst individual of the population with the best offspring generated from the selected parent after crossover and possibly mutation. Many different mutation and crossover rates were analized and the best results, obtained with 0.5 mutation rate and 0.7 crossover rate, are reported. As can be noticed, both or none of the genetic operators may be applied to a single individual for both SSGA and GGA. Table 5 summarizes the results obtained by HaEa, our implementation of SSGA and GGA, and compares them against some results reported by Digalakis and Margaritis [18][4], and Patton et all [19][5]. As shown, HaEa compares well in solving real encoding optimization problems. Moreover, HaEa(XUG) outperforms the SSGA and GGA approaches in all the tested functions. Variations of HaEa outperform the SSGA and GGA approaches in the Rosenbrock saddle function and in all the tested functions when Gaussian mutation is used. In general, the behavior of HaEa(XU) is similar to the behavior of SSGA(XU) and GGA(XU) for the Schwefel, Rastrigin and Griewangk functions.

Table 4. Solutions found by the tested EAs with the real functions

	Rosenbrock	**Schwefel**	**Rastrigin**	**Griewangk**
XUG	0.000509±0.001013	0.005599±0.011702	0.053614±0.216808	0.054955±0.029924
XU	0.004167±0.004487	1.362088±0.932791	0.240079±0.155614	0.530857±0.227458
XG	0.001322±0.003630	140.5647±123.7203	7.731156±3.223595	0.050256±0.025888
GU	0.000160±0.000258	201.9162±81.28619	6.320374±1.462898	1.586373±0.383703

[4] Digalakis and Margaritis varied the population size, the mutation rate and the crossover rate for a generational and a steady state genetic algorithm. The results reported in row 7 are the best results reported by Digalakis and Margaritis.

[5] Last row (Patton) reports the best result obtained by Patton et all [19] after 100000 evaluations.



(a) (b) (c)

Fig. 4. Performance evolution (first row) and Rates evolution (second row) using HaEa(MXT) on real functions. (a) Rosenbrock, (b) Schwefel, (c) Rastrigin

Table 5. Solutions found by the tested EAs on real functions

EA	Rosenbrock	Schwefel	Rastrigin	Griewangk
HaEa(XUG)	0.00051±0.00101	0.00560±0.01170	0.05361±0.21681	0.05495±0.02992
GGA(XU)	0.17278±0.11797	2.00096±1.21704	0.26500±0.15951	0.63355±0.24899
GGA(XG)	0.03852±0.03672	378.479±222.453	12.1089±5.01845	0.05074±0.02577
SSGA(XU)	0.06676±0.08754	0.88843±0.57802	0.12973±0.07862	0.32097±0.13091
SSGA(XG)	0.04842±0.04624	659.564±277.334	19.7102±7.80438	0.04772±0.02991
Digalakis [18]	0.40000000	-	10.000	0.7000
Patton [19]	-	-	4.8970	0.0043

5 Conclusions

In this paper, a new evolutionary algorithm (HaEa) was introduced. HaEa evolves the operator rates at the same time it evolves the solution. HAEA was tested on a variety of functions (using binary and real encoding), and the results indicated that HAEA is able to find good solutions.

References

1. J. H. Holland, *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.
2. M. Mitchell, *An intoduction to genetic algorithms.* MIT Press, 1996.
3. A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions in Evolutionary Computation*, vol. 3(2), pp. 124–141, 1999.
4. K. De Jong, *An analysis of the Behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.
5. J. Schaffer, R. Caruana, L. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization," in *Third International Conference on Genetic Algorithms*, pp. 51–60, 1989.
6. D. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," *Foundations of Genetic Algorithms*, no. 1, pp. 69–93, 1991.
7. T. Back and H. Schwefel, "An overview of evolutionary algorithms for parallel optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23, 1993.
8. F. Lobo, *The parameter-less genetic algorithm: rational and automated parameter selection for simplified genetic algorithm operation.* PhD thesis, Nova University of Lisboa, 2000.
9. L. Davis, "Adapting operator probabilities in genetic algorithms," in *Third International Conference on Genetic Algorithms and their Applications*, pp. 61–69, 1989.
10. B. Julstrom, "What have you done for me lately? adapting operator probabilities in a steady-state genetic algorithm," in *Sixth International Conference on Genetic Algorithms*, pp. 81–87, 1995.
11. A. Tuson and P. Ross, "Adapting operator settings in genetic algorithms," *Evolutionary Computation*, 1998.
12. W. Spears, "Adapting crossover in evolutionary algorithms," in *Evolutionary Programming Conference*, 1995.
13. M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *Transactions on Systems, Man and Cybernetics*, vol. 24(4), pp. 656–667, 1994.
14. T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, 1996.
15. D. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: motivation, analysis, and first results," *Complex Systems*, vol. 3, pp. 493–530, 1989.
16. S. Forrest and M. Mitchell, "Relative building blocks fitness and the building block hypothesis," *Foundations of Genetic Algorithms*, no. 2, 1993.
17. A. Simoes and E. Costa, "Transposition: a biologically inspired mechanism to use with genetic algorithms," in *Fourth International Conference on Neural Networks and Genetic Algorithms*, pp. 612–619, 1999.
18. J. Digalakis and K. Margaritis, "An experimental study of benchmarking functions for genetic algorithms," in *IEEE Conferences Transactions, Systems, Man and Cybernetics*, vol. 5, pp. 3810–3815, 2000.
19. A. Patton, T. Dexter, E. Goodman, and W. Punch, "On the application of cohort-driven operators to continuous optimization problems using evolutionary computation," *Evolutionary Programming*, no. 98, 1998.