"Alexandru Ioan Cuza" University of Iași
Department of Computer science

High probability mutation in Genetic Algorithms

PhD. dissertation summary

PhD. candidate: *Nicolae-Eugen Croitoru*

Scientific coordinator: *Prof. Henri Luchian, PhD.*

December 2016

Contents

Acknowledgements

I would like to thank my supervisor, Prof. Henri Luchian, PhD., and everyone who participated in the review of my work.

I owe special thanks to an anonymous reviewer who, during the SYNASC 2014 conference review process, suggested investigating Error Thresholds in the context of High-Probability Mutation.

Published Work

Work underlying parts of this dissertation has been previously published:

- N. E. Croitoru. High-probability mutation in basic genetic algorithms. *Proceedings of the 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), IEEE*, pages 301–305, September 2014

- N. E. Croitoru. High probability mutation and error thresholds in genetic algorithms. *Proceedings of the 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), IEEE*, pages 271–276, September 2015

- N. E. Croitoru. Lowering evolved artificial neural network overfitting through high-probability mutation. *18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, September 2016

1 Introduction

Genetic Algorithms[11] are a class of meta-heuristic search algorithms, inspired by the biological evolution by natural selection. They manage good search and optimisation performance on many types of problems, despite modelling the complex process of biological evolution in a relatively simple way.

Genetic Algorithms are a population-based method (as opposite to a trajectory-based one): they maintain a population of genomes (also named genotypes, individuals or, in haploid Genetic Algorithms, chromosomes - all these names are used interchangeably in this work). In the simplest case, all genomes have equal lengths, and their basic building blocks, nucleotide-analogues, are simple bits, 0 and 1.

A series of nature-analogous basic operators are used to simulate evolution on that population. Each genome expresses its genes, is evaluated according to that phenotype expression, and given a fitness. Individuals are treated as replicators, and are subjected to differential selection according to their relative fitness. The selection operator directs evolution, aiming it towards maximising the fitness. By choosing an appropriate phenotype expression function and an appropriate fitness measure, the Genetic Algorithm's search can be directed towards solving a problem.

Recombination (or cross-over) emulates simple sexual reproduction or bacterial gene transfer[13]; by allowing already-existing genes to permute from one genome configuration to

another, favourable recombinations appear. Cross-over attempts to improve fitness by exploiting the information already available in the gene-pool.[1]

Mutation, based on the biological phenomenon of replication error, induces random changes to genes; new genes, and thus, new traits can appear through this random process. The mutation operator, by randomly introducing new information into the population, explores new possible genome configurations. [2]

However, evolution takes time (emulated as the successive application of the different operators on the population). Simple attempts at satisfying the objective function - at obtaining a good fitness - are evolved, then replaced with better genomes - better potential solutions - over and over again. The combined information gained by evolution is stored within all the genomes in the population. New individuals and new generations rely on the old ones for their evolutionary improvement and optimisation.

A danger when using the mutation operator is setting the induced error rate too high; a high-enough mutation randomly mutates genes beyond the ability of the evolutionary process to correct or compensate for. Instead of evolving a population of one or several groups of mostly-similar individuals (or quasispecies[10]), the population degenerates in a group of divergent, dissimilar genomes. As genes are no longer kept mostly intact from parents to descendants, evolution cannot accumulate information, and the whole evolutionary process is disrupted. In biological evolution, this most often results in

[1] In this work, cross-over is often shortened to *cx*.

[2] Where appropriate, mutation is replaced with the letter m. For instance, mutation probability can appear as p_m.

immediate cellular death; in Genetic Algorithms it results in Random Search[21].

While, in the context of the No Free Lunch Theorem[28], Random Search Algorithms have applications where they outperform Genetic Algorithms, hybridising the two by increasing the mutation rate is generally avoided.

However, in the context of Genetic Algorithms, a high mutation rate and evolution disruption are not synonymous. If the probability of mutating each gene in the population is very high (in the proximity of 1), due to the synchronised nature of mutation in a genetic algorithm (i.e. all operator actions happen at the same time: the mutations in a generation happen at once, not at random times) and the simple binary encoding of the representation, evolution will not be disrupted. Instead, if mutation bit-flips most genes in one generation, the vast majority of those genes will be bit-flipped back. By observing gene change every two generations, the effective changes brought by high-probability mutation can be shown to be relatively low.

This is the core argument of this dissertation: that high-probability mutation is a variant type of mutation in the context of Genetic Algorithms; it is not a Random Search Algorithm hybridisation.

Beyond theoretical arguments, low- and high-probability mutation are compared to each other, and contrasted with high-entropy, evolution-scrambling mutation. In addition to statistical measurements of Genetic Algorithm performance, population evolution is analysed in-depth: by using Consensus Sequence Plots[17], the thresholds between functioning and disrupted evolution are found. Such Error Thresholds under which evolution can happen undisrupted are found, even for high-probability mutation.

In addition to well-known optimisation benchmark problems, the characteristics of high-probability mutation are put to the test using real-world data. It is shown to lower overfit in a Neuroevolution algorithm attempting to predict the future state of an Internet-based social network.

1.1 Aim

The purpose of this work is to better understand high-probability mutation, and show it as worthwhile tool in the Genetic Algorithm toolbox. This understanding allows for stating a basic rule-of-thumb, to advise when high-probability mutation might be useful, and when it will likely be detrimental to a Genetic Algorithm.

Another objective is to show high-probability mutation in Genetic Algorithms is different from a Random Search Algorithm. This is accomplished in three ways:

- theoretically, by statistically computing event chains occurring during mutation.
- indirectly, by showing that low- and high-probability mutation have similar performances to each other, and different performance to high-entropy, random-search mutation.
- directly, by visually and statistically showing the formation of stable evolved structures within the population for both low- and high-probability mutation, where for high-entropy mutation they do not.

Additionally, it is desired for the current study to be as relevant as possible. Since the majority of Genetic Algorithms used at the time of writing are incremental or evolved derivatives of John Holland's initial method[11][3], the Genetic Algorithm used here is a close derivative.

It is hoped that results relevant to this algorithm will apply to many other GA variants and hybrids used or researched. Through a "phylogenetic metaphor", the root of the GA variant tree is expected to be most representative to the class, and, overall, the single variant most similar to all of them.

It is due to this desire for relevance that most tests are performed on well-known and widely-used benchmark function instances; at the same time, real-world data is used to test the method, to show its practical applicability.

[3] As shown by the comparatively high number of citations that dissertation received

2 Initial Study

The focus on High-probability Mutation was initially motivated by the results of a preliminary study.

The goal of the study was to better choose and fine-tune probabilities for the Mutation and Crossover operators. A better choice for operator probabilities improves the quality of the optima found by the Genetic Algorithm - and this optimum quality is the measure for the success for a certain operator probability.

The design of the experimental phase is simple: Mutation and Crossover probabilities are varied from 0 to 1 in 0.01 increments - each percent being tested. For each Mutation probability and Crossover probability combination, a Genetic Algorithm instance is run to sample it 32 times, with the produced optima being recorded.

The test problems used are Rosenbrock's Function[23] and the Six-Hump Camel Back Function[8]. For both functions, the optimum is a minimum.

The number of individual GA runs is $2 \times 101 \times 101 \times 32$: the number of function instances times the number of p_m values chosen, times the number of p_{cx} values, times the sample size for each parameter set.

2.1 Results

The results are presented in bi-dimensional box-and-whiskers plots. For the sake of brevity, only mutation-related results are shown.

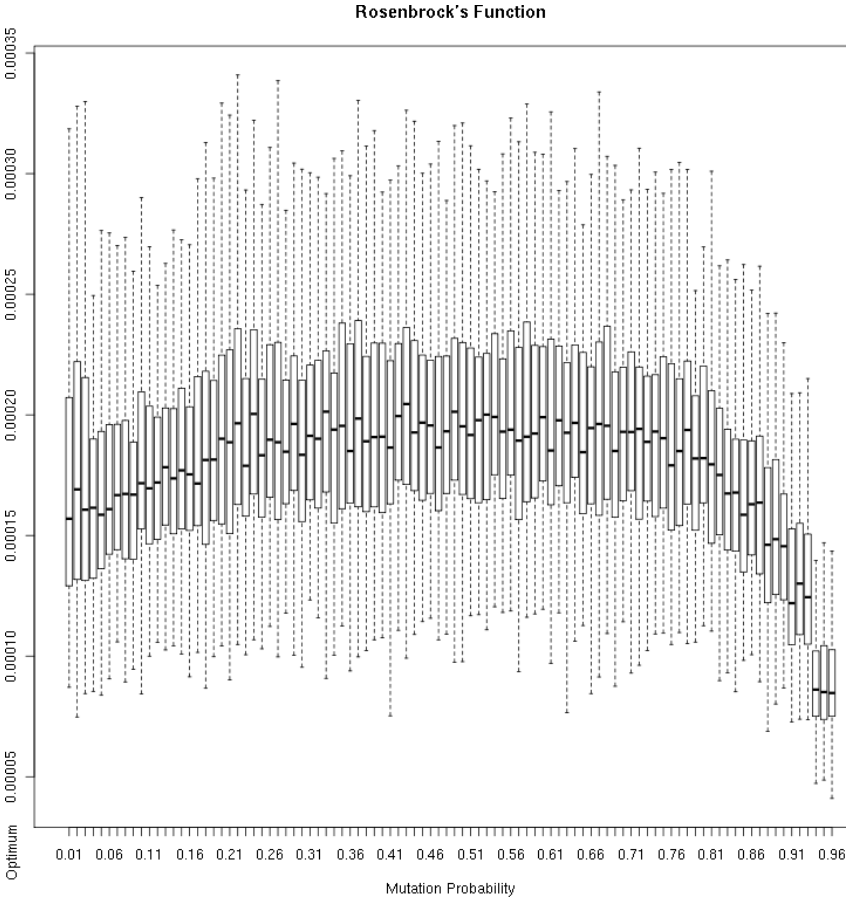

Figure 2.1: Rosenbrock's Function: p_m vs. optimum quality, boxplot

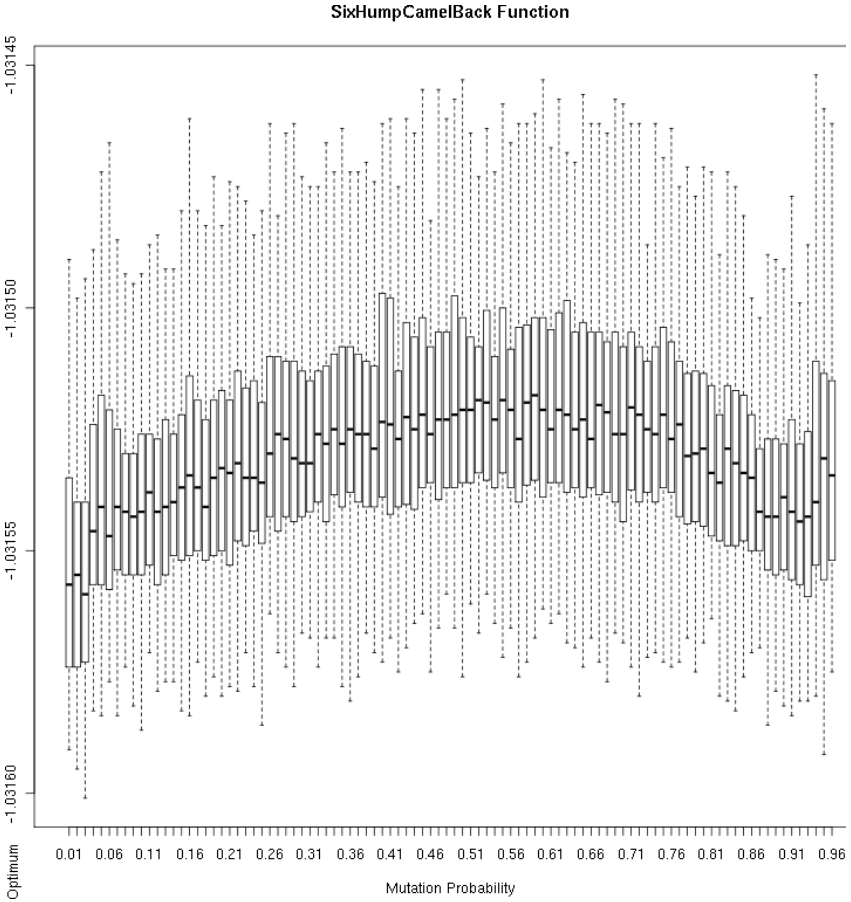

Figure 2.2: Six-Hump Camelback Function: p_m vs. optimum quality, boxplot

The plots (Fig. 2.1 and 2.2) show a decrease in optimum quality as p_m is increased away from 0. However, this deterioration plateaus for p_m values around $(0.5, 0.6)$. Afterwards, the optimum quality increases as p_m increases.

It is this good optimum quality at high p_m values that moti-

vated the later studies. Specifically, in the case of Rosenbrock's Function, the optima found for $p_m \in \{0.94, 0.95, 0.96\}$ are the best among all tested mutation probability values[1]. In the case of the Six-Hump Camelback Function, even though high-probability mutation performs worse than low-probability mutation, it still delivers good performance; this was interpreted as another reason to investigate High-Probability Mutation.

2.2 Interpretation

Mutation probability values show the best results in close proximity to 0 and 1, while the worst optima values are found in a wide range around $p_m = 0.5$.

As p_m increases to 0.5 and beyond, genomes are scrambled until evolution is disrupted (at least partially), and Random Search appears. At $p_m = 0.5$, each gene has a 0.5 chance to be 0 and a 0.5 chance to be 1 (immediately after mutation). The value at a certain locus no longer depends on previous values - on the steady evolution during past generations, within the Genetic Algorithm - but on the random chance at the current mutation step. Crossover merely recombines uniformly random genes in an uniformly random way; selection does skew the population distribution towards better fitness, but there can be little persistence, since the population is uniformly scrambled at the next mutation step.

However, as p_m increases even further, genes have a greater chance to be changed back to where they started. Information across generations is again preserved, and evolution can take

[1] A visual indication can be found in Fig. 2.1, where samples with $p_m \in \{0.94, 0.95, 0.96\}$ show interquartile Q1-Q3 separation from all other samples

place. For e.g. $p_m = 0.9$, the value of at a certain locus will be anti-correlated with the previous value at that position; after two generations, the anti-correlation will result, for the binary encoding, in a likely return to the initial value. Since these changes are synchronised across the whole population, all the bits will be flipped in step, and the genetic information will 'reappear' in its standard encoding every other generation.

High-Probability Mutation is not a Random Search Algorithm, but a basic way for the Genetic Algorithm to work with two encodings - a base encoding and its binary negation.

3 High-probability Mutation

3.1 Description

In the context of Genetic Algorithms, high-probability mutation is defined by two competing measurements:

- the basic mutation probability - the simple average rate of applying mutation to any locus - is high;
- the effective mutation probability - the cumulated rate of gene change brought upon by mutation in the course of the algorithm - is low.

For binary-encoded Genetic Algorithms, the effective mutation rate can be measured every two generations. This is due to the fact that any locus has only two alleles, and because basic mutation simply flips between the two possible values of a locus. After 2 generations, high-probability mutation will flip most bits back.

The effective mutation rate after 2 generations for binary-encoded genomes can be calculated by considering the outcomes of each possible mutation. The reasoning below works for constant mutation rates, and can be extended to cover small variations in mutation rate (genome-dependent, locus-dependent or time-dependent).

Assuming a random locus, there are two outcomes as the mutation operator is applied in a generation: either the bit is flipped, or it is not. Let us name these two events f, for flipped, and $\overline{f}$ for not flipped. Obviously, they are statistically complementary, the chance of f occurring is $p_f = 1 - p_{\overline{f}}$.

During the second mutation pass (in the following generation), the same locus has the same chances of being flipped or not. For the information contained at that locus, there are two possibilities: it has either retained the value it has started two generations with, or it has not.

The probability of a bit-flip is equal to the mutation probability, $p_f = p_m$ (in the constant mutation probability case) . However, the rate of effective mutation, p_{em}, represents the rate of information change over the course of two generations. The relation between the p_m and p_{em} can be seen by following the chain of events that lead to each outcome.

- For a locus to have changed its value, there are two possible scenarios:
  - It had been changed by the first mutation, and was unchanged by the second.
  - It was changed by the second mutation pass, but remained unchanged during the first.
- For a locus to have preserved its value, there are two possible scenarios:
  - It has remained unchanged during both mutation passes.
  - It has been bit-flipped by both mutation passes.

Thus, the probability for a change (over the course of two successive generations is):

$$p_{em} = (1 - p_m) \cdot p_m + p_m \cdot (1 - p_m) = \tag{3.1}$$
$$= 2 \cdot p_m \cdot (1 - p_m) \tag{3.2}$$

Its opposite, the probability for retaining the initial value, is:

$$p_{\overline{em}} = (1 - p_m)^2 + p_m^2 = \tag{3.3}$$
$$= 1 - 2 \cdot p_m + 2 \cdot p_m^2 = \tag{3.4}$$
$$= 1 - 2 \cdot p_m \cdot (1 - p_m) \tag{3.5}$$

which verifies the relationship $1 - p_{em} = p_{\overline{em}}$.

The maximum of $p_{em} = 0.5$ is reached at $p_m = 0.5$. The minima of $p_{em} = 0$ are reached for $p_m \in \{0, 1\}$.

A low effective mutation rate corresponds with good optima, and high p_{em} with bad optima - except for the very extremities of the probability interval. Effective mutation rates are equal for complementary mutation probability: $p_{em}(p_m) = p_{em}(\overline{p_m})$. The effective mutation rate over two generations for a $p_m = 0.95$ is $2 \cdot 0.95 \cdot (1 - 0.95) = 2 \cdot 0.95 \cdot 0.05 = 0.095$.

In this context, p_{em} is a better predictor for GA solution quality than p_m. Low-probability mutation explores one region of promising p_{em} values, while high-probability mutation explores the other region.

3.2 Performance Comparison

Choosing between low-probability mutation and high-probability mutation remains an open question; as Chapter 2 shows, for different function instances we obtain different performances.

Furthermore, mutation interacts with the rest of the Genetic Algorithm: hypotheses formulated only in terms of mutation need to be tested in the context of the whole GA. Even the simple p_{em} formula does not hold accurately since, after each generation, selection skews the probability distribution of genes according to fitness landscape.

In order to help choose between probabilities and understand mutation better, a wider set of function instances has been investigated: De Jong 1[6], Michalewicz's [14], Rastrigin's[22], Rosenbrock's [23], Schwefel's[25], Six-Hump Camelback [8] GA Royal Road[16] and the GA Trap Function[7].

The numerical functions have minimal optima, while the bit-block functions have maximal optima. For better identification, an upwards arrow ↑ next to the function name signifies an optimum that is a maximum, while a downwards arrow ↓ a minimum.

Teach table cell contains 3 numerical values, placed vertically. Each displays different information about the statistical sample[1] collected for each parameter set. Starting with the top value, they signify: the median of the sample, the mean of the sample and the standard deviation of the sample:

Median values appearing in **bold font face** mark the best median for that function instance. Average values so marked mean that, following a statistical test, a significant difference has been observed between that sample and all others. Furthermore, it means that sample is better (closer to the optimum).

The statistical test was the following: the samples were compared against the other, using Welch's unequal variance t-test[26] ($\alpha = 0.01$). Welch's t-test was selected because

[1]Sample size is constant at 1024

the standard deviations among samples tend to differ. Afterwards, the resulting p-values were adjusted using the Holm-Bonferroni method[12]. The Holm method was chosen because it is more conservative[2], lowering the probability of false-positives (at the cost of increasing false-negative chance).

Function name	$p_{mutation}$			
	0.0005	0.001	0.01	0.95
De Jong 1 $^{\downarrow}$	6.51e-05	6.81e-05	**6.49e-05**	6.31e-04
	0.00010	0.00010	0.00010	0.00009
	0.00012	0.00012	0.00011	0.00097
Michalewicz$^{\downarrow}$	**-1.80129**	-1.80128	-1.80128	-1.80062
	-1.80127	-1.80127	-1.80126	-1.80007
	3.48e-05	4.17e-05	4.44e-05	0.00169
Rastrigin$^{\downarrow}$	0.00257	**0.00243**	0.00258	0.05708
	0.00634	0.00575	0.00622	0.10497
	0.01130	0.01023	0.01253	0.13632
Rosenbrock$^{\downarrow}$	0.00027	0.00026	0.00027	**9.91e-05**
	0.00396	0.00538	0.00408	**0.00006**
	0.03789	0.04692	0.03394	0.00843
Schwefel$^{\downarrow}$	**-837.961**	-837.961	-837.961	-837.515
	-831.379	-831.199	-829.648	**-836.258**
	25.49138	25.85371	28.13673	10.56529
SixHump$^{\downarrow}$	-1.03155	-1.03156	**-1.03156**	-1.03148
	-1.03149	-1.03150	-1.031509	-1.03137
	0.00015	0.00016	0.00014	0.00031

Table 3.1: Low-probability vs. high-probability mutation. Numerical functions, $p_{cx} = 0.2$

Function name	$p_{cross-over}$ 0.2		0.5	
	$p_{mutation}$ 0.01	0.95	0.01	0.95
Royal Road$^{\uparrow}$	40	40	40	**48**
	42.5	**47**	41.625	**48.125**
	7.25	11.96	4.31	10.522
Trap$^{\uparrow}$	53	**56**	53	**56**
	52.640	**56.140**	52.656	**56.421**
	1.2	1.48	1.15	1.29

Table 3.2: Low-probability vs. high-probability mutation. Bit-block functions.

Low-probability mutation and high-probability mutation show different relative performances on different function instances. It is interesting to note that low-probability mutation performs much better on function landscapes where the optimum is surrounded by relatively steep walls, e.g. Rastrigin's Function. The simplest example to illustrate this is De Jong's Function 1, which consists of nothing but a single optimum surrounded by steep walls (it is continuous, strictly convex and multivariate uni-modal). On these function landscapes, we find the largest relative difference between low- and high-probability mutation (approximately by a factor of 10 in median, mean and standard deviation, although these differences cover just a small part of the function's ranges).

By contrast, when a function landscape consists of relatively flat plateaus surrounding optima, high-probability mutation performs better (such as Rosenbrock's Function). When the two types of features are mixed, the gap between low- and high-probability mutation shrinks.

I interpret these results as an increase in exploration on the part of high-probability mutation: climbing down the steep walls of a spherical landscape requires little exploration, but constant exploitation - the shortest way to the optimum is that given by a hill-climbing algorithm following the greatest height/fitness difference. Attempting to explore another point, on this type of landscape, is very likely to mean a step back, and is a (heuristically) suboptimal choice. In the case of Rastrigin's Function, exploitation is the best strategy after exploration finds the global optimum's attraction basin.

Navigating plateaus usually means escaping them - exploitation cannot (strictly) improve the solution on a flat plateaus - or 'jumping ahead' of exploitation. This is best illustrated by Rosenbrock's Valley Function, where the optimum is at the bottom of a mostly-flat banana-shaped valley. If the exploration tendency of the algorithm is insufficient, the GA will prematurely converge on the plateau. I believe that high-probability mutation works, in this context, by forcing the GA to take a step away from where it has converged. By flipping the vast majority of bits, a dual, Boolean-negated representation is created; if the majority of chromosomes are in close proximity on the landscape, the their dual representations are likely to be farther apart (subject to landscape properties, but true on studied function instances). By undergoing recombination and selection, the genomes are moved on the landscape; upon undergoing high-probability mutation again, they are returned to approximately similar positions - but not the same. This way, solution trajectories (for similar parts of the population) no longer need to travel along the function's surface - they can also use approximate dual landscapes to jump.

When increased exploration and population variety are not desired, the dual representation still exists and takes up com-

putational resources, and induces a drift, however small, in the 'primary' population. This disrupts exploitation, and explains the lower performance of high-probability mutation on function landscapes where global optima have steeper attraction basins.

On the two bit-block function instances tested, high-probability mutation outperforms low-probability mutation. Considering that Genetic Algorithms are outperformed by Random Search on the GA Royal Road Function[16], the increase in exploration brought by high-probability mutation can explain the increase performance of the GA.

The same is true in the case of the GA Trap Function[7] - the trap represents a point of premature convergence, and high-probability mutation occasionally allows the GA to escape from it. It is interesting to see that high-probability mutation dual encodings do not overpower the primary encoding. The local optimum for the GA Trap Function is a bitstring of 1's, while the global optimum is a bitstring of 0's. Ideally, the Genetic Algorithm would arrive at the local optimum, then bit-flip into the global optimum; since this does not happen in the general case, primary and dual encodings seem to be in a tug-of-war, the spread of a gene being conditioned by the fitness of both its encodings.

Using high-probability mutation does not allow the GA to outperform Random Search Algorithms on the above bit-block functions - it merely improves on the baseline performance of Simple Genetic Algorithms with low-probability mutation.

4 Error Thresholds and High-probability Mutation

4.1 Description

In biological evolution, the Critical Mutation Rate represents the maximal mutation rate for which a gene-encoding molecule can preserve its information across generations[9]. If the mean mutation rate is kept below the critical rate, the molecule will be able to reproduce into similar descendants, creating a similarity-based cluster (or quasispecies[10], [24]), where the initial genetic information will be kept largely intact. If the mutation rate exceeds the Critical Mutation Rate, it will induce too many errors upon replication: the information contained in the group of descendants will no longer converge, on average, on the initial parent, but will start diverging rapidly.

Since, in biological evolution, mutation occurs as random errors mainly encountered in the process of information-encoding molecule (e.g DNA, RNA) multiplication, and since the transition layer between information convergence and divergence is thin, Critical Mutation Rate is also named Error Threshold: the thresholds beyond which the error rate disrupts evolution.

Genetic Algorithms are inspired by biological evolution, and

Error Thresholds have been introduced[20] to them in an effort to augment theoretical analysis and performance predictions for GAs[18].

Error Thresholds can be used to determine optimal mutation rates[18]: by increasing mutation up to the Error Threshold (but strictly below it), the exploratory tendencies of the GA can be maximised, while still maintaining long-term evolved structures within the population.

Solid lines and squares correspond to non-destructive cross-over, while dashed lines and triangles to destructive cross-over. The lines have been interpolated, while the vertices of the plot represent experimental data.

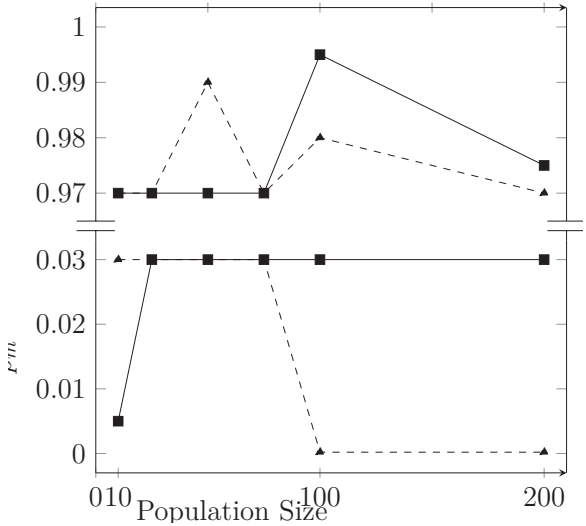

Figure 4.1: Error Thresholds: Rastrigin's Function

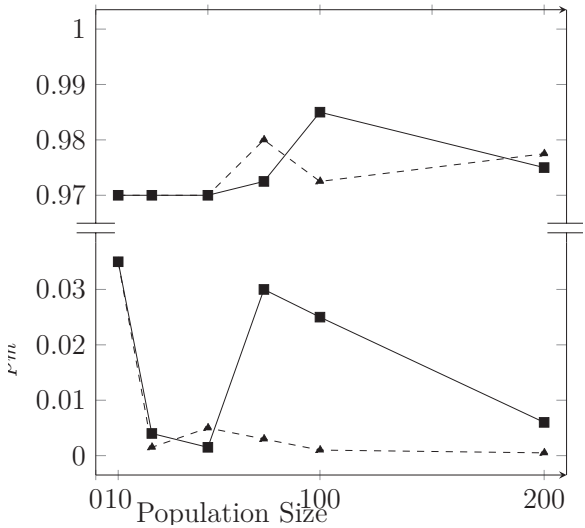

Figure 4.2: Error Thresholds: GA Trap Function

4.2 Conclusions

By allowing stable Consensus Sequences to exist, high-probability mutation differentiates itself from simple random search happening within a Genetic Algorithm. Even if the majority of bits in the population experience binary negation every generation, the Consensus Sequence returns to its previous form every other generation. The large-scales structures that are formed by the process of evolution persist and progress under high-probability mutation.

Although not perfectly mirrored, Error Thresholds form at opposites ends of the mutation probability range, for low- and high-probability mutation.

5 Lowering overfit in Neuroevolution

5.1 Description

There are many problems approachable by way of Genetic Algorithms; by encoding bitstrings, the GA can search patterns; by encoding real-valued parameters (as fixed-point binary numbers), numerical optimisation tasks can be undertaken.

By encoding the synaptic weights of Artificial Neural Networks, Genetic Algorithms can evolve ANNs for specific tasks, in a flexible and adaptive way.

5.1.1 Problem description

The problem consists in predicting the future state of a social network, from records of its past states. The studied network is Flickr[1], a social network oriented towards uploading, viewing and sharing photographs.

Through its public API, a series of large-scale statistical measures about the network can be obtained. In this case, the data consists of 10 successive snapshots of the most popular tags on the network, taken at fixed time intervals.

The problem goal is to predict the 10-th snapshot from the

previous 9. Each snapshot contains the relative popularity percentages of the 200 most popular tags at the time.

Solving the problem involves lowering the prediction error - that error is measured, specifically, in terms of Mean Absolute Error:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |{}_{p}tag_i - {}_{r}tag_i|$$

where n is the number of tags in the whole data set, ${}_{p}tag_i$ is the predicted value for the i^{th} tag, and ${}_{r}tag_i$ is the recorded value of the i^{th} tag.

The MAE was chosen over other error computing measurements because it can be translated directly into tag assignments mistakenly predicted by the algorithm: a MAE of 0.4 means that 40 tag assignments, out of 100, have been mispredicted.

5.2 Results

Due to limits on computational resources, this experiment only investigates ANN hidden-layer sizes up to 2^8 neurons. Furthermore, while Neuroevolution runs up to hidden-layer sizes of 2^7 have associated sample sizes of 32, for 2^8, the sample size is just 4.

The shorthand terms *proposed* and *best* have the following meanings:

- Proposed: the solution proposed by the Genetic Algorithm, as the best-evaluated (on the training set) in the last generation.

- Best: the best solution visited by the Genetic Algorithm in a run, as the best-evaluated on the test data in all generations.

The experimental results do show an improvement in proposed solutions found as high-probability mutation is used (compared to low-probability mutation).

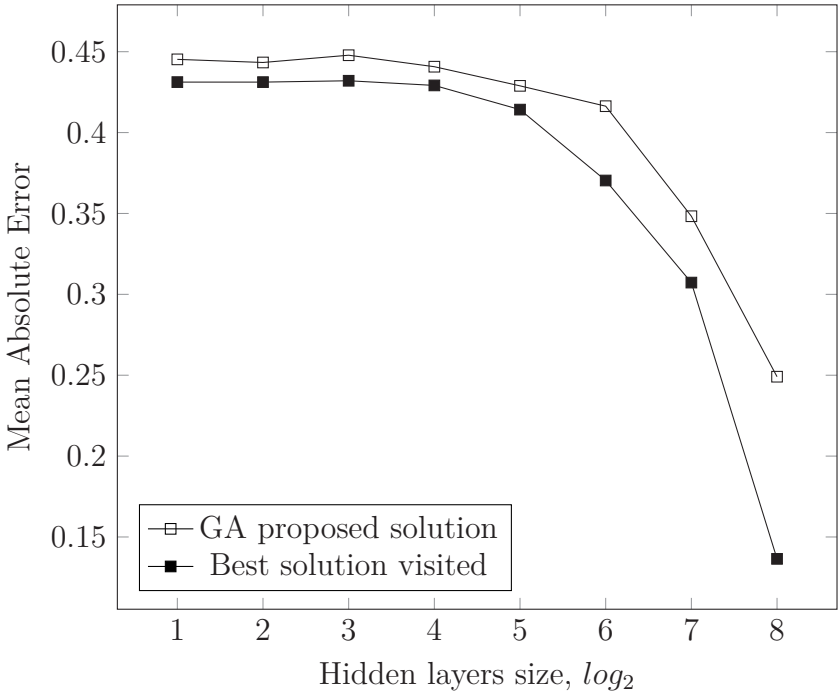

Figure 5.1: Neuroevolution $p_m = 0.01$ results

MAE decreases as ANN size increases: the data is quite complex, and it requires large ANNs, capable of approximating complex relationships. Better results are found only with ANN

hidden layer sizes of above 2^6. Results are largely unchanged between ANN sizes lower than that.

Overfitting is an issue: the best predictive models visited by the GA have MAE values below 0.15, and overfit increases, comparatively, with ANN size.

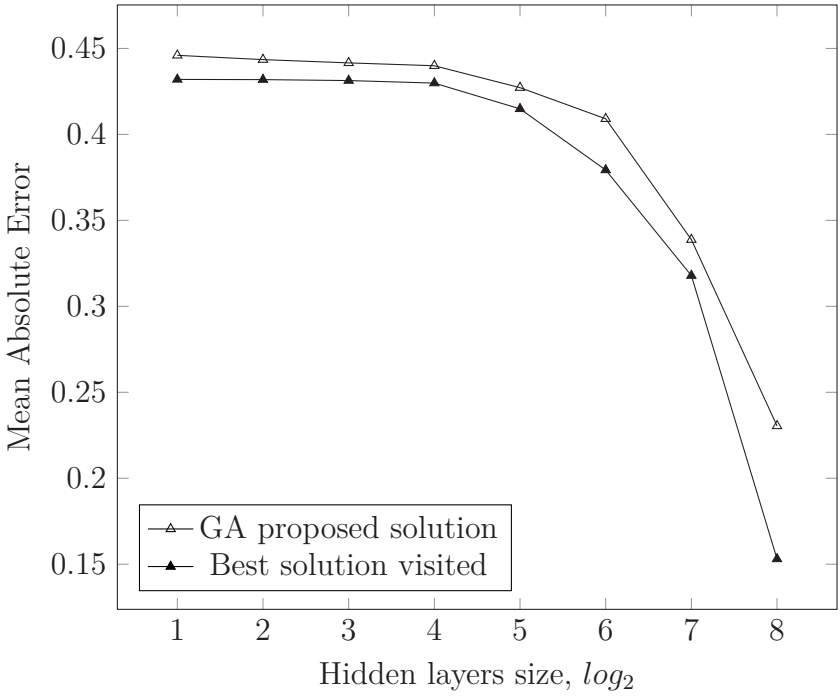

Figure 5.2: Neuroevolution $p_m = 0.95$ results

A GA using the mutation probability of 0.95 shows the same general behaviour: error improves as ANN size increases, but overfit increases. However, the distance between the Best solution visited and the GA proposed solution is reduced through

the use of high-probability mutation, compared to low-probability mutation.

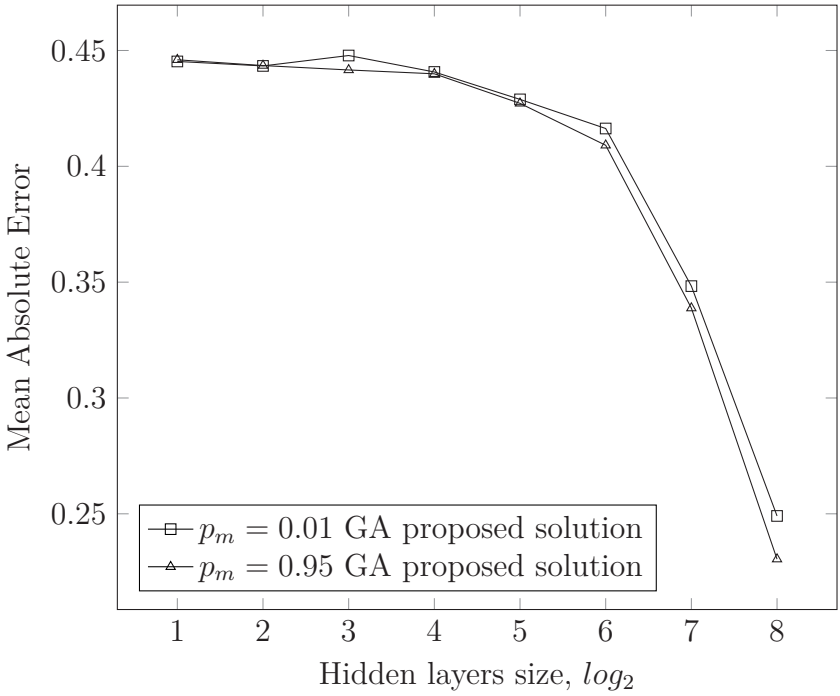

Figure 5.3: Neuroevolution: GA proposed solutions comparison

A direct comparison of the GA proposed solutions shows that a mutation probability $p_m = 0.95$ lowers the prediction error, relative to $p_m = 0.01$.

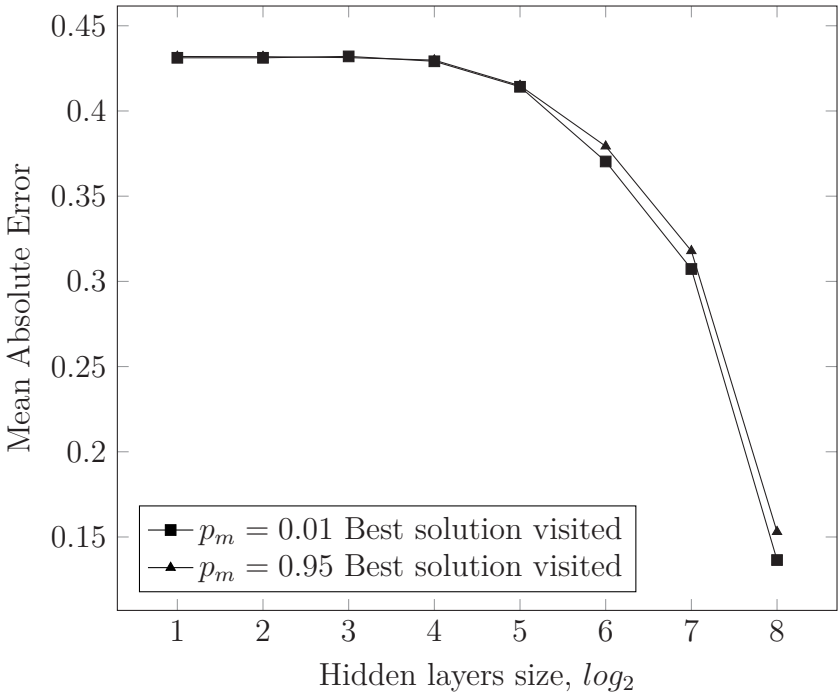

Figure 5.4: Neuroevolution: Best solutions visited comparison

The improvement of the proposed solution comes at the cost of worsening of the best solution visited.

5.3 Conclusions

The experimental results show that using high-probability mutation improves the solution the GA proposes and lowers the empirically-found overfit.

The initial assumption was that high-probability mutation, in an adaptive avoidance of plateaus and premature conver-

gence, would guide the Genetic Algorithm away from overfit. This experiment, aimed at testing that assumption, does not contradict it.

There are some limitations - high-probability mutation does not eliminate overfit, merely lowers it. Furthermore, while this overfit reduction results in solution improvements, it also worsens the quality of best visited solutions. This limits both the ability to draw on best visited solutions as a resource, and our ability to directly, empirically, measure overfit in high-probability mutation GAs.

Finally, the proposed approach does not cause the proposed solution to coincide with the best visited solution; due to this, it is a partial success, and bears improvement.

6 Conclusions

Effective mutation provides a basic measure of the entropy introduced in the population by way of the mutation operator. There are high-entropy regions, and low entropy-regions; the Genetic Algorithm behaves similar to a Random Search Algorithm hybrid in the high-entropy regions, forgoing evolution.

Evolution is possible only if the mutation operator allows enough stability for the information contained in the genomes. While the precise threshold between feasible evolution and random search is sensitive to many factors (Chap. 4), the phase transition between the two does not contradict the conclusions derived from the concept of effective mutation. High values for effective mutation correspond to random search and no stable Consensus Sequences; low values for effective mutation allow for stable evolution.

Low-probability mutation lies in one low effective mutation area; high-probability mutation lies in the other. Both permit the formation of stable Consensus Sequences, and the occurrence of evolution.

Having analysed a sample of the full range of mutation probabilities (Chap. 3), characteristics of low- and high-probability mutation subsets, and having directly determined where Error Thresholds form, an account for the whole range of probability values of the mutation operator can be given:

- For p_m values at or very close to 0, there is not enough ex-

ploration in the Genetic Algorithm. By merely exploiting the same solutions through recombination, finding global optima is unlikely, and the GA performs poorly.

- For non-0 mutation probabilities bellow the Error Threshold, the mutation operators allows for stable evolution, while the errors it introduces in genomes push it to explore new solution candidates.

- For mutation values in a wide range centred around $p_m = 0.5$, where stable Consensus Sequences cannot form, evolution does not take place. Instead, a hybrid Random Search Algorithm is created. There are two ways of identifying this range: first is by computing the entropy mutation imparts to the population, not in a generation, but across multiple ones; second, by identifying the central area on the mutation probability range of values that is delimited by Error Thresholds.

- As mutation probability increases beyond a certain value, the cumulative errors it imparts on the population start decreasing, eventually below the Error Threshold beyond which evolution can happen again. This is the area of high-probability mutation.

- Very close to or at $p_m = 1$, the rate of errors induced in the population is insufficient to drive exploration. As such, the algorithm returns to exploiting the same areas of the landscape, with generally poor results.

- Small variations in the probability of applying the mutation operator can still lead to large changes, and problem-specific particularities are expected. The p_m values used,

> while aimed at sampling the whole $[0, 1]$ range, do not and cannot cover the whole range; the assumption underlying the sampling is that very similar mutation probabilities will lead to similar Genetic Algorithm behaviour. In the context of the huge number of possible p_m values, it is likely a large number of exceptions exist.

By creating an approximate dual, binary-negation representation for each genome, very fit genomes are prevented from replicating until they fill the population. Instead, in the "dual" generations, other genomes are given the chance to reproduce. By taking this step back, high-probability mutation prevents the population from being locked into an area of the landscape - it prevents population convergence.

When navigating a plateau, due to the low fitness differences between genomes, differential selection is slow to follow promising leads: plateaus are slow to navigate. By working in the approximate dual representation, high-probability mutation offers "shortcuts", ways for the population to explore other areas of the landscape, less by the generally good fitness of the plateau.

The same features make high-probability mutation slow to converge to good optima - there is no general way to know, ahead of time, if an attraction basin leads to a local or global optimum.

Thus, a rule-of-thumb regarding the use of high-probability mutation can be formulated: If the function landscape is known or suspected to contain plateaus, or if the Genetic Algorithm struggles with premature convergence, high-probability mutation should be used.

From a practical perspective, the most important aspect of high-probability mutation is its ease-of-use: it consists of sim-

ply choosing a mutation operator probability ≈ 0.95. In the Simple Genetic Algorithm case and its close derivatives, there is no need to perform other algorithm changes. For existing algorithmic implementation, this generally means there is no additional development cost, source code modification or, for closed-source or antiquated implementations, recompilation.

This also means that the computational costs associated with high-probability mutation are very small. In most Genetic Algorithms, the time devoted to mutation is small compared to the costlier operations, like genome evaluation.

Bibliography

[1] Flickr, a yahoo company, www.flickr.com/services/api.

[2] H. Abdi. Holm's sequential Bonferroni procedure. *Encyclopedia of research design*, 1, 2010. Sage Thousand Oaks, CA.

[3] N. E. Croitoru. High-probability mutation in basic genetic algorithms. *Proceedings of the 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), IEEE*, pages 301–305, September 2014.

[4] N. E. Croitoru. High probability mutation and error thresholds in genetic algorithms. *Proceedings of the 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), IEEE*, pages 271–276, September 2015.

[5] N. E. Croitoru. Lowering evolved artificial neural network overfitting through high-probability mutation. *18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, September 2016.

[6] K. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.

[7] K. Deb and D. E. Goldberd. Analyzing deception in trap functions. *FOGA*, 2:98–108, July 1992.

[8] L. C. W. Dixon and G. P. Szego. *Towards Global Optimization II.* New York: North Holland, 1978.

[9] M. Eigen. Selforganization of matter and evolution of biological macromolecules. *Naturwissenschaften*, 58(10):465–523, 1971.

[10] M. Eigen and P. Schuster. *The Hypercycle: A Principle of Natural Self-Organization.* Berlin: Springer-Verlag, 1979.

[11] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U. Michigan Press, 1975.

[12] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[13] M. G. Lorenz and W. Wackernagel. Bacterial gene transfer by natural genetic transformation in the environment. *Microbiological reviews*, 58(3):563–602, 1994. Am. Soc. Microbiol.

[14] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Berlin, Heidelberg, New York: Springer-Verlag, 1992.

[15] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, 1996.

[16] M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. *Proceedings of the first european conference on artificial life*, pages 245–254, 1992. Cambridge: The MIT Press.

[17] G. Ochoa. Consensus sequence plots and error thresholds: tools for visualising the structure of fitness landscapes. *Parallel Problem Solving from Nature*, VI:129–138, 2000.

[18] G. Ochoa. *Error Thresholds and Optimal Mutation Rates in Genetic Algorithms*. PhD thesis, COGS, The University of Sussex, Brighton, UK, 2001.

[19] G. Ochoa. Error thresholds in genetic algorithms. *Evolutionary Computation Journal*, 14(2):157–182, 2006. MIT Press.

[20] G. Ochoa, I. Harvey, and H. Buxton. Error thresholds and their relation to optimal mutation rates. *European Conference on Artificial Life (ECAL'99), Lecture Notes in Artificial Intelligence 1674, Springer-Verlagm*, 1999.

[21] L. A. Rastrigin. Convergence of random search method in extremal control of many-parameter system. *Automation and Remote Control*, 24(11):1337, 1964. PLENUM PUBL CORP CONSULTANTS BUREAU, 233 SPRING ST, NEW YORK, NY 10013.

[22] L. A. Rastrigin. *Systems of extremal control*. Zinatne, 1974.

[23] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[24] P. Schuster and J. Swetina. Stationary mutant distributions and evolutionary optimization. *Bulletin of mathematical biology*, 50(6):635–660, 1988.

[25] H.-P. Schwefel. *Numerical optimization of computer models*. Chichester: Wiley & Sons, 1981.

[26] B. L. Welch. The generalization of "Student's" problem when several different population variances are involved. *Biometrika*, 34(1-2):28–35, 1974.

[27] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[28] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.