ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

IN SEARCH OF INVERSION ANALOGY CHROMOSOMES BITS AND MACHINE LEARNING

Zhanyl S.Ixymbayeva*

[*] Computer Engineering and Software Faculty of S.Seifullin Kazakh AgroTechnical University, Astana, Republic of Kazakhstan

DOI: 10.5281/zenodo.290126

ABSTRACT

In this paper we proposed stated, a genetic algorithm is a programming technique that mimics biological evolution as a problem-solving strategy. Given a specific problem to solve, the input to the genetic algorithm is a set of potential solutions to that problem, encoded in some fashion, and a metric called a fitness function that allows each candidate to be quantitatively evaluated.

A genetic algorithms (GA) are machine learning search techniques inspired by Darwinian evolutionary models. The advantage of GA over factor analytic and other such statistical models is that GA models can address problems for which there is no human expertise or where the problem seeking a solution is too complicated for expertise based approaches.

KEYWORDS: A genetic algorithm, inversion, assembly language, crossover, chromosomes bits, binary numbers, mutation rate, binary representations

INTRODUCTION

The basic outline of a genetic algorithm is as follows [1]:

1. Initialise a population of individuals. This can be done either randomly or with domain specific background knowledge to start the search with promising seed individuals. Where available the latter is always recommended.
    - Individuals are represented as a string of bits. This is not a restriction for the type of problem because other data types (numbers, strings, structures) can also be encoded as bit strings.
    - A fitness function must be defined that takes as input an individual and returns a number (or a vector) that can be used as a measure of the quality (fitness) of that individual.
    - The application should be formulated such that the desired solution to the problem coincides with the most successful individual according to the fitness function.
2. Evaluate all individuals of the initial population.
3. Generate new individuals. The reproduction probability for an individual is proportional to its relative fitness within the current generation. Reproduction involves domain specific genetic operators. Operations to produce new individuals are:
    - Mutation. Substitute one or more bits of an individual randomly by a new value (0 or 1).
    - Variation. Change the bits in a way that the number encoded by them is slightly incremented or decremented.
    - Crossover. Exchange parts (single bits or bit strings) of one individual with the corresponding parts of another individual. Originally, only one-point crossover was performed but theoretically one can process up to L - 1 different crossover sites (with L as the length of the individual). For one-point crossover, two individuals are aligned and one location on their strings is randomly chosen as the crossover site. Now the parts from the beginning of the individuals to the crossover site are exchanged between the two. The resulting hybrid individuals are taken as the new offspring individuals.
4. Select individuals for the new parent generation.

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

- In the original genetic algorithm all offspring were selected while all parents were discarded. This is motivated by the biological model and is called total generation replacement.
- More recent variations of generation replacement compare the original parent individuals and the offspring which are then ranked by their fitness values. Only the n best individuals (n is the population size, i.e. the number of individuals in one generation) are taken into the next generation. This method is called elitist generation replacement. It guarantees that good individuals are not lost during a run. With total generation replacement good individuals may "die out" because they produce only offspring inferior in terms of the fitness function. Another variant is steady state replacement. There, two individuals are randomly selected from the current population. The genetic operators are applied and the offspring replace the parents in the population. Steady state replacement often converges sooner because on average it requires fewer fitness evaluations than elitist or total generation replacement.

5. Go back to step 2 until either a desired fitness value is reached or until a predefined number of iterations is performed.

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form that a computer can process. One common approach is to encode solutions as binary strings: sequences of 1's and 0's, where the digit at each position represents the value of some aspect of the solution. Another, similar approach is to encode solutions as arrays of integers or decimal numbers, with each position again representing some particular aspect of the solution. This approach allows for greater precision and complexity than the comparatively restricted method of using binary numbers only and often "is intuitively closer to the problem space" [2-3]

This technique was used, for example, in the work of Steffen Schulze-Kremer, who wrote a genetic algorithm to predict the three-dimensional structure of a protein based on the sequence of amino acids that go into it [4]. Schulze-Kremer's genetic algorithm used real-valued numbers to represent the so-called "torsion angles" between the peptide bonds that connect amino acids. (A protein is made up of a sequence of basic building blocks called amino acids, which are joined together like the links in a chain. Once all the amino acids are linked, the protein folds up into a complex three-dimensional shape based on which amino acids attract each other and which ones repel each other. The shape of a protein determines its function.)

For every application of a genetic algorithm one has to decide on a representation formalism for the "genes". In this application, the so-called hybrid approach is taken [5]. This means that the genetic algorithm is configured to operate on numbers, not bit strings as in the original genetic algorithm. A hybrid representation is usually easier to implement and also facilitates the use of domain specific operators.

However, three potential disadvantages are encountered:
1. Strictly speaking, the mathematical foundation of genetic algorithms holds only for binary representations, although some of the mathematical properties are also valid for a floating point representation.
2. Binary representations run faster in many applications.
3. An additional encoding/decoding process may be required to map numbers onto bit strings.

In this application a simple steric potential energy function was chosen as the fitness function (i.e. the objective function to be minimised). It is very difficult to find the global optimum of any potential energy function because of the large number of degrees of freedom even for a protein of average size. In general, molecules with n atoms have 3n - 6 degrees of freedom. For the case of a medium-sized protein of 100 residues this amounts to:
((100 residues * approximately 20 atoms per residue)* 3) — 6 = 5994
degrees of freedom [5]. Systems of equations with this number of free variables are analytically intractable today. Empirical efforts to heuristically find the optimum are almost as difficult. If there are no constraints for the conformation of a protein and only its primary structure is given the number of conformations for a protein of medium size (100 residues) can be approximated to:
(5 torsion angles per residue* 5 likely values per torsion angle)^{100} = 25^{100}.
This mean that in the worst case 25100 conformations would have to be evaluated to find the global optimum. This is clearly beyond the capacity of today's and tomorrow's super computers. As can be seen from a number of previous applications genetic algorithms were able to find sub-optimal solutions to problems with an equally large search space [6-8]. Sub-optimal in this context means that it cannot be proven that the solutions generated by the genetic algorithm do in fact include an optimal solution but that some of the results generated by the genetic algorithm

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

practically surpassed any previously known solution. This can be of much help in non-polynomial complete problems where no analytical solution of the problem is available.

Before you can use a genetic algorithm to solve a problem, a way must be found of encoding any potential solution to the problem. This could be as a string of real numbers or, as is more typically the case, a binary bit string. Let's refer to this bit string from now on as the chromosome. A typical chromosome may look like this:

10010101110101001010011101101110111111101

At the beginning of a run of a genetic algorithm a large population of random chromosomes is created. Each one, when decoded will represent a different solution to the problem at hand. Let's say there are N chromosomes in the initial population [9-11]. Then, the following steps are repeated until a solution is found

[1] Test each chromosome to see how good it is at solving the problem at hand and assign a fitness score accordingly. The fitness score is a measure of how good that chromosome is at solving the problem to hand.
[2] Select two members from the current population. The chance of being selected is proportional to the chromosomes fitness. Roulette wheel selection is a commonly used method.
[3] Dependent on the crossover rate crossover the bits from each chosen chromosome at a randomly chosen point.
[4] Step through the chosen chromosomes bits and flip dependent on the mutation rate.
[5] Repeat step 2, 3, 4 until a new population of N members has been created.

ROULETTE WHEEL SELECTION

This is a way of choosing members from the population of chromosomes in a way that is proportional to their fitness. It does not guarantee that the fittest member goes through to the next generation merely that it has a very good chance of doing so. It works like this:

Imagine that the population's total fitness score is represented by a pie chart, or roulette wheel. Now you assign a slice of the wheel to each member of the population. The size of the slice is proportional to that chromosomes fitness score. i.e. the fitter a member is the bigger the slice of pie it gets. Now, to choose a chromosome all you have to do is spin the ball and grab the chromosome at the point it stops.

This is simply the chance that two chromosomes will swap their bits. A good value for this is around 0.7. Crossover is performed by selecting a random gene along the length of the chromosomes and swapping all the genes after that point.

e.g. Given two chromosomes

10001001110010010
01010001001000011

Choose a random bit along the length, say at position 9, and swap all the bits after that point

So the above become:

10001001101000011
01010001010010010

WHAT'S THE MUTATION RATE?

This is the chance that a bit within a chromosome will be flipped (0 becomes 1, 1 becomes 0 - this operation is called digit inversion. We give an example of inversion realization on low level programming language assembly -appendix 1[12-14]). This is usually a very low value for binary encoded genes, say 0.001

So whenever chromosomes are chosen from the population the algorithm first checks to see if crossover should be applied and then the algorithm iterates down the length of each chromosome mutating the bits if applicable.

PRACTIC IMPLEMENTATION

To hammer home the theory you've just learnt let's look at a simple problem:
Given the digits 0 through 9 and the operators +, -, * and /, find a sequence that will represent a given target number. The operators will be applied sequentially from left to right as you read.
So, given the target number 23, the sequence 6+5*4/2+1 would be one possible solution.

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

If 75.5 is the chosen number then 5/2+9*7-5 would be a possible solution.
Please make sure you understand the problem before moving on. I know it's a little contrived but I've used it because it's very simple.

Stage 1: Encoding

First we need to encode a possible solution as a string of bits… a chromosome. So how do we do this? Well, first we need to represent all the different characters available to the solution... that is 0 through 9 and +, -, * and /. This will represent a gene. Each chromosome will be made up of several genes.

Four bits are required to represent the range of characters used:

0: 0000
1: 0001
2: 0010
3: 0011
4: 0100
5: 0101
6: 0110
7: 0111
8: 1000
9: 1001
+: 1010
-: 1011
*: 1100
/: 1101

The above show all the different genes required to encode the problem as described. The possible genes 1110 & 1111 will remain unused and will be ignored by the algorithm if encountered.

So now you can see that the solution mentioned above for 23, ' 6+5*4/2+1' would be represented by nine genes like so:

0110 1010 0101 1100 0100 1101 0010 1010 0001
6 + 5 * 4 / 2 + 1

These genes are all strung together to form the chromosome:

011010100101110001001101001010100001

Because the algorithm deals with random arrangements of bits it is often going to come across a string of bits like this:

001000101010111010110111 0010

Decoded, these bits represent:

0010 0010 1010 1110 1011 0111 0010
2 2 + n/a - 7 2

Which is meaningless in the context of this problem! Therefore, when decoding, the algorithm will just ignore any genes which don't conform to the expected pattern of: number -> operator -> number -> operator …and so on. With this in mind the above 'nonsense' chromosome is read (and tested) as:
2 + 7

Stage 2: Deciding on a Fitness Function

This can be the most difficult part of the algorithm to figure out. It really depends on what problem you are trying to solve but the general idea is to give a higher fitness score the closer a chromosome comes to solving the problem. With regards to the simple project I'm describing here, a fitness score can be assigned that's inversely proportional to the difference between the solution and the value a decoded chromosome represents.

If we assume the target number for the remainder of the tutorial is 42, the chromosome mentioned above

ISSN: 2277-9655
Impact Factor: 4.116
CODEN: IJESS7

01101010010111000100110100101010000 1

has a fitness score of 1/(42-23) or 1/19.

As it stands, if a solution is found, a divide by zero error would occur as the fitness would be 1/(42-42). This is not a problem however as we have found what we were looking for... a solution. Therefore a test can be made for this occurrence and the algorithm halted accordingly.

Stage 3: Getting down to business
Please tinker around with the mutation rate, crossover rate, size of chromosome etc to get a feel for how each parameter effects the algorithm. Hopefully the code should be documented well enough for you to follow what is going on!

Note: The code given will parse a chromosome bit string into the values we have discussed and it will attempt to find a solution which uses all the valid symbols it has found. Therefore if the target is 42, + 6 * 7 / 2 would not give a positive result even though the first four symbols("+ 6 * 7") do give a valid solution.

CONCLUSIONS

There are different selection techniques to use, different crossover and mutation operators to try and more esoteric stuff like fitness sharing and speciation to fool around with. All or some of these techniques will improve the performance of your genetic algorithms considerably.

There are different ways of selecting the individuals for the next generation. Given the constraint that the number of individuals should remain constant some individuals have to be discarded. Transition between generations can be done by total replacement, elitist replacement or steady state replacement. For total replacement only the newly created offspring enter the next generation and the parents of the previous generation are completely discarded. This has the disadvantage that a fit parent can be lost even if it only produces bad offspring once. With elitist replacement all parents and offspring of one generation are sorted according to their fitness. If the size of the population is n, then the n fittest individuals are selected as parents for the following generation. This mode has been used here. Another variant is steady state replacement where two individuals are selected from the population based on their fitness and then modified by mutation and crossover. They are then used to replace their parents**.**

REFERENCES

[1] L. Davis, (ed.) Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.
[2] Fleming and Purshouse 2002, Genetic algorithms in control systems engineering p. 1228.
[3] Mitchell, C.A., Davies, M.J., Grounds, M.D., McGeachie, J.K., Crawford, G.J., Hong, Y., and Chirila, T. (1996) J. Biomat. Appl., 10, 230-249.
[4] Schulze-Kremer, Steffen. Molecular bioinformatics: algorithms and applications. Walter de Gruyter& Berlin&New York, 1996, pp. 262-267.
[5] J. T. Ngo, J. Marks, Computational complexity of a problem in molecular-structure prediction, Protein Engineering, vol 5, no 4, pp. 313-321, 1992.
[6] L. Davis, (ed.) Handbook of Genetic Algorithms, van Nostrand Reinhold, New York, 1991.
[7] C. B. Lucasius, G. Kateman, Application of Genetic Algorithms to Chemometrics, Proceedings 3rd International Conference on Genetic Algorithms, (J. D. Schaffer, ed.), Morgan Kaufmann Publishers, San Mateo, CA, pp. 170-176, 1989.
[8] P. Tuffery, C. Etchebest, S. Hazout, R. Lavery, A new approach to the rapid determination of protein side chain conformations, J. Biomol. Struct.Dyn, vol 8, no 6, pp. 1267-1289, 1991.
[9] Painter TS (1933). "A new method for the study of chromosome rearrangements and the plotting of chromosome maps". Science 78 (2034): 585–586. doi:10.1126/science.78.2034.585. PMID 17801695.
[10] Gardner R.J.M. and Sutherland G.R. 2004. Chromosome abnormalities and genetic counseling. Oxford.
[11] Lehtonen S, Myllys L, Huttunen S (2009). "Phylogenetic analysis of non-coding plastid DNA in the presence of short inversions". Phytotaxa 1: 3–20
[12] Hyde, Randall. "Chapter 12 – Classes and Objects". The Art of Assembly Language, 2nd Edition. No Starch Press. © 2010
[13] Evans, David. "x86 Assembly Guide". University of Virginia, 2010.
[14] Salomon, David. Assemblers and Loaders (pdf). p. 67. 2012.