Evolutionary Traps

Eytan H. Suchard,

Abstract

Genetic Algorithms are a good method of optimization if the fitness function to be optimized conforms to some important demands. An important demand from a fitness function is that the sought for gene can be approached by cumulative mutations that improve the fitness function. However, even simple fitness functions that comply with that demand and that are not deceptive can be hard for Genetic Algorithms. The reason is that there should be another demand, that the Markov chain which models the intermediate genes leading to the optimal gene, has a probability that doesn't tend to zero as the gene grows. This demand is not always fulfilled when there are attractors other than the optimal gene. The offered fitness function has a parameter, namely 4. If the parameter is below 4 than canonical genetic algorithms can converge to the optimal solution. If the parameter is 4 and above, canonical genetic algorithms will converge to genes that are not optimal. Values below 4 and close to 4 e.g. 3.90 can be used to test the speed of convergence and thus the offered fitness function is an important test tool for general genetic algorithms.

Keywords: Genetic Algorithms, Deceptive fitness functions, Evolutionary Traps

ANT / ANB – Applied Neural Technologies / Biometrics - http://www.anbsys.com
Email: eytan_il@netvision.net.il
Phone: +972-4-8700391
Facsimile: +972-4-8732622

1. Introduction - Constructing a hard fitness function

It is important to understand the motivation of this paper. It is legitimate to test the limits of the capabilities of genetic algorithms to converge to a globally optimal gene. Every computational tool has its limits. These limits are put to test via difficult cases. This paper describes a fitness function that challenges standard Genetic Algorithms by its convergence to local optimum genes. This situation can be remedied by large scale especially tailored mutations such as flipping all the bits of the binary gene, however, by representing each bit by a pair of two bits and a XOR operation between them, it is easy to construct an example of a new fitness function that defies large scale bit flips. A pair of two bits in the new gene with the new fitness function will be then 01 or 10 representing 1 and 00 or 11 representing 0. A fitness function that defies bit rotations can also be constructed by adding bits in which a certain sequence of 1s and 0s must always appear. More and more complex fitness functions can be constructed such that they defy large scale (many bit flips) tailored mutations.

In other words, large scale especially tailored mutations represent knowledge about the fitness function that has to be optimized.

There are ways to achieve convergence even if the fitness function is hard, by simply forcing genes to be different from each other. An external control that punishes for resemblance is equivalent to brute force search because a large enough population in which all individuals are forced to be different from each other must cover all possible genes. GAs - Genetic Algorithms – that use very large populations are also equivalent to brute force search because then every mutation, even most rare has a feasible probability, however if the size of the population is more than the atoms that constitute the crust of the Earth then the GAs become unrealistic.

Let us use the word Crux to represent the following function of m

$$\text{Crux} = 2^1 + 2^3 + 2^5 + \ldots + 2^{(2m+1)}, m \in \{1,2,3,\ldots\}, m > 10 \qquad \textbf{(1.1)}$$

So Crux is the sum of all odd powers of 2 up to the power $2m+1$.

In binary representation it is 010101010101 …

Let the numbers

$$2^0, 2^1, 2^2, \ldots, 2^{2m}, 2^{2m+1} \qquad \textbf{(1.2)}$$

be represented in binary digits by either 1 if they are summed or 0 if not in the following term, Sum. Let Sum designate the sum of the powers of two that were selected by the digit 1.

$$\text{Sum} = \sum_{k=0}^{2m+1} \alpha_k 2^k \text{ s.t. } \alpha_k \in \{0,1\} \qquad \textbf{(1.3)}$$

We will call Sum, the "gene" and α_k, its representation.

Let

$$Delta = |Crux - Sum| \qquad \textbf{(1.4)}$$

Or $Delta = |010101010101\ldots.. - Sum|$ in binary representation.

Now let us provide an additional arbitrary rule in building our fitness function - if some of the numbers

$$2^0, 2^2, 2^4, \ldots, 2^{2k}, \ldots, 2^{2m} \qquad \textbf{(1.5)}$$

are summed in (1.3) then $Delta$ is divided by 4 to the power of the number of selected numbers from the set $2^0, 2^2, 2^4, \ldots, 2^k, \ldots, 2^{2m}$, in other words, even powers of 2.

The result is the Target / Fitness function that we will try to minimize.

$$\text{Target} = \frac{Delta}{4^{Factor}} \quad \text{s.t. Factor} = \sum_{k=0}^{m} \alpha_{2k} \qquad \textbf{(1.6)}$$

Please note that the number 4 whose powers constitute the denominator is discussed in the abstract.

For example 10101000 will be (Crux - (1+4+16))/(4*4*4).

010101 will be Crux - (2+8+32) and will not be further divided by 4 because no even power of 2 was summed in (1.3). In other words, if $\alpha_{2\mathrm{k}} = 0$ for $0 \le k \le m$

$\mathrm{Target} = Delta$.

The target function which is described in (1.6) is the fitness function that will be discussed $\mathrm{Fitness} \equiv \mathrm{Target}$. The term "target" is mostly used in optimization problems rather than in Genetic Algorithms.

Gene definition: The gene will be $\alpha_{\mathrm{k}} \in \{0,1\}$ s.t. $\mathrm{k} \in \{0,1,2,..2\mathrm{m}+1\}$

If there are 52 bits then the optimal gene is $\alpha_{2\mathrm{k}} = 0, \alpha_{2\mathrm{k}+1} = 1$ or in bit semantics

0101010101010101010101010101010101010101010101010101

which minimizes the fitness function to 0. The choice of the denominator in (1.6) is not trivial at all. Other choices could be made such as denominator 5^{Factor}

$\mathrm{Target} = \dfrac{Delta}{5^{Factor}}$ s.t. $\mathrm{Factor} = \sum_{\mathrm{k}=0}^{\mathrm{m}} \alpha_{2\mathrm{k}}$ but such a choice would yield a function which is not interesting to GA researchers.

The reason for this choice $\mathrm{Target} = \dfrac{Delta}{4^{Factor}}$ of a fitness function will be seen in the following analysis.

2 Analysis

Before we proceed it is useful to mention another work on hard fitness functions, J. Horn, D.E. Goldberg 1995 [1]. There are some important differences.

For example : 4^{Factor} in (1.6) can be replaced by C^{Factor} for some constant C. If $C \ge 4$ then simple Genetic Algorithms do not converge to the optimal gene 010101...01. So we have a family of hard fitness functions for GAs and we can choose the level of hardness.

We first recall from Genetic Algorithms that if there are n bits in the gene and the probability of a bit flip is some $0 < q < 1$. The probability that k bits at specific indices will be flipped (1 to 0 or 0 to 1) is $q^k(1-q)^k$.

Special symbols definition:

The ratio of probabilities between k bit flips and L bit flips both at specific k indices of α_k in (1.3) and (1.6) will be defined as

$r(k,L) \equiv \dfrac{q^k(1-q)^{n-k}}{q^L(1-q)^{n-L}} = \dfrac{q^{k-L}}{(1-q)^{k-L}}$ and If $q < \dfrac{1}{2}$ then let p designate $p \equiv \dfrac{q}{1-q} < 1$

then $r(k,L) = \dfrac{p^k}{p^L}$ and for $\dfrac{\mathrm{k}}{\mathrm{L}} \to \infty \Rightarrow r(k,L) \to 0$. In the analysis let the symbol $\{p^k\}$ designate $q^k(1-q)^k$.

The analysis assumes $q < \dfrac{1}{2}$ and thus $k > L \Rightarrow r(k,L) < 1$. The strategy of the analysis will be to show that genes will statistically need more bit flips, say $k > 3$ to reduce edit distance from the
optimal gene 01 01 … 01 than the bit flips that are required to reduce the fitness function (GAs try to minimize the fitness function).

Throughout the analysis, bits will be arranged in pairs. Sequence
… 10 10 10 …10 will mean that bits indexed
$2\mathrm{r}, 2\mathrm{r}+2, 2\mathrm{r}+4, \ldots 2\mathrm{r}+2\mathrm{h},\ \mathrm{r},\mathrm{h} \in \{0,1,2,..\mathrm{n}/2\}$ will be set to 1 and bits indexed
$2\mathrm{r}+1, 2\mathrm{r}+3, 2\mathrm{r}+5, \ldots, 2\mathrm{r}+2\mathrm{h}+1$ will be set to 0 such that n is the number of bits
$n = 2m+2$ regarding (1.1)-(1.6).

Conversely, the sequence
… 01 01 01 … 01 will mean that bits indexed $2\mathrm{r}, 2\mathrm{r}+2, 2\mathrm{r}+4, \ldots 2\mathrm{r}+2\mathrm{h}$ will be set to 0 and bits indexed $2\mathrm{r}+1, 2\mathrm{r}+3, 2\mathrm{r}+5, \ldots, 2\mathrm{r}+2\mathrm{h}+1$ will be set to 1.

There exist genes namely,
10 10 10 …. 10 10 11 and 11 11 11 …. 11 11 10
to which the algorithm converges with probability that tends to 1 as the number of bits grows, instead of converging to the optimal gene 0101 … 01010101 providing that complex mutations, such as inverting all the bits, are not used. Please remember the short discussion in the introduction about fitness

functions that are unaffected by bit inversion or other large scale mutations.
We will focus on the definition of the fitness function (1.6) as is, such that mutations are simply random bit flips rather than highly organized flips as mentioned in the introduction. Also if $q > \frac{1}{2}$ then $r(k,L) > 1 \ s.t.\ k > L$ and the results that we reach will not hold for such an extremely high mutation probability.

Progress by mutation

Obviously if we treat the genes that change during runtime as a Markov chain we have to show that the ratio between the probability to get correct mutations – ones that reduce edit distance from the optimal gene –and to get other mutations that reduce the fitness function is greater than 1 for all genes of the chain.
Let us recall $\mathrm{Crux} = 2^1 + 2^3 + 2^5 + \ldots + 2^{(2\mathrm{m}+1)}$ which is represented by sum over 01 01 …. 01.
We now evaluate the following 10 10 10 …. 10 10 1**1** 01 01 … 01 01 01 **(2.1)**
such that the last 1 from left to right out of the pair of two successive 1s is bit $2r+1$ starting from bit index 0 on the left. In other words $\alpha_{2r+1} = 1$ and also $\alpha_{2r} = 1$.
Bit $\alpha_{2r+1} = 1$ represents $-2^{2\mathrm{r}+1}$ when calculating $Delta$, (1.4) so its contribution to the numerator in (1.6) is $2^{2\mathrm{r}+1} - 2^{2\mathrm{r}+1} = 0$. The pair $\alpha_{2r} = 1, \alpha_{2r+1} = 1$ contribute $2^{2\mathrm{r}+1} - 2^{2\mathrm{r}+1} - 2^{2\mathrm{r}} = -2^{2\mathrm{r}} = +2^{2\mathrm{r}} - 2 * 2^{2\mathrm{r}} = 4^r - 2 * 4^r$. We will use this simple equality and our $Delta$ for this "gene" becomes by (1.1) and (1.4),

$$Delta = |Crux - Sum| =$$

$$\left|(2^{2\mathrm{r}+1} - 2^{2\mathrm{r}+1} - 2^{2\mathrm{r}}) + (2^{2\mathrm{r}-1} - 2^{2\mathrm{r}-2}) + (2^{2\mathrm{r}-3} - 2^{2\mathrm{r}-4}) + \ldots + (2-1)\right| = \qquad \textbf{(2.2)}$$

$$\left|-4^r + (4^{r-1} + 4^{r-2} + 4^{r-3} + \ldots + 4^0)\right| = \left|\frac{4^r - 1}{3} - 4^r\right| = \left|-\frac{2}{3}4^r - \frac{1}{3}\right| = \frac{2}{3}4^r + \frac{1}{3}$$

and the fitness function to be minimized by our algorithm becomes

$$\text{Target} = \frac{Delta}{4^{Factor}} = \left|\frac{4^0 + 4^1 ... + 4^r - 2*4^r}{4^{r+1}}\right| =$$

$$\left|\frac{\frac{4^{r+1}-1}{3} - 2*4^r}{4^{r+1}}\right| = \left|\frac{1}{3} - \frac{1}{2} - \frac{1}{3*4^{r+1}}\right| = \left|-\frac{1}{6} - \frac{1}{3*4^{r+1}}\right| \qquad \textbf{(2.3)}$$

The term

$$\frac{1}{3*4^{r+1}} \qquad \textbf{(2.4)}$$

decreases when r grows or in other words,

if the algorithm already converged to 10 10 10 …. 10 10 11 01 01 … 01 01 01

then the fitness function for 10 10 10 …. 10 10 10 11 01 … 01 01 01

such that the last 1 out of the pair of two successive 1s, is bit $2r+3$ starting from bit index 0 on the left and that represents -2^{2r+3} when calculating $Delta$, (1.4).

The fitness function becomes,

$$\text{Target} = \frac{Delta}{4^{Factor}} = \frac{1}{6} + \frac{1}{3*4^{r+2}} < \frac{1}{6} + \frac{1}{3*4^{r+1}} \qquad \textbf{(2.5)}$$

then moving (mutating) in the wrong direction of evolution (edit distance from the optimum gene grows with evolution) by flipping two bits decreases the fitness function that we try to minimize !!!

If the probability of a bit flip is q then the probability of two bits flip is $\{P^2\}$.

Comparing that probability to the bits inversion of the sub-sequence 10 10 10 …. 10 10 10 suppose that the sub-sequence has $2k$ bits, then the ratio between $\{P^{2k}\}$ and $\{P^2\}$ is $r(2k,2)$ which tends to zero as k grows. In the rest of the discussion we may implicitly refer to $r(k,2)$, $r(k,3)$, $r(2k,2)$, $r(k,1)$ simply by writing $\{p^2\}$, $\{p^3\}$, $\{p\}$ to compare between the probability of bit flips that are required to reduce the edit distance to the optimized gene versus the probability

of bit flips that are required to reduce the fitness function.
Unless all $2k$ bits are flipped, any mutation in the sequence 10 10 10 .... 10 will only increase the fitness function that we try to minimize. Let us now examine another more general “gene”

01 01 01 .... 10 10 10 ... 10 10 11 01 01 ... 01 01 01 **(2.6)**

for which the first wrong bit (in relation to the best “gene” 01 01 01 ... 01), is bit $2u$ starting from bit index 0 on the left on the left encodes $-2^{2u}, \text{s.t.} u < r$ when calculating $Delta$, (1.4) and the contribution of the subsequence $\alpha_{2u} = 1 \wedge \alpha_{2u+1} = 0$ to the target function is $2^{2u+1} - 0 * 2^{2u+1} - 1 * 2^{2u} = 4^u$. The last 1 out of the pair of two successive 1s represents -2^{2r+1} or more precisely this number is subtracted from $Crux$ and thus the contribution of $\alpha_{2r+1} = 1$ to the target function is $2^{2r+1} - 2^{2r+1} = 0$.
Our fitness function then becomes

$$\text{Target} = \frac{Delta}{4^{Factor}} = \left| \frac{4^u(4^0 + \ldots + 4^{r-u} - 2 \cdot 4^{r-u})}{4^{r-u+1}} \right| =$$

$$\left| \frac{4^u(\frac{4^{r-u+1} - 1}{3} - 2 \cdot 4^{r-u})}{4^{r-u}} \right| = (\frac{1}{6} + \frac{1}{3 * 4^{r-u+1}})4^u \qquad \textbf{(2.7)}$$

(2.7) shows that even if the GA is on its way to the best gene 01 01 01 .... 01 01 01 $\alpha_{2k} = 0, \alpha_{2k+1} = 1$ then any wrong sub-sequence ...01 01 01 **10 10 10 ... 10 10** 01 01

01 01... in the middle of the sequence grows either left by changing 01 to 10 with probability $\{p^2\}$ or by changing two right bits 01 to 10 with probability $\{p^2\}$ which decreases u which decreases the fitness function (2.7) or to the right which increases r and thus decreases the fitness function in (2.7). If the terms “right” and “left” annoy the reader, the author would say that using “left” for low indices

and “right” for high indices i of α_i s.t. $0 \leq \mathrm{i} \leq 2\mathrm{m}+1$ are intuitive and are easier to follow. Other options of wrong mutations will be discussed next. It is mentioned again that for a sequence 10 10 … 10 10 10 with $2k$ bits, the probability that the sequence will become the optimal one 01 01 01 … 01 01 is $\{p^{2k}\}$ because the only way to reduce the fitness function and get a correct sequence 01 01 01 … 01 is to flip the entire $2k$ bits.

The shortest edit distance to reduction of the fitness function in (2.7) is by flipping two successive bits, either bits $2r+1$ and $2r+2$ counting from bit index 0 on the left or by flipping bits $2u-2$ and $2u-1$. Again the probability of progress which gets farther from the gene is $\{P^2\}$, given that the probability of a bit flip is q.

A very similar proof to (2.5) can be constructed for the gene

11 11 11 .... 11 11 11 … 11 **11 10 01** 01 … 01 01 01 **(2.8)**

Such that the first zero bit is bit $2r+1$ starting from bit 0 to the right.

The fitness function then evaluates to,

$$\text{Target} = \frac{Delta}{4^{Factor}} = \left| \frac{2^{2r+1} - 2^{2r} - 2^{2r-2} - \ldots 1}{4^{r+1}} \right| = \frac{\frac{4^{r+1}}{2} - \frac{4^{r+1}-1}{3}}{4^{r+1}} = \frac{1}{6} + \frac{1}{3*4^{r+1}} \qquad \textbf{(2.9)}$$

Flipping three bits and 10 01 01transforms into 11 10 01 with probability $\{P^3\}$ and reduces the fitness function.

The probability of turning a sequence 11 11 11 … 11 11 11 into 01 01 01 … 01 01 01 when the number of pairs is k, is simply to flip k bits which is $\{p^k\}$ because if every other bits in the 11 11 11 … 11 sequence are flipped then the fitness function only increases and thus the entire wrong k bits have to be flipped. $\{p^k\}$ tends to zero for big k and thus flipping three bits with probability $\{P^3\}$ is more likely to occur because $r(k,3) = \frac{\{p^k\}}{\{p^3\}} = \{p^{k-3}\} < 1$.

Now we will explore a more general “gene”

01 01 01 …. 11 11 11 … 11 11 10 01 01 … 01 01 01

such that the sequence of successive 1 bits starts at bit $2u$ when the first bit index is 0 and the first 0 to the right of this sequence is the bit indexed $2r+1$. Similar arguments to the ones that lead to (2.7) yield the very same value,

$$\text{Target} = \frac{Delta}{4^{Factor}} = \left| \frac{-4^{u}(4^{0}+\ldots+4^{r-u})+2*4^{r-u}}{4^{r-u+1}} \right| = (\frac{1}{6}+\frac{1}{3*4^{r-u+1}})4^{u} \qquad \textbf{(2.10)}$$

It is easy to see that 01 01 01 …. 11 11 11 … 11 11 **01** 01 01 … 01 01 01

changes to 01 01 01 …. 11 11 11 … 11 11 1**0** 01 01 … 01 01 01

with probability $\{p^{2}\}$. Comparing that to k bit flips of 11 11 11 ... 11 into 01 01 01 … 01 we have a probabilities ratio $r(k,2)$.

By looking at sub sequences of the form 010101 … 10 10 10 10 … 11 01 01 … 01, and 11 11 11 11 … 11 11 10

the fitness function of every "gene" can be expressed by summation of several sums like (2.7)

$$\text{Target} = \sum_{k=1}^{S} (\frac{1}{6}+\frac{1}{3*4^{r_S-u_S+1}})4^{u_S} \qquad \textbf{(2.11)}$$

such that S is the number of such sub-sequences.

Now let us explore concatenation of two sub sequences of the form

10 10 10 … 10 11 10 10 10 … 11 **(2.12)**

It is easily verifiable that a single flip of the second bit out of the three successive ones with value 1 reduces the two sub sequences to one sub sequence and also reduces the fitness function. It is also the shortest edit distance that reduces the

fitness function which is (the edit distance) here the number of bit flips.
The probability of such a mutation is $\{P\}$, the probability of a single bit flip.
There are two other cases that we have to check, namely

11 11 11 … 11 11 10 10 10 10 … 11 **(2.13)**

and

10 10 10 … 10 11 11 11 11 11 … 10 **(2.14)**

Looking at (2.7) the power 4^u offers the greatest decrease of
the term $(\frac{1}{6}+\frac{1}{3*4^{r-u+1}})4^u$ which means that the most probable
mutation of (2.13) will be

11 11 11 … 11 11 10 10 10 10 … 11 **(2.15)**
11 11 11 … 11 1**0** 10 10 10 10 … 11

which is a flip of one bit. The probability of such a concatenation is $\{P\}$, the probability of a single bit flip (in relation to other bit flips).

For a very similar reason the most probable mutation of (2.14) will be

10 10 10 … 10 11 11 11 11 11 … 10 **(2.16)**
10 10 10 … 1**1** 11 11 11 11 11 … 10

We are almost done. What is left to handle is edit distances.
By (2.11), (2.12), (2.13), (2.14), (2.15), (2.16) for a sub sequence of the form

…10 10 10 … 10 11… **(2.17)**

Or

…11 11 11 … 11 10…
that has W bits, it takes $2W-1$ bit flips to reach a sub – sequence of the optimal "gene" / solution and otherwise the fitness function only grows if not all the $2W-1$ are flipped. "sub – sequence of the optimal gene" means …01 01 01 01 … 01….

We have also shown in (2.15) and (2.16) that it is most likely that the algorithm will statistically converge to a "gene" of the type that appears in the right sub-sequence. Such a progress is of probability $\{P\}$, the probability of a single bit flip. What we have shown is that if the "gene" edit distance from the best fitness function is Q then there is no other statistical way to reach that "gene" except for performing all Q bit flips otherwise the "gene" by selection will either flip two bits or one to get the fastest improvement.
Finally, the initial bit values are at random, sub sequences such as 10 10 … 10 and 11 11 11 … 11 are at unlimited number and at unlimited length as the number of bits in the gene n grows which proves the difficulty of the fitness function for Genetic Algorithms that use mutation operators alone because the probabilities to improve the gene by wrong mutations are .
The probability ratio of $r(k,3)=\frac{p^k}{p^3}, r(k,2)=\frac{p^k}{p^2}, r(k,1)=\frac{p^k}{p^1}$ s.t. $p=\frac{q}{1-q}$ which proves that unless all the bits are 01 01 01 01 … 01 01 then the most probable mutations only increase the edit distance from the optimal gene.

Progress by crossover

The discussion about (2.13) and (2.14) leads to the following cases,

Crossover between

10 10 10 … and 10 1010 10 …
or between
11 11 11 … and 11 11 11 …

or between

11 11 11 … and 10 10 10 10 … **(2.18)**

or between

01 01 01 … and 01 01 01 …

Or between

01 01 01 … and 11 11 11 …

Or between

01 01 01 … and 10 10 10 …

Crossover of 10 10 10 … and 10 1010 10 leaves the bits $\alpha_{2k+1} = 0, k = 0,1,2,...$ and thus do not contribute to reduction of edit distance from the optimal genes.
Crossover of 11 11 11 … and 11 11 11 … leaves the bits $\alpha_{2k} = 1, k = 0,1,2,...$ and thus do not reduce the edit distance either.
Crossover between 11 11 11 … and 10 10 10 10 leaves the bits $\alpha_{2k} = 1, k = 0,1,2,....$ By (2.15) and (2.16) the latter crossover will converge to one of the following genes, 11 11 11 11 … or 10 10 10 10 … with the already discussed probabilities.
The more interesting case is crossover between the sub sequences
01 01 01 … and 11 11 11 … or 01 01 01 … and 10 10 10 … .
By the arguments after (2.6) and after (2.10) the result of such crossover is inferior to crossover between 10 10 10 … and 10 1010 10 … or
11 11 11 … and 11 11 11.
To summarize, the crossover operator does not improve the convergence to the optimal gene unlike in Michael D. Vose, Alden H. Wright 1995 [2] discussion in which positive crossover rate does improve convergence to the optimal gene.
In our case the fitness function under canonical Genetic Algorithms has three attractors and it converges only to two of them.

There is a way to achieve convergence to the optimal gene by huge populations. In the case of 52 bits, a population of $\frac{1}{q^{26}}$ will averagely have one case in which all $\alpha_{2k}, k = 0,1,2,...$ bits can be $\alpha_{2k} = 0$. Such huge populations are used to mimic brute force search. Using for example 1024 bits fitness functions renders GAs that use such huge populations practically infeasible.

To understand how unsuccessful crossover between 10 10 10 … 10 and 01 01 01 01 … 01 sequences can be we now continue.

Calculating the contribution of the sequence 10 10 10 … 10 10 11 and one of the pairs replaced by 01 instead of the previous 10 such that the index of the 0 in 01 is $2S$ we have:

10 10 10 .. **01**, 10 … 10 10 11 **(2.19)**

$$\alpha_{2S} = 0, \ \alpha_{2S+1} = 1 \qquad \textbf{(2.20)}$$

Let the index of the left 1 be $2u$. By (2.7) before replacing the pair $\alpha_{2S} = 1, \ \alpha_{2S+1} = 0$ with $\alpha_{2S} = 0, \ \alpha_{2S+1} = 1$, the contribution of the sequence to the fitness function is

$$\left| 4^u \frac{\left(\frac{4^{r-u+1}-1}{3} - 2*4^{r-u}\right)}{4^{r-u-1}} \right| \qquad \textbf{(2.21)}$$

The 01 subsequence subtracts $4^s = 2^{2s+1} - 2^s$ from the numerator and divides the denominator by 4 because the false bit $\alpha_{2S} = 1$ was flipped to $\alpha_{2S} = 0$. So we have:

$$\left| 4^u \frac{\left(\frac{4^{r-u+1}-1}{3} - 2*4^{r-u} - 4^S\right)}{4^{r-u}} \right| = 4^u\left(\frac{1}{3} - 2 - 4^{s-r+u}\right) \qquad \textbf{(2.22)}$$

Which means that any such crossover dramatically increases the target function that our genetic algorithm is supposed to minimize.
Other crossover results between 01 01 01 … 01 and 10 10 10 … 10 also increase the function that the genetic algorithm is supposed to minimize.
A very similar analysis follows the crossover between 01 01 01 … 01 and 11 11 11 … 11.
Let $C(g_a, g_b)$ be defined as the crossover operator between genes g_a and g_b. Crossover $C(g_a, g_b)$ proves to be successful if odd indexed (starting from index 0) bits taken from g_a are equal to 1, even indexed bits (starting from index 0) taken from the first gene g_a are equal to 0 and the same applied to gene g_b. The "progress by mutation" analysis shows that sequences such as 10 10 10 10 … 11 will most likely grow in the discussions between (2.3) and (2.7) and also between (2.9) and (2.10). Unless most of the bits will form the sequence 01 01 01 01 ,,, $\alpha_{2S} = 0,\ \alpha_{2S+1} = 1$, the resulting gene will only increase the target function that the algorithm is supposed to minimize and thus crossover will be unsuccessful.

3. Experimental performance

The division by power of 4 produces genes for which the fitness function is well below 1 and well above 0 and yet are not the optimal, for which the fitness function is 0.00. Artificial punishments for resemblance to other individuals (genes) of the population and other advanced GA methods did not solve the problem of convergence to the wrong gene although such intervention in the process is quite typical to GAs and in large populations is equivalent to brute force search. Complex tailored mutations such as inverting all the bits did help, however, by modification to the fitness function it is easy to construct a new fitness function for which large scale bit inversion and bit rotations do not change the fitness function or makes it bigger if we try to minimize the function.
The reader may say that the fitness function (1.6) is designed to fail Genetic Algorithms. That is quite true. A lot of thought was put in (1.6) prior to this paper,

However, it raises a legitimate need to develop a theory that will identify which fitness functions can be and which can't be minimized or maximized using Genetic Algorithms. The offered fitness function may be hard to other optimization methods and not only to genetic algorithms.

Here are some hands-on results. Simulations included 52 bits.

The 52 bits optimal gene that was evasive is:

0101010101010101010101010101010101010101010101010101 Target 0.00 **(3.1)**

Under 0.39% and 1.56% mutations, the 52 bits program converged to:

1111111111111111111111111111111111111111111111111110 Target 0.17 **(3.2)**

or to

1010101010101010101010101010101010101010101010101011 Target 0.17

On the way to the first two genes, stayed hundreds of generations in …

1111111111111111111111111111111111111111111111100101 Target 0.17

1111111111111111111111111111111111111111111111111001 Target 0.17

1111111111111111111111111111111111111111111001010101 Target 0.17

The latter is unstable due to (2.8).

With 12.5% mutation the false gene that is mostly reached after thousands of generations is:

1010101010101010101010101010101010101010101010101011 Target 0.17

Conscious rotation to the right of all the bits or inversion of all bits of the last gene yields almost the real gene, however, by re-indexing or adding some bits to slightly modify the fitness function, even rotation which consists of 52 ordered mutations doesn't reach the optimal "gene". Example for re-indexing, replace the indices of 2^0 and 2^1 then of 2^4 and 2^5 etc. In general, swap the indices of 2^{4k} and 2^{4k+1} leave all the other indices as they were and update the definition of (1.6).

$$\text{Target} = \frac{Delta}{4^{Factor}} \quad \text{s.t.} \quad \text{Factor} = \sum_{k=0}^{4k+2<2m+1} \alpha_{4k+2} + \sum_{k=0}^{4k+1\leq 2m+1} \alpha_{4k+1} \qquad \textbf{(3.3)}$$

(3.3) shows that even a previously tailored set of ordered mutations need not solve our convergence problem in the most general case of minimizing arbitrary fitness functions. There was an attempt by the author to avoid Drift and Scaling problems as defined by Rudolph, G [3], and J.L. Shapiro [4] - by controlling diversity in the population. This was done by multiplication of single individual results by a wide range of arbitrary punishments. The leading individual was selected as the one that achieved the smallest fitness function. The second result was multiplied by a punishment 1.1 if the gene of the second individual was the same as the gene of the first chosen individual. The best such gene was sought for. The third individual was chosen by calculating its fitness function. Then if the third individual was identical to the first, then the fitness was multiplied by 1.1 and if it was identical to the second it was further punished by an additional multiplication by a factor of 1.1. A third such best fitness function was sought for. The selection process continued like in the description above until a quarter of the population was selected. In large populations the offered selection algorithm is obviously equivalent to brute force search. To summarize, large scale tailored mutations and external population control, or their equivalent terms - brute force search and complex conscious mutations - such as Lin-Kernighan transformations are means that are used to help the genetic algorithm to overcome local minima. For references on large scale mutations in the Traveling Salesman Problem genetic algorithm, the reader can refer to D. Applegate, R. E. Bixby, V. Chvàtal & W. Cook [5] and K-T. Mak & A. J. Morton [6].
Part of the large scale mutations – that involve an algorithm for more than one bit flip – have a “natural counterpart“ in retrotransposon and transposon jumps. There is a lot of bibliography on retrotransposons, e.g. Eugene M McCarthy and John F McDonald [7]. Unfortunately large scale mutations do not remedy convergence problems in the general case as mentioned in the introduction.

Here is an example of code that forces the Genetic Algorithm to perform brute force search.

```
// ---------------------------------------------------
// Population degeneracy prevention algorithm.
// ---------------------------------------------------

// Sort the first survivors/2 by fitness and be difference
// from leading. This algorithm promotes some gene diversity.

s_half = member_n_survivors >> 1;

s_idx = s_sort_class.member_array[0].member_index;

member_organism_class_array[member_n_population] =
member_organism_class_array[s_idx];

member_organism_class_array[s_idx].function_print_console();

printf(" %.2lf",member_results_array[s_idx]);

member_results_array[member_n_population] =
member_results_array[s_idx];

for(i=1;i<s_half;i++)
{
  double s_grade,s_min=0;

  int j,s_min_idx = -1;

  for(j=0;j<member_n_population;j++)
  {
    s_grade = member_results_array[j];

    for(k=0;k<i;k++)
    {
      if (member_organism_class_array[member_n_population + k] ==
          member_organism_class_array[j])
      {
        s_grade *= 1.1; // Punish for identity.
      }
    }

    if (s_grade < s_min || s_min_idx < 0)
    {
      s_min = s_grade;
      s_min_idx = j;
    }
  }


  member_organism_class_array[member_n_population+i] =
  member_organism_class_array[s_min_idx];

  member_results_array[member_n_population+i] =
  member_results_array[s_min_idx];
}

// ---------------------------------------------------
// End of 'Population degeneracy prevention algorithm'.
// ---------------------------------------------------
```

Experimental report:

The settings of the Genetic Algorithms that were tried can be viewed in Table 1 (In a separate document).

4. Short analysis

What we have shown is the following

Let $G_1, G_2, G_3, G_4, ..., G_L$ represent genes such that:

4.1. G_1 is a zero initial state in which zero is assigned to all the bits of the gene. In the program both random initialization and zero initializations can be used.

4.2. The edit distance $\text{Edit\_Dist}(G_i, G_{i+1}) = 1$ and the fitness function for G_{i+1} is better than in G_i, $T\arg et(G_{i+1}) < T\arg et(G_i)$.

4.3. G_L yields the minimal fitness function.

4.4. By (2.4) the difference in the fitness function that is sufficient for the algorithm not to converge to the best "gene" is infinitesimally small.
4.4 is a bit surprising (and not only as a game of words).

Then GAs need not converge to G_L !
Obviously the states 01000000……, 0101000000000…., 010101000000000…
which represent $2^1, 2^1 + 2^3, 2^1 + 2^3 + 2^5, 2^1 + 2^3 + 2^5 + ... + 2^{(2k+1)}$ form such Gs.

5. How to run the demo – online publication

The C++ console application files GA.CPP,MSORTIDX.CPP,RAND.CPP, MSORTIDX.H,RAND.H have to be included in a new Console Application project and should be compiled.

There are three arguments that the program expect:

5.1. Number of generations. Typical values are between 2000 and 1000000.

5.2. Deciding whether the fitness function is (1.6) or simply (1.4). If this number is 0 then the fitness function uses a denominator which consists of powers of 4 as previously mentioned. If not then the fitness function is simply the absolute value |Crux - Sum|.

5.3. The third parameter is the number of survivors in each generation or epoch. The number of individuals in the population is four times that number. Each surviving individual will have at least three offspring.

Program example command line is: GA 1000000 0 80

To stop the program while running then please press S or s or N or n.

6. Conclusions

GAs are a stochastic way to optimize a target function that is also known as Fitness function. There could be some traps, however, even infinitesimally small as the number of bits grow, that can fool the genetic algorithm and cause it to converge to genes that are very far in terms of edit distance from the optimal ones.

This article shows the need for a theory that will be able to point out which fitness functions can be and which can't be optimized by genetic algorithms.

Such a theory is a productive goal in the research of GAs.

7. Acknowledgement

My thanks are to my Colleagues Mr. Raviv Yatom and Mr. Irad Heller for their remarks on Genetic Algorithms.

8. References

[1] J. Horn, D.E. Goldberg, Genetic Algorithms Difficulty and the Modality of Fitness Landscapes, In L.D. Whitley & M.D. Vose (Eds.), Foundations of Genetic Algorithms 3 (FOGA 3), pp. 243-269, Morgan Kaufmann. (1995)

[2] Michael D. Vose, Alden H. Wright, Stability of Vertex Fixed Points and Applications, In D.Whitley and M.Vose (Eds.) Foundations of Genetic Algorithms 3, Morgan Kaufmann, San Mateo, CA, 103-114. 1995.

[3] Rudolph, G., "Convergence analysis of canonical genetic algorithms", IEEE Transactions On Neural Networks, Jan. 1994, pages 96-101

[4] J.L. Shapiro, "Drift and Scaling in Estimation of Distribution Algorithms", Evolutionary Computation, MIT Press Cambridge, MA, USA, 2005.

[5] D. Applegate, R. E. Bixby, V. Chvàtal & W. Cook, “Data Structures for the Lin-Kernighan Heuristic”, Talk presented at the TSP-Workshop, CRCP, Rice University (1990).

[6] K-T. Mak & A. J. Morton, “A modified Lin-Kernighan traveling-salesman heuristic”, Oper. Res. Let., 13, 127-132 (1993).

[7] Eugene M McCarthy and John F McDonald, "Long terminal repeat retrotransposons of Mus musculus" Genome Biology 2004, 5:R14, 13 February 2004