Genetic Algorithms and Irreducibility,

Eytan H. Suchard,
Metivity Ltd.
Email: Eytan_il@netvision.net.il
Phone: +972-4-8700391

Abstract

Genetic Algorithms are a good method of optimization if the target function to be optimized conforms to some important properties. The most important of all is that the sought for solution can be approached by cumulative mutations such that the Markov chain which models the intermediate genes has a probability that doesn't tend to zero as the gene grows. In other words each improvement of the gene - set of 0s and 1s - follows from a reasonable edit distance – minimum number of bits that change between two genes - and the overall probability of these mutations does not vanish. If for reaching an improvement, the edit distance is too big then GAs are not useful even after millions of generations and huge populations of millions of individuals.
If on the other hand the probability of a chain of desired mutations tends to zero as the chain grows then also the GA fails.
There are target functions that can be approached by cumulative mutations but yet, statistically defy GAs. This short paper represents a relatively simple target function that its minimization can be achieved stepwise by small cumulative mutations but yet GAs fail to converge to the right solution in ordinary GAs .

Keywords: Genetic Algorithms, Irreducible Genes.

1. Constructing the fitness function

We will choose a fitness function for which we will construct an easy C++ program. Most important is that our target function will have a gradual path by which it can be minimized unlike for example a key to a safe !!!

The fitness function is as follows:

Let

$$\text{Crux} = 2^1 + 2^3 + 2^5 + ... + 2^{(2m+1)} \qquad \textbf{(1.1)}$$

Let the numbers

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

be represented by either 1 if they are summed or 0 if not.

Let Sum be the sum of the powers of two that were selected by the digit 1.

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

Let

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

Now let us provide an additional rule - if some of the numbers

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

are selected then $Delta$ is divided by 4 to the power of the number of selected numbers from the set $2^0, 2^2, 2^4, ..., 2^k, ..., 2^{2m}$.

The result is the target function that we will try to minimize.

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

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

010101 will be Crux - (2+8+32).

If there are 52 bits then the optimal solution is

0101010101010101010101010101010101010101010101010101

which minimizes the target to 0. The choice of the denominator in (1.6) is not trivial at all. Other choices could be made such as

$\text{Target} = \frac{Delta}{5^{Factor}}$ s.t. $\text{Factor} = \sum_{k=0}^{m} \alpha_{2k}$ but such a choice would yield a function

which is not interesting to evolutionary computation researchers.

The reason for this choice will be seen in the proof of following theorem.

2 Theorem

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 solution 0101 … 01010101 if complex mutations, such as inverting all the bits, are not used.

By reordering the indices of the bits or by adding bits that must be 0 for the target function to be optimal both bits inversion and bits rotation can't work. For now we will focus on the definition of the target function (1.6) as is, such that mutations are simply random bit flips rather than highly organized flips. That is because we can then rely on evolutionary computation theory and on the fact that the probability converges to zero if edit distance that is required to reach a better "gene" diverges in infinity as the number of bits grows.

Proof

Let us recall $\mathrm{Crux} = 2^1 + 2^3 + 2^5 + \ldots + 2^{(2m+1)}$.

We now evaluate following 10 10 10 …. 10 10 11 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.

This bit encodes -2^{2r+1} when calculating $Delta$, (1.4).

Our $Delta$ for this "gene" becomes by (1.1) and (1.4),

$$Delta = |Crux - Sum| =$$
$$\left|(2^{2r+1} - 2^{2r+1} - 2^{2r}) + (2^{2r-1} - 2^{2r-2}) + (2^{2r-3} - 2^{2r-4}) + ... + (2-1)\right| = \quad \textbf{(2.2)}$$
$$\left|-4^{r} + (4^{r-1} + 4^{r-2} + 4^{r-3} + ... + 4^{0})\right| = \left|\frac{4^{r}-1}{3} - 4^{r}\right| = \left|-\frac{2}{3}4^{r} - \frac{1}{3}\right| = \frac{2}{3}4^{r} + \frac{1}{3}$$

and the target function to be minimized by our algorithm becomes

$$\text{Target} = \frac{Delta}{4^{Factor}} = \frac{\frac{2}{3}4^{r} + \frac{1}{3}}{4^{r+1}} = \frac{1}{6} + \frac{1}{3 * 4^{r+1}} \quad \textbf{(2.3)}$$

The term

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

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 target 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 encodes -2^{2r+3} when calculating $Delta$, (1.4).
The target function becomes,

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

then moving in the wrong direction of evolution by flipping two bits decreases the target 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), 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 such that the last 1 out of the pair of two successive 1s encodes -2^{2r+1} when calculating $Delta$, (1.4). Our target function then becomes

$$\text{Target} = \frac{Delta}{4^{Factor}} = \frac{\frac{2}{3}4^{r-u} + \frac{1}{3}}{4^{r-u+1}} = (\frac{1}{6} + \frac{1}{3 * 4^{r-u+1}})4^{u} \quad \textbf{(2.7)}$$

(2.7) shows that even if the evolutionary program is on its way to the best solution 01 01 01 …. 01 01 01 then any wrong sub-sequence that ends with a correct bit …10 10 10 … 10 10 11… in the middle grows either left and decreases

u which decreases the target function (2.7) or to the right which increases r and thus decreases the target function in (2.7).

The shortest edit distance to reduction of the target 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 $2u-1$.

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 target function then evaluates to,

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

Now we will explore a more general "gene"

01 01 01 …. 11 11 11 … 11 11 10 01 01 … 01 01 01 **(2.10)**

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,

$$\text{Target} = \frac{Delta}{4^{Factor}} = \frac{\frac{2}{3}4^{r-u} + \frac{1}{3}}{4^{r-u+1}} = (\frac{1}{6} + \frac{1}{3 * 4^{r-u+1}})4^{u}$$

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

$$\text{Target} = \sum_{k=1}^{q} (\frac{1}{6} + \frac{1}{3 * 4^{r_q - u_q + 1}})4^{u_q} \qquad \textbf{(2.11)}$$

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

Now let use 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 reduces the two sub sequences to one sub sequence and also reduces the target function. It is also the shortest improvement in the target function in edit distance which is here the number of bit flips because our gene doesn't allow insertions.

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.

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 target function only grows if not all the $2W-1$ are flipped.

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.
What we have shown is that if the "gene" edit distance from the best target 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.

3. Experimental performance

The division by power of 4 produces solutions for which the target function is well below 1 and well above 0 and yet are not the optimal one for which the target function is 0.00. Artificial punishments for resemblance to other individuals of the population and other advanced GA methods did not solve the problem of convergence to the wrong solution although such intervention in the process is

quite typical in GAs. Complex mutations such as inverting all the bits did help, however, by addition of several bits that each adds a constant value to the target function and randomly indexing these bits, bit inversion fails.
The reader may say that the target 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 target functions can be and which can't be minimized or maximized using Genetic Algorithms.
Simulations included 52 bits.

The 52 bits real solution:

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)**

1010101010101010101010101010101010101010101010101011 Target 0.17

And on the way to the first two "genes" or solutions, stayed hundreds of generations in

1111111111111111111111111111111111111111111111100101 Target 0.17

1111111111111111111111111111111111111111111111111001 Target 0.17

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

1010101010101010101010101010101010101010101010101011 Target 0.17

This could be a result of some artificial control that was added to the genetic program and is probably not related to theory (the author which is also a programmer tries not to say the word Bug).
Only conscious rotation to the right of all the bits or inversion of all bits of the last

solution yields the real solution, however, by re-indexing or adding some bits to slightly modify the target 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).

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

(3.3) shows that even a conscious set of ordered mutations need not solve our convergence problem in the most general case of minimizing arbitrary target functions.

There was an attempt to avoid Drift and Scaling problems [1], [2] by promoting 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 target 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 target function then if the third individual was identical to the first then the target was multiplied by 1.1. and if it was identical to the second it was also punished by a factor of 1.1. A third such best target function was sought for.

The process went on like that until a quarter of the population was selected.

Complex conscious mutations such as Lin-Kernighan transformations [3], [4] - which resemble retrotransposon jumps [5] – and that are used in the Traveling Salesmen Problem are legitimate if we know the order of the bits and indeed a big mutation of inverting all the bits does solve the convergence problem that is seen in (3.2). The problem is that by minor amendments of the target function

such as changing the order of bits and adding some bits that must be 0 for the optimal “gene” these methods cease to work. In other words, using conscious complex mutations requires the target function to have a known structure.
The code for this anti Drift algorithm can be found in the code section of GA.CPP

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

// Sort the first survivors/2 by target 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'.
// ------------------------------------------------
```

For a short summery of the experiment:
The population was set to four times the number of survivors.
The mutation rate per bit was set to 0.39%, to 1.56% and to 12.5%.
In the program there is a parameter GA_MUTATION_MASK that is set to a power of 2 minus one.
If it is set to 255 then the mutation rate is 0.39% =~ 1/256.
If it set to 63 then the mutation rate is 1.56% =~ 1/64.
Survivors were 80 which means a total population of 320 and 200 which means a total population of 800 individuals.
The use of stochastically irreducible fitness function was detrimental to evolution and the program drifted towards wrong solutions.

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 $\mathrm{Edit\_Dist}(G_i, G_{i+1}) = 1$ and the target 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 target function.

4.4. By (2.4) the difference in the target 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

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. Whether the target function is statistically reducible or not. If this number is 0 then the target function uses a denominator which consists of powers of 4 as previously mentioned. If not then the target 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 full the genetic algorithm and cause it to converge to “genes” / solutions 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 which fitness functions can be and which can’t be optimized by genetic algorithms.
Such a theory is a productive goal in the fascinating research of evolutionary computation.

7. Acknowledgement

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

8. References

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

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

[3] 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).

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

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