ADAPTIVE EVOLUTIONARY ALGORITHMS ON UNITATION, ROYAL ROAD AND LONGPATH FUNCTIONS

J. Neal Richter, John Paxton
Computer Science Department, Montana State University,
357 EPS, Bozeman, Montana 59714
http://www.cs.montana.edu
{richter, paxton}@cs.montana.edu

ABSTRACT
Genetic algorithms (GAs) are powerful tools that allow engineers and scientists to find good solutions to hard computational problems using evolutionary principles. The classic genetic algorithm suffers from the configuration problem, the difficulty of choosing optimal parameter settings. Genetic algorithm literature is full of empirical tricks, techniques, and rules of thumb that enable GAs to be optimized to perform better in some way by altering the GA parameters. However these techniques are often analyzed on only a narrow set of fitness functions. This paper is a first empirical step in analyzing several parameter adaptive techniques on the unitation class of fitness functions, where fitness is a function of the number of ones in the binary genome.

KEY WORDS
Genetic Algorithms, Neural-Fuzzy-Genetic Systems, Adaptive and Optimal Control

1. Introduction

The genetic algorithm (GA) is a powerful tool that allows us to find solutions to problems that are poorly understood, have many interdependencies, or are otherwise too complex to solve directly. Modeled after natural selection and the evolutionary process, the basic GA is a population of possible solutions that is analyzed for fitness and recombined to form the next generation. This is repeated until a desirable solution is found [1].

The GA can be configured in many ways, and these different setups can have strong effects on the solutions found. Crossover operators, mutation operators, selection operators, and population size are just a few of the many parameters that are available to be modified or optimized to fit a given fitness function.

The GA literature is crowded with papers on various techniques researchers have found to set up these parameters for various problem domains. Other researchers have found interdependencies between operators and formulated helpful heuristics to follow when designing a GA system for a specific domain [1].

Many researchers agree that the classic GA has serious flaws. It can be a robust method, although when setup improperly that property diminishes. Chief among those pitfalls is the inability of the classic GA to adapt to the changing characteristics of the solution space as the population moves around in it.

For example, a GA can have difficulty if the inherent step-size of the algorithm (as defined by the operators) is larger than the optimal step size of fitness landscape. Hill climbing algorithms can suffer from the same problem.

This paper will concentrate on mutation-only Evolutionary Algorithms and use several methods to adapt the mutation rate. The experiments are restricted to binary genomes on the unitation class of fitness functions and one example of each of the Royal Road and LongPath families of functions. Unitation functions are fitness functions defined only by the number of ones in the binary genome.

The Royal Road functions were designed by Holland and coworkers to highlight the building block approach the GA was thought to take in problem solving [1]. Horn et al. designed the LongPath [2] also to highlight the supposed building block explanation of GA function.

This empirical study is a preliminary step in a theoretical analysis of these types of adaptive operators. The experiments show a general failure of adaptive methods on simple-looking unitation functions, while the adaptive methods fare well on the well-known and challenging Royal Road problem. The adaptive method also shows improvement on the LongPath problem, which is fairly easy for the basic GA.

2. Types of Parameter Adaptation

Hinterding et al. [3] surveyed general GA parameter adaptation. They classify adaptation into four types and four levels. The types are:

- static (unchanging parameters)
- deterministic dynamic (parameters changed with a deterministic function)

- dynamic adaptive (parameters changed with feedback)
- dynamic self-adaptive (adaptation method encoded into chromosome).

The levels are:

- environmental (changing fitness function response to individuals with a heuristic)
- population (any parameters affecting entire population)
- individual (mutation rate, age of individuals)
- component (varying parameters for individual genes).

2.1 Deterministic Dynamic

Deterministic mutation schedules are well known to the GA community. They have been used for decades in the Evolutionary Strategies literature [4]. Bäck and Schütz [5] introduced the deterministically decreasing function given in Equation 1. This function works on the theory that higher mutation rates in early generations are good for exploration and lower mutation rates in later generations are good for exploiting the local fitness landscape. T is the total number of generations the GA will be run for and l is the length of the chromosome. The mutation rate given by $p_{BS}(t)$ is bounded by (0, 1/2]. This function showed good results on hard combinatorial optimization problems. Note that it would be advantageous to floor the function near $1/2l$. Mutation rates lower than this are rarely effective.

$$p_{BS}(t) = \left(2 + \frac{l-2}{T-1}t\right)^{-1} \quad (1)$$

Droste [6] uses a cyclic mutation operator. The idea of this operator is to try a number of different probabilities in a repeated cycle, giving the GA many chances to use different probabilities during the various natural stages of the GA. The bounds of $p_{Dr}(t)$ are [1/l, 1/2] and the method cycles over log l different mutation probabilities.

$$p_{Dr}(t) = 2p(t-1) \quad \text{if } p(t) > 1/2, \text{ set } p(t) = 1/n \quad (2)$$

2.2 Dynamic Adaptive

Thierens [7] introduced two mutation adaptive schemes. The Constant Gain scheme is loosely patterned after the Manhattan learning algorithm. Equation 3 contains the mutation update specification. The exploration factor ω and learning factor λ usually have different values ($1 < \lambda < \omega$). To avoid oscillations in the learning process λ is restricted via $\omega > \lambda$. Example values are λ=1.1 and ω=1.5.

Thierens also introduced the Declining Adaptive mutation scheme. This variant of the first scheme is intended to promote a more aggressive step size while suppressing the wild oscillations that can happen with high learning rate λ. Equation 4 details the scheme.

Thierens' Constant Gain adaptive mutation rule

1. Mutate the current individual (x, p_m) three ways
$$\begin{aligned} &M(x, p_m/\omega) \rightarrow (x_1) \\ &M(x, p_m) \rightarrow (x_2) \\ &M(x, \omega p_m) \rightarrow (x_3) \end{aligned} \quad (3)$$
2. Select the fittest individual and corresponding new mutation rate

$$MAX\{(x, p_m), (x_1, p_m/\lambda), (x_2, p_m), (x_3, \lambda p_m)\}$$

Thierens' Declining Adaptive mutation rule

1. Mutate the current individual (x, p_m) three ways
$$\begin{aligned} &M(x, \omega p_m) \rightarrow (x_1) \\ &M(x, p_m) \rightarrow (x_2) \\ &M(x, \omega p_m) \rightarrow (x_3) \end{aligned} \quad (4)$$
2. Decrease the mutation rate of the parent
$$(x, p_m) \rightarrow (x, \gamma p_m)$$
3. Select the fittest individual and corresponding new mutation rate

$$MAX\{(x, \gamma p_m), (x_1, \lambda p_m), (x_2, p_m), (x_3, \lambda p_m)\}$$

The difference between the two schemes is that the second contains no method to increase the mutation rate and that the current mutation rate will decrease unless there is success at the current rate. Factor bounds are as follows: λ>1, ω>1 and 0.9<γ<1. Typical settings are λ=2.0, ω=2.0 and γ=0.95.

Rechenberg introduced the '1/5 success rule' for Evolutionary Strategies [4]. The basic idea is to adapt the mutation rate to balance the percentage of fitness-beneficial mutations at 1/5. This rule, shown in Equation 6, is applied periodically and not during every generation. A typical value for the learning rate is λ=1.1.

Rechenberg's 1/5 success rule

$$\begin{aligned} &\text{if } \varphi(k) < 1/5 \rightarrow (p_m/\lambda) \\ &\text{if } \varphi(k) = 1/5 \rightarrow (p_m) \\ &\text{if } \varphi(k) > 1/5 \rightarrow (\lambda p_m) \end{aligned} \quad (5)$$

Where $\varphi(k)$ is the percentage of sucessful mutations over x generations and λ is the learning rate.

2.3 Dynamic Adaptive with Fuzzy Logic

Shi, et al. [8] introduced a fuzzy logic rule set for adapting the mutation and crossover rate. Figure 1 gives

the rule set associated with the mutation rate. This rule set does require providing fuzzy membership functions for the various metrics and mutation rates.

IF *BF* is low or medium THEN *MR* is low
IF *BF* is medium and *UN* is low THEN *MR* is low
IF *BF* is medium and *UN* is medium THEN *MR* is medium
IF *BF* is high and *UN* is low THEN *MR* is low
IF *BF* is high and *UN* is medium THEN *MR* is medium
IF *UN* is high THEN *MR* is random(high, med, low)

Figure 1. Shi et al. Fuzzy adaptive rule set for mutation. *BF* = Best Fitness, *UN* = Number of generations since last *BF* change, *MR* = Mutation Rate

Improvement over a GA with fixed parameters was shown using this rule set for evolving classifier systems. For the purposes of this paper the Sugeno method output functions was used. The Sugeno fuzzy method [9] assigns either constant or linear functions to the output of the fuzzy inferencing method, rather than defining fuzzy membership functions for output. Here this means that three different mutation rates were chosen, one for each fuzzy output.
The original Shi et al. rule set used a metric called 'variance of fitness' with *high*, *med* and *low* fuzzy memberships. For this paper that metric was eliminated and three rules were combined to form the last rule in Figure 1.

3. Functions of Unitation

Unitation functions are fitness functions where fitness is a function of the count of ones in a chromosome x:$\{1,0\}^l$, where l is the length of the chromosome of the GA. All fitness values are non-negative:

$$u(x):\{0,1\}^l \rightarrow \mathbb{R}^+ \quad (6)$$

An example function for 3 bits is

$$u(0) = 3 \;\; u(1) = 2 \;\; u(2) = 1 \;\; u(3) = 4$$

This definition allows us to reduce the dimensionality of any analysis from 2^l to $(l+1)$. This is useful in that theoretical analysis of these functions is computationally easier while still using a function with a complex Hamming landscape.

3.1 Example Functions

The three fitness functions given in Equation 7 and pictured in Figure 1 are called NEEDLE, BINEEDLE and ONEMAX, and have been theoretically studied for fixed parameter simple GAs by Rowe [10], Wright [11] and Richter et al. [12]. The ONEMAX fitness function has been called the 'fruit fly' of GA research [7]. Here l=10 and α=9 are used for NEEDLE and BINEEDLE.

$$\text{NEEDLE}=\begin{Bmatrix}1+\alpha & \text{all ones string}\\ 1 & \text{otherwise}\end{Bmatrix}$$

$$\text{BINEEDLE}=\begin{Bmatrix}1+\alpha & \text{all ones string}\\ 1 & \text{otherwise}\\ 1+\alpha & \text{all zeros string}\end{Bmatrix} \quad (7)$$

$$\text{ONEMAX}=\text{number of ones in string}$$

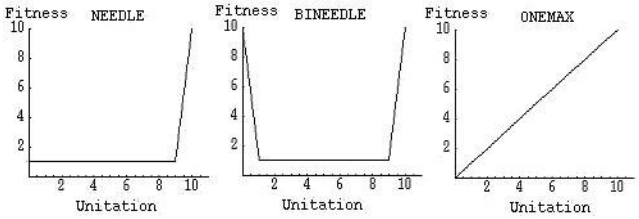

Figure 2. NEEDLE, BINEEDLE and ONEMAX fitness functions.

3.2 Deceptive Functions

Trap functions are piecewise linear functions that divide the search space into two Hamming space basins [13]. Each basin has an optimal point, one of which is the global optimum. In Deb and Goldberg [13], a set of conditions for calling a fitness function 'fully deceptive' is given. A fully deceptive function, referred to here as DECTRAP, from [13] is detailed in Equation 8.

$$f(x)=10*\begin{Bmatrix}1 & \text{if } u(x)=l\\ 1-\dfrac{1+u(x)}{l} & \text{otherwise}\end{Bmatrix} \quad (8)$$

Figure 3 illustrates DECTRAP. The trap function is at fitness of 9 for the all zeros string, and is fitness 10 for the all ones string. The all zeros basin of attraction takes up the majority of the function space.

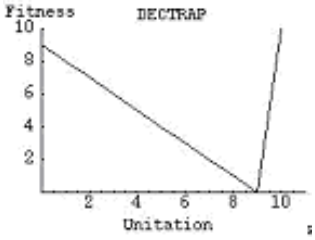

Figure 3. Fully deceptive trap functions DECTRAP.

Figure 4 illustrates a trap function containing two traps, referred to as 2TRAP. This trap was formed in an ad-hoc manner to build a landscape with a large sub-optimal basin at the center of distribution of the unitation classes.

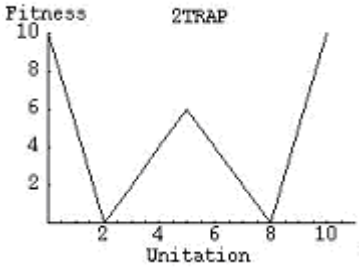

Figure 4. Double trap function 2TRAP.

Figure 5 and Equation 9 show a deceptive double trap function, or DEC2TRAP. This function is modeled after the fully deceptive function given in [13].

$$f(x) = 10 * \begin{cases} 1 & \text{if } u(x) = l/2 \\ 1 - \dfrac{1+u(x)}{l/2} & \text{if } u(x) < l/2 \\ \dfrac{u(x) - d/2 - 1}{l/2} & \text{if } u(x) > l/2 \end{cases} \quad (9)$$

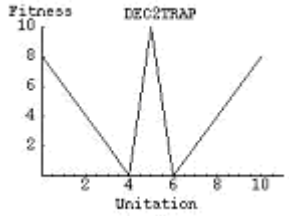

Figure 5. Deceptive double trap function DEC2TRAP.

4. Royal Road and LongPath

Mitchell et al. [14] crafted a fitness function called the 'Royal Road' which was intended to highlight the building block nature of the GA. The Royal Road is a collection of bit patterns that assigns progressively higher fitness to bit strings that build up long sequences of shorter bit patterns. A modest example for 6-bit strings is given in Equation 10. This paper uses a common reference implementation with 16 blocks (each of size 4 bits), a gap-size of 3bits and a 2-bit reward threshold. This forms a genome of 128 bits. See the authors code [15] for more details.

{11****, **11**, ****11}	fitness = 2
{1111**, 11**11, **1111}	fitness = 4
{111111}	fitness = 8

(**10**)

Horn et al. designed a fitness function called LongPath. This function is crafted by choosing a desired path length $k < 2^l$. Next a path from a starting bit-string to an ending bit-string is constructed such that it takes k one-bit mutations to 'follow' the path. Each point on the path is one 'hamming distance' away from each neighbouring point. Equation 11 shows a simple example for a 6-bit genome. This paper uses a 9-bit path detailed in Rudolph [16]. The reader is encouraged to examine the code [15] for more details.

12-path is {000000, 000001, 000011, 000010, 000110, 000100, 001100, 001000, 011000, 010000, 110000, 100000}

$$f(x) = \begin{cases} 3*2^{(l-1)/2} - 2 - Pos(x) & \text{if } x \text{ is on the path} \\ 3*2^{(l-1)/2} - 2 + \| x \|_1 & \text{otherwise} \end{cases} \quad (11)$$

Where *Pos(x)* is the number of the step in the path the genome is on. 000110 is the 4th step in a zero based count for the path above.

5. Algorithms and Experiments

Thierens [7] applied his two dynamic adaptive schemes as well as fixed rate and one deterministic scheme to the ONEMAX (or Counting Ones) problem. He used the well-known (1+1) EA strategy [6] for the fixed mutation rate and deterministic schemes and the (1+3) EA for his adaptive schemes. Different variants made the comparison more difficult. Algorithm 1 gives the (1+3) EA.

Algorithm 1: The (1+3) EA[1]

1. Choose $p(n) \in (0,1/2]$
2. Choose $x \in \{0,1\}^d$ uniformly at random.
3. Create three children $\{a,b,c\}$ by flipping each bit of x independently with $p(n)$.
4. Select $x := max(x, a, b, c)$
5. Update $p(n)$ according to some scheme *Optional*
6. Continue at line 3

Algorithm 1 will be applied to the five unitation functions given above with each of the seven mutation rate schedules/schemes also given above. One each of the Royal Road and LongPath functions are also used for the seven schedules. This totals 49 experiments. Each experiment will have 25 trials run. Note that 20-bit versions of the unitation functions are used. The basic shape of each unitation fitness function is the same as presented before.

For Rechenberg's rule, λ=1.1 is used. Constant Gain settings are λ=1.1 and ω=1.5. The Declining Adaptive method used λ=2.0, ω=2.0 and γ=0.95. For the Shi fuzzy rule set, the fuzzy assignments of mutation rate are: *high*=4/*d*, *med*=2/*d*, *low*=1/*d*.

6. Experimental Results

Table 1 shows the results for the ONEMAX function. For each adaptive method there is an average and standard deviation for the number of fitness function evaluations

[1] The (1+1) EA produces only 1 child in step 3. Step 4 chooses the maximum fitness of the parent and child.

performed. The 'Failed Runs' column gives a count of the number of GA runs that failed to find the optimal solution in 1000 generations.

All performed comparatively with the exception of Bäck-Schütz, the clear loser. Rechenberg had the best performance with a low average and a tight standard deviation. The rest of the methods are generally grouped together within the standard deviation of one another, meaning that they are statistically equivalent methods.

The NEEDLE and DECTRAP results are in Table 2. No run of the GA resulted in any scheme finding the optimal points of either fitness landscape. The NEEDLE function is hard and likely needs a very high number of mutations to get enough coverage of the 2^l fitness landscape to find the 'needle'. All of the algorithms were able to ascend to the local maxima of DECTRAP with performance similar to the ONEMAX results. This should also be not surprising since the DECTRAP function's landscape is a near clone of ONEMAX.

Mutation scheme	Avg num of fitness evals	Std dev num of fitness evals	Failed runs
Static rate	298	159	11
Droste	181	87	-
Bäck-.Schütz	815	275	1
Constant Gain	259	223	-
Declining	142	81	-
Rechenberg	99	22	-
Shi	158	83	-

Table 1. ONEMAX results

Mutation scheme	Failed runs
Static rate	25
Droste	25
Bäck-Schütz	25
Constant Gain	25
Declining	25
Rechenberg	25
Shi	25

Table 2. NEEDLE and DECTRAP results

The DEC2TRAP results are given in Table 3. Again the various methods had much difficulty, only the Droste and Bäck-Schütz schemes were successful at frequently finding the global optima, and finding it quickly. The other methods were total failures. This result is surprising since the DEC2TRAP function does contain a reasonably sized fitness basin centered around the optimal point. It's likely that these methods would be more successful on 50 and 100 bit versions of DEC2TRAP. The optimal fitness basins in those cases would be quite large.

Mutation scheme	Avg num of fitness evals	Std dev num of fitness evals	Failed runs
Static rate	-	-	25
Droste	33	37	6
Bäck-.Schütz	42	34	-
Constant Gain	-	-	25
Declining	NS	NS	21
Rechenberg	-	-	25
Shi	-	-	25

Table 3. DEC2TRAP results

In comparison to DEC2TRAP there is a performance reversal with 2TRAP. See Table 4 for details. The successful dynamic schemes that worked quickly on DEC2TRAP were total failures on 2TRAP. The reversal continues to an extent with the adaptive methods. These methods found the optimal points at least 10 times each and required a relatively few number of fitness evaluations. The fuzzy method did escape total failure on DEC2TRAP by finding the optimal in 5 of the 25 runs.

Mutation scheme	Avg num of fitness evals	Std dev num of fitness evals	Failed runs
Static rate	210	65	22
Droste	-	-	25
Bäck-Schütz	-	-	25
Constant Gain	56	26	14
Declining	73	59	10
Rechenberg	41	20	15
Shi	56	9	20

Table 4. 2TRAP results

Table 5 shows the results for the 128-bit Royal Road function. Algorithm 1 was run for a maximum of 20,000 generations. All of the adaptive methods except Rechenberg performed well on this function with the Bäck-Schütz scheme the clear winner, as it has the lowest average and the tightest standard deviation. The failure of Rechenberg is so far unexplained; λ=1.1 and λ=2.0 were tried with the same failure result.

Mutation scheme	Avg num of fitness evals	Std dev num of fitness evals	Failed runs
Static rate	-	-	25
Droste	28,312	12,413	6
Bäck-Schütz	18,615	5,251	-
Constant Gain	25,678	12,865	1
Declining	27,548	10,036	-
Rechenberg	-	-	25
Shi	20,340	9,232	-

Table 5. Royal Road results

Table 6 gives the results for the LongPath function. All adaptive schemes fared well on this fairly easy function. The Thierens' Declining rule won out with the lowest average and tight standard deviation.

Mutation scheme	Avg num of fitness evals	Std dev num of fitness evals	Failed runs
Static rate	471	56	-
Droste	157	104	-
Bäck-Schütz	232	165	-
Constant Gain	235	133	-
Declining	129	83	-
Rechenberg	225	134	-
Shi	158	103	-

Table 6. LongPath results

In both the Royal Road and LongPath functions, the static mutation rate was well outperformed by the adaptive schemes.

7. Conclusion

There are many methods known in the literature for adapting GA parameters. It is also common to see these methods perform well in those papers. However, when a systematic experiment with a variety of methods is applied to a set of fitness functions displaying basic characteristics, the methods show very mixed results.

It is worth noting that combinations of the fitness functions above can be used to construct a wide variety of complex landscapes. The results above can be used as a guide to how a particular adaptive scheme might perform in a certain situation. For instance, if a more complex fitness landscape contains 'hills' followed by a flat 'plateau', an adaptive GA is likely to climb the hill well then get stuck in the plateau.

Of course, this problem exists in non parameter-adaptive GAs. However, simply adding adaptive schemes and heuristics does not necessarily cure the GA of the types of problems commonly seen in the analysis of a typical run of the GA. The *No Free Lunch* theorem of Wolpert and MacReady [17] is a general proof for GAs that one can not claim that a GA with a fancy operators is provably better than any other GA. Reeves and Rowe [18] detail the No Free Lunch theorem as well as the debate concerning the Building Block Hypothesis.

References:

[1] M. Mitchell, An Introduction to Genetic Algorithms (Complex Adaptive Systems Series). Cambridge, MA: MIT Press, 1996.

[2] J. Horn, D. E. Goldberg, and K. Deb. Long path problems. In Proc. of PPSN III (Parallel Problem Solving from Nature), LNCS 866, pages 149--158, Springer, Berlin. 1994.

[3] R Hinterding, Z. Michalewicz, and A.E. Eiben, "Adaption in evolutionary computation: a survey," in Proc. of 1997 IEEE Conf. on Evolutionary Computation (ICEC`97), 1997, pp. 65-69.

[4] H.-G. Beyer. *The Theory of Evolution Strategies*. Natural Computing Series. Springer, Heidelberg, 2001.

[5] Thomas Bäck and M. Schütz. Intelligent mutation rate control in canonical genetic algorithms. Proceedings of the 9th International Symposium, ISMIS 96, pages 158--167, June 1996. Springer-Verlag, Berlin (Germany).

[6] S. Droste, T. Jansen, I. Wegener, Dynamic parameter control in simple evolutionary algorithms, Proc. Foundations of Genetic Algorithms, Vol. 6, 2001, pp. 275-294.

[7] Thierens, D. (2002). Adaptive mutation rate control schemes in genetic algorithms.. In (Ed.), *Proceedings of the 2002 IEEE World Congress on Computational Intelligence: Congress on Evolutionary Computation* (pp. 980-985). IEEE Press.

[8] Y. Shi, RC Eberhart, and Y. Chen. Implementation of Evolutionary Fuzzy Systems. IEEE Trans. Fuzzy Systems, 7(2):109-119, 1999 .

[9] T.J. Ross, Fuzzy Logic with Engineering Applications. McGraw-Hill, New York. 1995.

[10] Rowe, J.: Population fixed-points for functions of unitation. In Foundations of Genetic Algorithms. Vol. 5. Colin Reeves, and Wolfgang Banzhaf (eds.). Morgan Kaufmann, (1998) 69-84.

[11] Wright A.H., Rowe J.E., Neil, J.R.: Analysis of the Simple Genetic Algorithm on the Single-peak and Double-peak Landscapes, Proc. of the 2002 Congress on Evolutionary Computation, Fogel et al. Editors, IEEE Press (1999), 214-219.

[12] Richter J.N, Paxton, J. Wright, A. EA Models and Population Fixed Points Versus Mutation for Functions of Unitation. To appear in Proceedings of GECCO 2005. Washington, D.C. July 2005.

[13] Deb, K., Goldberg, D.E.: Analyzing Deception in Trap Functions. In *Foundations of Genetic Algorithms, Vol. 2*. D Whitley, editor. Morgan Kaufmann, (1992): 93-108.

[14] Mitchell, M., Forrest, S., and Holland, J. H. (1992). The royal road for genetic algorithms: Fitness landscapes and GA performance. In Proceedings of the First European Conference on Artificial Life. Cambridge, MA: MIT Press/Bradford Books.

[15] http://www.cs.montana.edu/~richter/iasted_2005_code.php

[16] G. Rudolph. Convergence Properties of Evolutionary Algorithms. Verlag Dr. Kovac, Hamburg, 1997.

[17] D. H. Wolpert and W. G. MacReady. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, April 1996.

[18] CR Reeves, JE Rowe, Genetic Algorithms - Principles and Perspectives.. Kluwer Academic, Boston MA, 2003