The Best Genetic Algorithm I

A Comparative Study of Structurally Different Genetic Algorithms

Angel Fernando Kuri-Morales[1] and Edwin Aldana-Bobadilla[2]

[1] Instituto Tecnológico Autónomo de México
Río Hondo No. 1
México 01000, D.F.
México
akuri@itam.mx

[2] Universidad Nacional Autónoma de México
IIMAS
México 04510, D.F.
México
edwynjavier@yahoo.es

Abstract. Genetic Algorithms (GAs) have long been recognized as powerful tools for optimization of complex problems where traditional techniques do not apply. However, although the convergence of elitist GAs to a global optimum has been mathematically proven, the number of iterations remains a case-by-case parameter. We address the problem of determining the best GA out of a family of structurally different evolutionary algorithms by solving a large set of unconstrained functions. We selected 4 structurally different genetic algorithms and a non-evolutionary one (NEA). A schemata analysis was conducted further supporting our claims. As the problems become more demanding, the GAs significantly and consistently outperform the NEA. A particular breed of GA (the Eclectic GA) is superior to all other, in all cases.

Keywords: Global optimization, Genetic algorithms, Unconstrained functions, Schemata analysis.

1 Introduction

Optimization is an all pervading problem in engineering and the sciences. It is, therefore, important to rely on an optimization tool of proven efficiency and reliability. In this paper we analyze a set of optimization algorithms which have not been analyzed exhaustively before and achieve interesting conclusions which allow us to recommend one such algorithm as applicable to a large number complex problems. When attempting to assess the relative efficiency of a set of optimization algorithms one may take one of two paths: a) Either one obtains closed models for the algorithms thus allowing their parametric characterization [1], [2], [3] or b) One selects a set of problems considered to be of interest and compares the performance of the algorithms when measured vs. such a set. Modeling an algorithm is frequently a complex task and, more importantly, even slight changes in the algorithm lead to basically different models [4], thus making the purported characterization impractical. The second option, therefore, seems better suited for practical purposes. However, although there are many examples of such an approach (for instance see [5], [6], [7]) it is always true that a) The nature of the algorithms under study and their number are necessarily limited and b) The selection of the benchmarking functions obeys to subjective criteria. In this paper we choose to establish the relative efficiency of a set of genetic algorithms (GAs) which are structurally different from one another as will be discussed in the sequel. We have selected a set of such GAs and, for completeness, we have also included a particular non-evolutionary algorithm (the Random Mutation Hill Climber or RMH) whose efficiency has been reported in previous works [8], [9]. Many GAs are variations (i.e. different selection criteria [10], crossover strategies [11], population size [12], 13] relationship between Pc and Pm, [14], [15], etc.) of the initial one proposed by Holland (the so-called "Simple" or "Canonical" Genetic Algorithm [CGA] [16]) which do not significantly improve on CGA's overall performance. For benchmarking purposes the mentioned variations are not useful since they all share the same basic algorithmic structure. However there are GAs where the strategies to a) select, b) identify and c) recombine candidate solutions differ from the CGA's substantially. The purported changes impose structural differences between these algorithms which have resulted in remarkable performance implications. We have selected four GAs with this kind of diverse characteristics. We begin, in Section 2, by introducing the necessary notation; then presenting some concepts and definitions. In Section 3 we describe the five algorithms in our work. In section 4 we present the functions and results for a suite of problems that traditionally have been used for benchmarking purposes of optimization algorithms [17] [18]. In Section 5 we present our general conclusions.

2 Preliminaries

Throughout we use the following notation and definitions: A : Set of selected optimization algorithms; A_i : The *i-th* optimization algorithm (i.e. $A_i \in A$); $\vec{x}$: Vector in $\Re^n$; Ω : Feasibility region of the space $\Re^n$; B : Set defined as $B = \{0,1\}$; t: Iteration number such that $1 \le t \le G; t \in \mathbb{N};$ G: Upper bound on the number of iterations of A_i. Without loss of generality our discussion will be focused on numerical optimization problems. One such problem f is defined as:

$$\begin{aligned} &\textit{Minimize} && f(\vec{x}) && \\ &\textit{Subject to} && h_i(\vec{x}) = 0 && i = 1,...m \\ & && g_i(\vec{x}) \le 0 && i = m+1,...p \end{aligned} \qquad (1)$$

where $f(x): \Re^n \to \Re$ is the objective function, $h_i(\vec{x}) = 0$ and $g_i(\vec{x}) \le 0$ are constraint functions defining Ω. This means that if a vector $\vec{x}$ complies with all constraints it belongs to Ω. In a problem without constraints, such as the ones discussed here, all vectors $\vec{x}$ lie within Ω.

We briefly pause to define what we understand as a genetic algorithm. Elsewhere [20], it has been argued that an algorithm is "genetic" when it exhibits implicit parallelism. Instead, we list the characteristics an iterative algorithm must have to be considered "genetic". Implicit parallelism is a consequence of these.

Definition 1:
A genetic algorithm is one which satisfies the following conditions:

1. It works on an n-dimensional discrete space D defined in $\mathbb{N}^n$ rather than in $\mathbb{R}^n$.
2. It traverses D searching an approximation of the optimum vector $\vec{x}$ of (1) by simultaneously analyzing a finite set $S(t) \in D$ of candidate solutions.
3. The elements of $S(t) = \{s_1(t), s_2(t), ..., s_n(t)\}$ are explicitly encoded in some suitable way.
4. The information regarding the partial adequacy of the elements in $S(t)$ is extracted by solving the optimization problem for all $s_i(t)$.
5. The qualified elements of $S(t)$ are analyzed to select an appropriate subset in order to improve the search in the problem's space.
6. Selected sections of the codes of $s_i(t)$ are periodically combined.
7. Selected elements of the codes of the $s_i(t)$ are periodically and randomly altered.
8. A subset of the best solutions of $S(t)$ is preserved for all the future steps of the algorithms.
9. The algorithm cycles through steps 4-8 until a stopping condition is met.

The algorithms selected for this study satisfy all of the characteristics above and, therefore, may be aptly considered to be genetic in a broader sense then the one implied by the frequently cited "bio-inspired" analogy. In fact, this analogy, attractive as it may seem, frequently distracts the attention of the user from the basic efficiency elements which any optimization algorithm should incorporate. These issues must supersede other considerations when determining the desirability of one algorithm over others.

Consequently, set A includes the following GAs:

a) An elitist canonical GA (in what follows referred to as TGA [eliTist GA]) [21].
b) A Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation algorithm (CHC algorithm) [22].
c) An Eclectic Genetic Algorithm (EGA) [23].
d) A Statistical GA (SGA) [24] [25].

3 Selected Genetic Algorithms

It is frequent to cite the variations of the GAs by their "popular name". However, in so doing one incurs in the risk of not being precise on the details of the algorithm. One of the basic tenets of this paper is that even small variations lead to potentially important differences in their behaviors. For this reason, we now include the pseudo-codes of the algorithms in our study. Keep in mind that our results refer to their precise implementation and no others. As a case in point, when discussing SGA (the Statistical GA) it may be easy to confuse it with EDA (Estimation of Distribution Algorithm). However, in EDA no mutation is explicitly included, whereas in SGA it is (see the code below)

In the description of the algorithms which follows a) We denote the arguments $\vec{x} = (x_1,...,x_k)$ with x_i and the corresponding fitness function $f(\vec{x}) = f(x_1,...,x_k)$ with $f(x_i)$, b) The function $f(x_i)$ to be optimized is numerical, c) We aim to minimize $f(x_i)$, and d) The arguments x_i of the fitness function $f(x_i)$ are encoded in binary.

Let $G \equiv$ number of generations; $n \equiv$ number of individuals; $I(n) \equiv$ *the n-th individual;* $L \equiv$ length of the chromosome; $P_C \equiv$ probability of crossover; $P_M \equiv$ probability of mutation.

By "*Generation of a random population*" we mean that, for a population of n individuals each of whose chromosome's length is L we make

```
for i = 1 to n
    for j=1 to L
        Generate a uniform random number $0 \le \rho < 1$.
        If $\rho < 0.5$ make $bit_j \leftarrow 0$; else make $bit_j \leftarrow 1$.
    endfor
endfor
```

3.1 Elitist Canonical GA (TGA)

This is the classical CGA with two provisions: a) The best individual is kept along the whole process forming part of the evolving population and b) In step 3 of the algorithm

$$\varphi(x_i) = f(x_i) + |\, min(f(x_i))\,| + avg\big(|\, f(x_i)\,|\big) \tag{A.1}$$

is used. These two steps ensure that no fitness value is negative making the proportional selection scheme always feasible (see [28, 29, 30]).

```
0. Make $k \leftarrow 1$.
1. Generate a random population
2. Select randomly an individual from the population (call it best).
    Make $f(best) \leftarrow \infty$.
```

```
3. Evaluate.
      for i=1 to n
            Evaluate $f(x_i)$.
            Make $f(x_i) \leftarrow \varphi(f(x_i))$.
            If $f(x_i) < f(best)$ make $best \leftarrow x_i$ and $f(best) \leftarrow f(x_i)$
      endfor
4. If $k = G$ return best and stop.
5. Selection
      Make $F = \sum_{i=1}^{n} f(x_i)$
      for i = 1 to n; $PS_i = \frac{f(x_i)}{F}$; Endfor
      for i =1 to n; Select $I(i)$ with probability $PS_i$.; endfor
6. Crossover
for i = 1 to n step 2
      Randomly select two individuals (say $I(X)$ and $I(Y)$) with probabilities $PS_X$
      and $PS_Y$, respectively.
      Generate a uniform random number $0 \le \rho < 1$.
      If $\rho \le P_C$ do
        • Randomly select a locus $\ell$ of the chromosome; Pick the leftmost L-$\ell$
          bits of $I(X)$ and the rightmost $\ell$ bits of $I(Y)$ and concatenate them
          to form the new chromosome of $I(X)$; Pick the leftmost L-$\ell$ bits of
          $I(Y)$ and the rightmost $\ell$ bits of the previous $I(X)$ and concatenate
          them to form the new chromosome of $I(Y)$
      Make $I(i) \leftarrow I(X)$; $I(i+1) \leftarrow I(Y)$.
endfor
7. Mutation
for i = 1 to n
      Select $I(i)$
      for j=1 to L
            Generate a uniform random number $0 \le \rho < 1$.
            If $\rho \le P_M$ make $bit_j \leftarrow \overline{bit_j}$.
      endfor
endfor
8. Make $k \leftarrow k+1$ and go to step 3.
```

3.2 Cross Generational Elitist Selection, Heterogeneous Recombination and Cataclysmic Mutation GA (CHC)

This algorithm focuses on maintaining diversity while retaining the characteristics of the best individuals. Inter-generational survival-of-the-fittest is attempted by unbiased parent selection. Furthermore it tries to maintain diversity implementing the so-called HUX crossover (Half, Uniform X-over) and introducing cataclysmic mutations when the population's diversity falls below a pre-defined threshold (see [20]).

```
0. Make
       $\ell \leftarrow 0$
       $threshold \leftarrow L/4$
1. Generate a random population.
2. Evaluate $f(x_i)\ \forall i$
3. $f(best) \leftarrow min(f(x_i))\ \forall i$
    $best \leftarrow I(i \mid f(x_i)$ is best$)$
4. $\ell \leftarrow \ell + 1$
   If $\ell = G$ return best and stop.
5. Copy all individuals from population P into set C
6. [HUX Crossover]
Let $Bit_{xy}$ denote the y-th bit of individual x
for i=1 to n/2
       Randomly select individuals $I_X$ and $I_Y$ from C
       $hammingXY \leftarrow 0$
       for $j \leftarrow 1$ to L
                 if bit j $(I_X) \neq$ bit j$(I_Y)$; DiffXY[j]=true;
                   $hammingXY \leftarrow hammingXY+1$; else DiffXY[j]=false; f
       endfor
       if ($hammingXY/2 \leq threshold$)
                 eliminate C(X) and C(Y) from C
       else
                 $mutated \leftarrow 0$
                 while ($mutated < hammingXY/2$)
                          $j \leftarrow$ random number between 1 and L
                          if DiffXY[j]
                                   Interchange the j-th bit of $I_X$ and $I_Y$
                                   $mutated \leftarrow mutated+1$; DiffXY[j] $\leftarrow$ false
                 endwhile
endfor
Evaluate $f(x_i)$ in C( i) $\forall i$
Make $P' = P \cup C$
Sort P' from best to worst
$P' \leftarrow$ Best n individuals from P'; $f(best) \leftarrow f(x_1)$; $best \leftarrow P_1'$
if $P = P'$
       $threshold \leftarrow threshold-1$
       if threshold=0
                 for i=2 to n
                          Select the i-th individual
```

```
                    for j=1 to L
                              Generate a uniform random number $0 \le \rho < 1$
                              If $\rho \le 0.35$ make $bit_j \leftarrow \overline{bit_j}$
            threshold $\leftarrow$ L/4
    P $\leftarrow$ P'; go to 4
```

3.3 Eclectic GA (EGA)

This algorithm uses deterministic selection, annular crossover, uniform mutation and full elitism (a strategy akin to $\mu + \lambda$ selection of evolutionary strategies [31]). The probabilistic nature of EGA is restricted to parameters P_C and P_M. In EGA avoidance of premature convergence is achieved by a two-fold strategy. First, the $2n$ individuals from the last two generations are ordered from best to worst and only the best n are allowed to survive. Then the individuals are deterministically selected for crossover by mixing the best with the worst (*1* with *n*), the second with the second worst *(2* and *n-1*, . . .,), and so on. In this way *n* new individuals are generated. As the iterations proceed, the surviving individuals become the top elite of size *n* of the whole process. Annular crossover (equivalent to two-point crossover) is preferred because it makes the process less dependent on a particular encodings. This algorithm was first reported in [26] and included self-adaptation and periodic cataclysmic mutation. Later studies [27] showed that neither of the two mechanisms was necessary. EGA is relatively simple, fast and easy to program.

```
0. B2M $\leftarrow \lceil nL \times P_M \rceil$ ( Expected number of mutations per generation)
1. i $\leftarrow$ 1
2. Generate a random population
3. Evaluate the population.
4. [ Duplicate Population]
for j = 1 to n
    I(n+j) $\leftarrow$ I(j)
    fitness(n+j) $\leftarrow$ fitness(j)
endfor
5. [ Deterministic Selection Annular Crossover]
for j=1 to n/2
    Generate a uniform random number $0 \le \rho < 1$
    If $\rho \le P_c$
            Generate a random number $1 \le \rho < L/2$
            Interchange the semi-ring starting at locus $\rho$ between I(j)
             and I(n-j-1)
    endif
endfor
5. [Mutation]
for j=1 to B2M
```

```
        Generate uniform random numbers $0 \le \rho_1, \rho_2 < 1$
        Mutate Bit $\lceil \rho_2 L \rceil$ of I($\lceil \rho_1 n \rceil$)
    endFor
    6. [Evaluate the New Individuals]
    Calculate fitness($x_i$) for i=1,...,n
    7. [$\mu + \lambda$ Selection]
    Sort the 2n individuals by their fitness, ascending
    8. i ← i+1
      if i = G return I(1) and stop
     Go to 3
```

3.4 Statistical GA (SGA)

In this case the algorithm takes advantage of the fact that the average behavior of traditional TGA may be achieved without actually applying the genetic operators but, rather, statistically simulating their behavior [24]. SGA starts by generating the initial population's (P_0) individuals randomly. The fitness for each individual is calculated as per (A.1). It is then easy to determine its relative fitness $\Phi(x) \leftarrow f(x_j) / \sum_j f(x_j)$ which, immediately, induces a partial ordering in the population according to the value of $\Phi(x)$. Once this is done, the so-called *probabilistic genome* (PG) is calculated. In this genome, the probability that the *k-th* bit of a genome attains a value of 1 is derived from

$$P_k = \sum_j \Phi_j b_{jk} \quad k = 1,\dots,L \tag{A.2}$$

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. This is equivalent to defining a set of probability distribution functions (*pdfs*); one for each of the *L* bits in the genome. These *pdfs* are Bernoulli distributed and, initially, may have rather large variances (σ^2). 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*. Every new population is also Bernoulli distributed but with an increasingly small σ. Eventually the *pdfs* of the final population will have a Bernoulli distribution with $\sigma \approx 0$, implying approximate convergence. In a strict sense, the SGA avoids the need to include explicit mutation provisions. Preliminary tests showed that premature convergence is avoided if such provisions are made. The whole process may be seen as a search for a crisp encoding of the solution with a set of fuzzy bits. Each bit is progressively de-fuzzyfied in consecutive generations.

```
0. Make $k \leftarrow 1$;
   $B2M \leftarrow \lceil nL \times P_M \rceil$ ( Expected number of mutations per generation)
1. Generate a random population
2. Select randomly an individual from the population (call it best). Make $f(best) \leftarrow \infty$ .
3. [Get probabilistic genome]
      $PopFit \leftarrow 0$
      for i=1 to n
            Evaluate $f(x_i)$
            If $f(x_i)<f(best)$
                  $best \leftarrow I(i); f(best) \leftarrow f(x_i)$
            endif
            Make $f(x_i) \leftarrow \varphi(f(x_i))$; $PopFit \leftarrow PopFit + f(x_i)$
      Endfor
      for i=1 to n
            $RelFit_i \leftarrow f(x_i)/PopFit$;
      endfor
      for i=1 to L
            $PG_i \leftarrow 0$;
            for j=1 to n
                  if $bit_{ji} = 1 \rightarrow PG_i \leftarrow PG_i+RelFit_j$
            endFor
      endFor
4. [Get new population]
      [Probabilistic Individuals]
      for i = 1 to n
            for j = 1 to L
                  Generate a uniform random number $0 \le \rho < 1$
                  if $\rho \le PG_j \rightarrow Bit_{ij} \leftarrow 1$; else $Bit_{ij} \leftarrow 0$
            endFor
      endFor
      [Mutate Individuals]
      for i=1 to B2M
            Generate uniform random numbers $0 \le \rho_1, \rho_2 < 1$
            Mutate Bit $\lceil \rho_2 L \rceil$ of $I(\lceil \rho_1 n \rceil)$
      endFor
5. $k \leftarrow k+1$
   If $k = G$ return best and stop; else Go to step 3
```

3.5 Random Mutation Hill-Climber (RMH)

This algorithm is the only non-evolutionary one considered in this study. In general, a "hill-climber" is an algorithm which attempts, iteratively, to improve on its best found value by refining the last attempt.

```
1. [Generate the individual]
for i=1 to L
        Generate a uniform random number $0 \le \rho < 1$.
    If $\rho < 0.5$ make $bit_i \leftarrow 0$; else make $bit_i \leftarrow 1$.
endfor
Make best $\leftarrow$ I(0)
    BestFit $\leftarrow \infty$
2. [Iterate]
for i = 1 to G
    [Evaluate the individual]
    $f(i) \leftarrow fitness(x_i)$
    if fitness(i)<BestFit $\rightarrow$ best $\leftarrow$ I($x_i$); BestFit $\leftarrow$ fitness($x_i$)
    [Mutate]
     Generate a uniform random number $1 \le k \le L$; Make $bit_k \leftarrow \overline{bit_k}$
endfor
```

For the case of RMH, $S(t)$ is a unitary set such that $S(t) = \{s(t)\}$ where $s(t)$ is also a binary encoded candidate solution which is chosen at random and whose fitness is evaluated. We explored the behavior of the A_i's taking snapshots of their progress in steps of 50 generations up to 800. A GA works with several candidate solutions that allow it to explore different regions of Ω in parallel. On the other hand, the RMH works with a single candidate solution that allows it to explore a single region of Ω. The number of iterations of RMH needed for convergence, for this reason, differs significantly from that of a GA. For benchmarking purposes, therefore, we established the following standard.

1) Let M be the number of candidate solutions for a GA. Thus, for A_i it holds that $|S(t)| = M$
2) The upper bound on the number of iterations of a RMH is set to $M \times G$.
3) The upper bound on the number of iterations of any GA is set to G.

Any A_i will, therefore, approach the solution to a problem *f* in at most *G* iterations. For a detailed discussion of the algorithms see Appendix A.

4 Selected Functions

In this section we discuss the behavior of the algorithms for selected functions whose minima are known and, therefore, allow us to establish a measure of effectiveness relative to the best value found by the algorithm. The evaluation of all algorithms in *A* is based on a set of unconstrained functions (UF) which have some properties (multimodality, non-convexity, non-linearity, etc.) that make them good choices for benchmarking purposes, For reasons of space we may only succinctly present 6 of the 23 functions considered in this study. 1) **Hansen Function**. Unimodal; it is defined in a *n*-dimensional space $\forall n \in \mathrm{N}$: $f(n,x)=\sum_{i=0}^{4}(i+1)\cos(ix_0+i+1)\sum_{j=0}^{4}(j+1)\cos((j+2)x_1+j+1)$. Ω is $|x_i| \leq 10$; the known minimum is -176.54. 2) **De Jong's Function**. Continuous, convex and unimodal: $f(x)=\sum_{i=1}^{n}x_i^2$. Ω is $-5.12 \leq x_i \leq 5.12,\quad i=1,...,n$; the known minimum is 0: 3) **Rotated hyper-ellipsoid function**. Continuous, convex and unimodal: $f(x)=\sum_{i=1}^{n}\left(\sum_{j=1}^{i}x_j\right)^2$; Ω is -65536 <= x_i <= 65536; the known minimum is 0. 4) **Rosenbrock's valley function.** The global optimum lies inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial. However convergence to the global optimum is difficult and hence this problem has been frequently used to test the performance of optimization algorithms: $f(x)=\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(1-x_i)^2$. Ω is $-2.048 \leq x_i \leq 2.048,\quad i=1,...,n$; the known minimum is 0. 5) **Rastrigin's function.** It is based on the function of De Jong with the addition of cosine modulation in order to produce frequent local minima. The function is highly multimodal: $f(x)=10n+\sum_{i=1}^{n}[x_i^2-10\cos(2\pi x_i)$. Ω is $-5.12 \leq x_i \leq 5.12,\quad i=1,...,n$; the known minimum is 0. 6) **Schwefel's function.** It is deceptive in that the global minimum is geometrically distant, over the parameter space, from the next best local minima. Therefore, the search algorithms are potentially prone to convergence in the wrong direction: $f(x)=\sum_{i=1}^{n}[-x_i\sin(\sqrt{|x_i|}]$. Ω is $-500 \leq x_i \leq 500,\quad i=1,...,n$; the known minimum is -418.9828*n*.

All 23 functions have known optima. This allows us to define a relative measure of performance for A_i as follows:

$$Q(A_i,f_j)=\frac{y_j*-y_j}{y_j*} \tag{2}$$

where y_j* is the known optimum of f_j and y_j is the best value found by A_i. We ran every algorithm 100 times for every problem and obtained its average performance. We obtained this average performance for all A_i with *G* = 800. We got snapshots of the process every 50 generations. In Figure 1 we show the corresponding results.

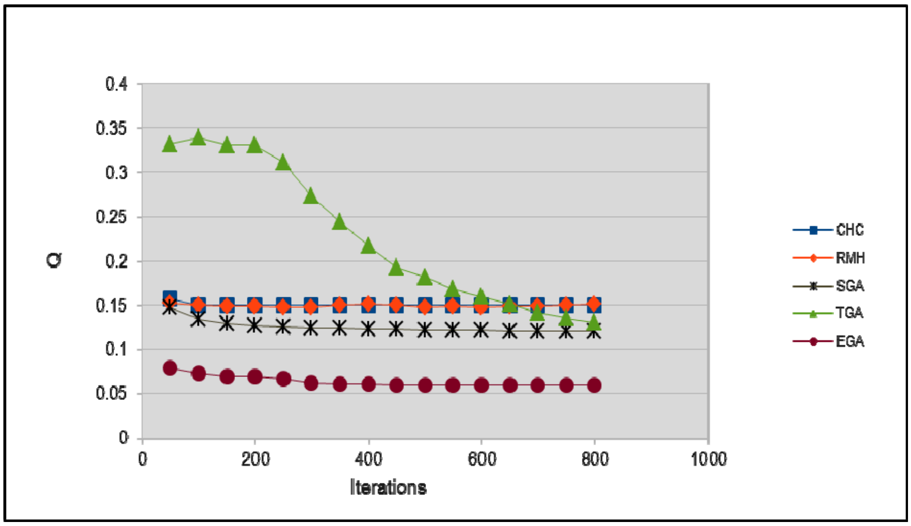

Fig. 1. Average Performance for UF

Notice that in these functions all of the GAs outperform the RMH only marginally, with the exception of EGA which is considerably better. Also notice that TGA (the canonical GA) is able to reach an acceptable value only after a large number of generations. From a further analysis we determined how the algorithms identify larger order schemata. We indicate that A_i is better than A_j when the order of the schemata of A_i is larger than the order of the schemata of A_j. (see Fig. 2). Consistent with the previous results, TGA is the slowest algorithm to identify schemata of higher order. That is, the sluggish nature of TGA is due to the fact that it spends many more generations in identifying the better individuals in terms of their schemata order.

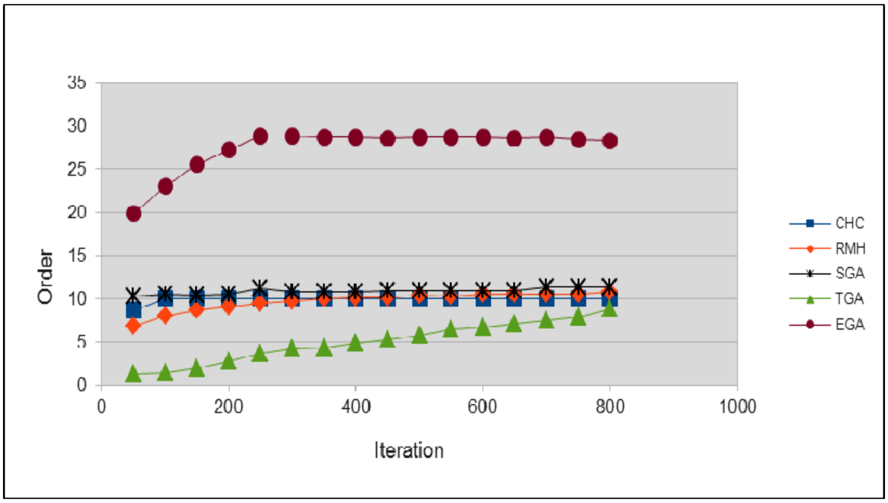

Fig. 2. Average order of the schemata for UF

5 Conclusions

The table for the suite of unconstrained problems (see Table 1) show that EGA outperforms the rest of the algorithms; in this case, notably so. TGA is, by far, the worst of the algorithms. Again RMH's behavior is close to SGA's and CHC's.

Table 1. Average minimum for the suite of unconstrained problems

Algorithm	Average Minimum	Relative Efficiency
EGA	0.0635	100.00%
SGA	0.1260	50.43%
RMH	0.1491	42.60%
CHC	0.1501	42.32%
TGA	0.2272	27.96%

The best algorithm is EGA. Considering the wide size and variety of the set of problems we can say that, with high probability, the EGA is the best global optimization algorithm in our study.

In concluding:

1. In all experiments, the EGA exhibited the best performance. We know that EGA is a good alternative in problems with hard search spaces (e.g. non-convex, constrained or highly dimensional spaces) .
2. From a large set of runs it is possible to obtain a practical estimate of the schemata found by the algorithms.
3. The analysis of schemata order of the algorithms leads to results consistent with the previous one.
4. A similar analysis including other optimization techniques (e.g. Simulated annealing, Evolution strategies, Ant colony optimization, Particle swarm optimization, Differential evolution, etc.) may be easily implemented.

References

1. MacKay, B.: Mathematics and Statistics Models, `http://serc.carleton.edu/introgeo/mathstatmodels/index.html`
2. Mooney, D., Swift, R., Mooney, D.: A Course in Mathematical Modeling. Cambridge University Press (1999)
3. Kolesárová, A., Mesiar, R.: Parametric characterization of aggregation functions. Fuzzy Sets Syst. 160(6), 816–831 (2009)
4. Back, T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms, ch. 4, pp. 149–159 (1996)
5. Coello, C.: A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowledge and Information Systems 1, 269–308 (1998)
6. De Jong, K.: An analysis of the behavior of a class of genetic adaptive systems, Diss. PhD thesis, Dept. of Computer and Comm. Sciences, Univ. of Michigan, Ann Arbor, MI (1975)

7. Endre, A., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation 3(2), 124–141 (1999)
8. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press (1996)
9. Mitchell, M., Holland, J., Forrest, S.: When Will a Genetic Algorithm Outperform Hill Climbing? In: Advances of Neural Information Processing Systems, vol. 6, pp. 51–58. Morgan Kaufmann (1994)
10. Baker, J.: Adaptive selection methods for genetic algorithms. In: Grefenstette, J. (ed.) Proceedings of the 1st International Conference on Genetic Algorithms and their Applications, pp. 101–111. Lawrence Earlbaum Associates, N.J. (1985)
11. Spears, W.M., Anand, V.: A Study of Crossover Operators in Genetic Programming. In: Raś, Z.W., Zemankova, M. (eds.) ISMIS 1991. LNCS, vol. 542, pp. 409–418. Springer, Heidelberg (1991)
12. Bäck, T.: Self-Adaptation in Genetic Algorithms. In: Varela, F., Bourgine, P. (eds.) Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, pp. 263–271. MIT Press (1991)
13. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press (1996)
14. De Jong, K.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral Dissertation, University of Michigan (1975)
15. De Jong, K.A., Spears, W.M.: An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms. In: Schwefel, H.-P., Männer, R. (eds.) PPSN 1990. LNCS, vol. 496, pp. 38–47. Springer, Heidelberg (1991)
16. Holland, J.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
17. Pohlheim, H.: GEATbx: Genetic and Evolutionary Algorithm Toolbox for use with MATLAB Documentation, Version 3.80 (released December 2006), `http://www.geatbx.com/docu/index.html`
18. Digalakis, J., Margaritis, K.: An experimental study of Benchmarking functions for genetic algorithms. Intern. J. Computer Math. 79(4), 403–416 (2002)
19. Vose, D.: Generalizing the notion of schema in genetic algorithms. Artificial Intelligence 50(3), 385–396 (1991)
20. Eshelman, L.: The CHC Adaptive Search Algorithm. How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In: Rawlins, G. (ed.) FOGA-1, pp. 265–283. Morgan Kaufmann (1991)
21. Rudolph, G.: Convergence Analysis of Canonical Genetic Algorithms. IEEE Transactions on Neural Networks 5(1), 96–101 (1994)
22. Eshelman: Op. cit. (1991)
23. Rezaee Jordehi, A., Hashemi, N., Nilsaz Dezfouli, H.: Analysis of the Strategies in Heuristic Techniques for Solving Constrained Optimisation Problems. Journal of American Science 8(10) (2012)
24. Sánchez-Ferrero, G.V., Arribas, J.I.: A Statistical-Genetic Algorithm to Select the Most Significant Features in Mammograms. In: Kropatsch, W.G., Kampel, M., Hanbury, A. (eds.) CAIP 2007. LNCS, vol. 4673, pp. 189–196. Springer, Heidelberg (2007)
25. Kuri-Morales, A.: A statistical genetic algorithm. In: Proc. of the 3rd National Computing Meeting, ENC 1999, Hgo., México, pp. 215–228 (1999)
26. Kuri-Morales, A., Villegas-Quezada, C.: A universal eclectic genetic algorithm for constrained optimization. In: Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing, vol. 1 (1998)

27. Kuri-Morales, A.F.: A methodology for the statistical characterization of genetic algorithms. In: Coello Coello, C.A., de Albornoz, Á., Sucar, L.E., Battistutti, O.C. (eds.) MICAI 2002. LNCS (LNAI), vol. 2313, pp. 79–88. Springer, Heidelberg (2002)
28. Back, T.: Evolutionary algorithms in theory and pactice: evolution strategies, evolutionary programming, genetic algorithms, ch. 4, pp. 149–159 (1996)
29. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press (1996)
30. Vose, D.: The Walsh Transform and the Theory of the Simple Genetic Algorithm. In: Pal, S., Wang, P. (eds.) Genetic Algorithms for Pattern Recognition. CRC Press (1996)
31. Rowhanimanesh, A., Sohrab, E.: A Novel Approach to Improve the Performance of Evolutionary Methods for Nonlinear Constrained Optimization. Advances in Artificial Intelligence (2012)