Applications Of Genetic Algorithm

Supervised by:

Dr. A. Chatterjee

Made by:

GROUP- 2

Devesh Garg	Y8191
Hemendra Goyal	Y8215
Utsav Kumar	Y8546
Vijesh Bhute	Y8561

Content:

INTRODUCTION

Genetic Algorithm has been developed by John Holland, his colleagues and his students at the University of Michigan in 1975. The book named "Adaptation in Natural and Artificial Systems" was the first by them that discusses GA.

Genetic Algorithms are search algorithm based on mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human touch. In every new generation a new set of strings is created using bits and pieces of fittest of the old. While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance.

The Central theme of research on genetic algorithms has been robustness the balance between efficiency and efficacy necessary for survival in many different environments. Genetic Algorithms are theoretically and empirically proven to provide robust search in complex spaces. These algorithms are computationally simple and yet powerful in their search for improvement.

The key features of GAs are:

1. GAs work with a coding of the parameter set and not the parameter themselves.
2. GAs search from a population of points and not from a single point.
3. GAs use objective function information and not derivatives or other auxiliary knowledge.
4. GAs are probabilistic transition rules and not deterministic rules.

Algorithm

There are four major steps in which Genetic Algorithm works

- Selection
- Reproduction
- Crossover
- Mutation

Stopping Criteria: We need to provide a stopping criteria like the minimum final tolerance in the function or the total time the GA runs or the total number of generations.

In our main problem statement, we have provided both minimum tolerance(=10^-6) and the maximum number of generations to be 400.

For the purpose of explaining the different parts of GA we have considered a problem statement and written code to explain different parts of the GA.

We have considered a simple problem in order to explain functioning of GA in optimization, i.e.

x(1)^2-x(2)^2-5.13

and the fitness function considered is……..

fitness function =*1/abs((x(1)^2-x(2)^2-5.13))*

INITIAL POPULATION

Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, covering the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

To generate population of 10 strings and take transpose so that each column represent the string that we have randomly chosen.

```
function y=generation()
x1=[];
for i=1:10
   x1=[x1;generate()];%8cross10
end
x1=transpose(x1);%10cross8
```

Where generate function is…

```
function y=generate()
for j=1:8
  if(rand>0.5)
```

```
    x1(j)=1;
  else
    x1(j)=0;
  end
end
y=x1;
end
```

It gives us a string of eight bit and assigning either 0 or 1 to it..

```
 initial_generation =

   1   1   0   1   1   0   1   0
   0   1   1   1   1   0   1   1
   1   1   1   0   1   0   1   0
   0   0   0   1   1   0   1   0
   0   0   1   1   0   0   0   1
   1   1   0   1   1   0   0   0
   1   0   1   0   1   0   1   1
   1   1   1   0   0   0   1   0
   1   0   1   0   0   0   1   0
   0   1   1   1   1   0   1   1
```

SELECTION

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.

Most functions are STOCHASTIC and designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large, preventing premature convergence on poor solutions. Some of the SELECTION methods are as follows:

Roulette Wheel Selection

It works on the concept of the game Roulette. Under this game each individual gets a slice of the wheel, but more fit ones get larger slices than less fit ones. The wheel is then spun, and whichever individual "owns" the section on which it lands each time is chosen. A form of fitness-proportionate selection in which the chance of an individual's being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness.

Scaling Selection

As the average fitness of the population increases, the strength of the selective pressure also increases and the fitness function becomes more discriminating. This method can be helpful in making the best selection later on when all individuals have relatively high fitness and only small differences in fitness distinguish one from another.

Tournament Selection

Subgroups of individuals are chosen from the larger population, and members of each subgroup compete against each other. Only one individual from each subgroup is chosen to reproduce.

Rank Selection

Each individual in the population is assigned a numerical rank based on fitness, and selection is based on these ranking rather than absolute differences in fitness. The advantage of this method is that it can prevent very fit individuals from gaining dominance early at the expense of less fit ones, which would reduce the population's genetic diversity and might hinder attempts to find an acceptable solution.

Hierarchical Selection

Individuals go through multiple rounds of selection each generation. Lower-level evaluations are faster and less discriminating, while those that survive to higher levels are evaluated more rigorously. The advantage of this method is that it reduces overall computation time by using faster, less selective evaluation to weed out the majority of individuals that show little or no promise, and only subjecting those who survive this initial test to more rigorous and more computationally expensive fitness evaluation.

REPRODUCTION

Reproduction is a process in which individual strings are copied according to their objective function values that is f. We can call this function as Fitness Function.

In other way around we can think this as some measure of profit, utility or goodness that we want to maximize. Here fitness values of string signifies that

Strings with a higher value of would have higher probability of contributing one or more offspring to next generation . This operator is an artificial version of natural selection. The reproduction operator may be implemented in algorithmic form in a number of ways . We have used the concept of Roulette wheel here, where each current string in the population has a roulette wheel slot size in the proportion to its fitness.

Each time we require another offspring , a simple spin of the weighted roulette wheel yields the reproductive candidate. In this way, more highly fit springs have a higher number of offspring in the succeeding generation . Once a string has been selected for replica this string is then entered in to a mating pool , a tentative new population for further operator action.

Program for selection …

Here x1 is a 10 cross 8 array. Where each column is the string that we have randomly generated. Convert is a function that converts the binary to decimal by breaking it into two 4 bit string and calculating its decimal value. So our range of solution space is from 0 to 15.

```
%binary is 2cross1
binary1=convert(x1(1:8));
binary2=convert(x1(9:16));
binary3=convert(x1(17:24));
binary4=convert(x1(25:32));
binary5=convert(x1(33:40));
binary6=convert(x1(41:48));
binary7=convert(x1(49:56));
binary8=convert(x1(57:64));
binary9=convert(x1(65:72));
binary10=convert(x1(73:80));
%main gives scalar
```

Fitness is calculated by passing the values to the main function where values are inserted in actual equation ,error is calculated and so the fitness value, which is returned back.

```
fitness1=[main(binary1); main(binary2);main(binary3); main(binary4); main(binary5); main(binary6);
main(binary7); main(binary8); main(binary9); main(binary10)];
```

Roulette Selection is used to select the string which will go for the Reproduction . So, probability of each of the 10 strings are calculated that will undergo Reproduction on the basis of their fitness value. Using Random value and passing it down to different sets of condition we find the string which will undergo Reproduction. Also, those strings which are selected for Reproduction are stored in matrix named next.

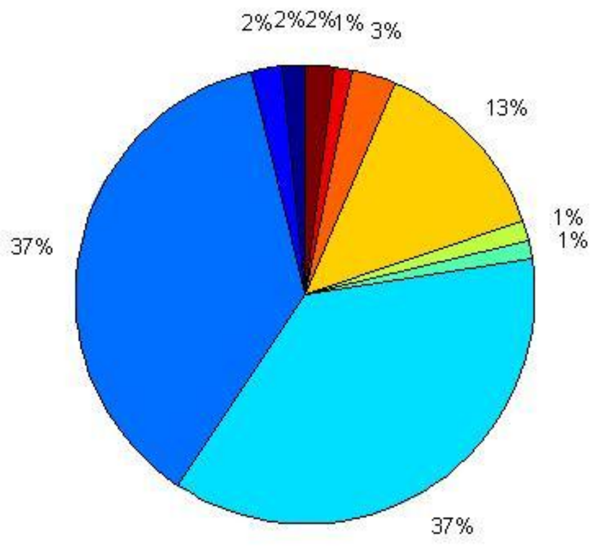
```
sum=0;
for i=1:10
   sum=sum+fitness1(i);
end
sum
probability=[];%10cross1
probability(1)=fitness1(1)/sum;
for i=2:10
   probability(i)=probability(i-1)+fitness1(i)/sum;
end
next=[];%8cross10 after selection
for i=1:10
  r=rand;
   if(rand<probability(1))
          next=[next;x1(1:8)];
   elseif(rand<probability(2))
          next=[next;x1(9:16)];
   elseif(rand<probability(3))
          next=[next;x1(17:24)];
   elseif(rand<probability(4))
          next=[next;x1(25:32)];
   elseif(rand<probability(5))
          next=[next;x1(33:40)];
   elseif(rand<probability(6))
          next=[next;x1(41:48)];
```

```
    elseif(rand<probability(7))
        next=[next;x1(49:56)];
    elseif(rand<probability(8))
        next=[next;x1(57:64)];
    elseif(rand<probability(9))
        next=[next;x1(65:72)];
    elseif(rand<probability(10))
        next=[next;x1(73:80)];
    end
end
initial_generation=transpose(x1);%8 cross 10  // Row represent string
after_selection=transpose(next);%10 cross 8   // Column represent string

after_select=next;   // Finally String undergoing Reproduction is stored
```

```
after_select =

     1     1     1     0     1     0     1     0
     0     0     1     1     0     0     0     1
     0     0     1     1     0     0     0     1
     0     0     1     1     0     0     0     1
     0     0     1     1     0     0     0     1
     1     1     0     1     1     0     0     0
     0     0     1     1     0     0     0     1
     0     0     1     1     0     0     0     1
     1     1     0     1     1     0     0     0
     1     1     1     0     1     0     1     0
```



Pie chart representing the probability distribution in the Roulette Selection.

CROSSOVER

After reproduction, simple crossover may proceed in two steps. First each member of the newly reproduced strings in the mating pool is mated at random. Second, each pair of strings undergoes crossing over as follows. An integer position k along the string is selected uniformly at random between 1 and the string length less one [1 , l – 1]. Two new strings are created by swapping all characters between k+1 and l inclusively.

Here we crossover the selected strings using the above formulae and rest of the strings are directly inherited in the matrix.

```
%let crossover probability be 80%
cross=[];%8crossn stores the strings selected for the crossover
count=0;%counts the length of crossover strings
set=[];%takes care of the indexes of the strings not chosen for crossover
for i=1:10
   if(rand<=0.8)
      cross=[cross;after_selection((8*i-7):8*i)];
      count=count+1;
   else
      set=[set;i];
   end
end
rest=[];
for i=1:length(set)
   rest=[rest;after_selection((8*set(i)-7):8*set(i))];
end
%rest
%cross_selected=transpose(cross);
%cross
tran_cross=transpose(cross);%ncross8
count1=[0 0 0 0 0];
%CROSSOVER
comb1=[];
c1=greatest_integer(count/2);
if(c1<count/2)
   comb1=[tran_cross((8*count-7):8*count)];
end
for i=1:2:(count-1)
   r=rand*8;
   great_int=greatest_integer(r);
   temp1=tran_cross((8*i-7):(8*(i-1)+great_int));
   temp2=tran_cross((8*i-7+great_int):8*i);
   temp3=tran_cross((8*(i+1)-7):(8*i+great_int));
   temp4=tran_cross((8*(i+1)-7+great_int):8*(i+1));
   crossover_d11=[temp1 temp4];
   crossover_d12=[temp3 temp2];
   comb1=[comb1;crossover_d11;crossover_d12];
end
after_crossover=[rest;comb1];
```

Where greatest_integer function is…

```
function g=greatest_integer(A)
g=A-mod(A,1);
end
```

```
after_crossover =

   0   0   1   1   0   0   0   1
   0   0   1   1   0   0   0   1
   1   1   1   1   0   0   0   1
   0   0   1   0   1   0   1   0
   0   0   1   1   0   0   0   1
   0   0   1   1   0   0   0   1
   0   0   0   1   1   0   0   0
   1   1   1   1   0   0   0   1
   1   1   1   0   1   0   1   0
   1   1   0   1   1   0   0   0
```

MUTATION

Mutation adds new information in a random way to the genetic search process and ultimately helps to avoid getting trapped at local optima.

Mutation in a way is the process of randomly disturbing genetic information. They operate at the bit level; when the bits are being copied from the current string to the new string, there is probability that each bit may become mutated. This probability is usually a quite small value, called as mutation probability. A coin toss mechanism is employed; if random number between zero and one is less than the mutation probability, then the bit is inverted, so that zero becomes one and one becomes zero. This helps in introducing a bit of diversity to the population by scattering the occasional points. This random scattering would result in a better optima, or even modify a part of genetic code that will be beneficial in later operations. On the other hand, it might produce a weak individual that will never be selected for further operations.

Mutation value is taken as 0.01 and using the random value we change the value of the binary digit if probability allows to do so.

```
%MUTATION
changes=0;
for i=1:length(after_crossover)
   if(rand<0.01)
      after_crossover(i)=invert(after_crossover(i));
      changes=changes+1;
   end
end
after_mutation=after_crossover;
initial_generation
after_select
```

```
after_crossover
after_mutation
changes%gives number of changes
initial_fitness=sum;
initial_fitness
final_fitness=fitness(after_mutation);
final_fitness
end
```

```
after_mutation =

     0     0     1     1     0     0     0     1
     0     0     1     1     0     0     0     1
     1     1     1     1     0     0     0     1
     0     0     1     0     1     0     1     0
     0     0     1     1     0     0     0     1
     0     0     1     1     0     0     0     1
     0     0     0     1     1     0     0     0
     1     1     1     1     0     0     0     1
     1     1     1     0     1     0     1     0
     1     1     0     1     1     0     0     0


changes = 0

initial_fitness =  0.4753
final_fitness = 1.4484
```

MAIN PROBLEM STATEMENT

The main problem statement which we are going to present is taken from the research paper "The application of Genetic Algorithm (GA) to estimate the rate parameters for solid state reduction of iron ore in presence of graphite". GA's have found their application in many diverse fields like physics, astronomy, finance, chemistry, etc. Here, we have applied GA to estimate the various rate parameters using experimental data and compared the results with those values presented in the literature. The reduction of hematite to iron has been considered to occur in three sequential steps namely as:

(i) Hematite (Fe_2O_3) to Magnetite (Fe_3O_4),
$3Fe_2O_3 + CO = 2Fe_3O_4 + CO_2$

(ii) Magnetite (Fe_3O_4) to Wustite (FeO),
$Fe_3O_4 + CO = 3FeO + CO_2$

(iii) Wustite (FeO) to Iron (Fe),
$FeO + CO = Fe + CO_2$

Mass balance equations were assumed to follow the first order kinetics.

$$\frac{dH}{dt} = -H * kh * \exp\left(-\frac{Eh}{RT}\right)$$
$$\frac{dM}{dt} = 0.97 * H * kh * \exp\left(-\frac{Eh}{RT}\right) - M * km * \exp\left(-\frac{Em}{RT}\right)$$
$$\frac{dW}{dt} = 0.93 * \ M * km * \exp\left(-\frac{Em}{RT}\right) - W * kw * \exp\left(-\frac{Ew}{RT}\right)$$
$$\frac{dF}{dt} = 0.78 * \ W * kw * \exp\left(-\frac{Ew}{RT}\right)$$

Where, H,M,W represent the concentrations of hematite, magnetite and wustite at time t respectively.

The unknown parameters are three rate constants: kh, km, kw and activation energies, Eh, Em and Ew which we are going to find using GA and experimental data. The parameter which is measured using the experiment is defined as the degree of reduction coefficient which is found by using the formula : $\alpha = (4Z + 8) * \frac{\Delta W}{(0.3 * Wh)}$, where Wh is the total weight of hematite at time t, ΔW is the loss in weight of the packed bed and Z is the exit gas composition, i.e., CO/CO_2 ratio.

The predicted values of degree of reduction have been obtained from the respective rates of hematite, magnetite, and wustite, as follows:

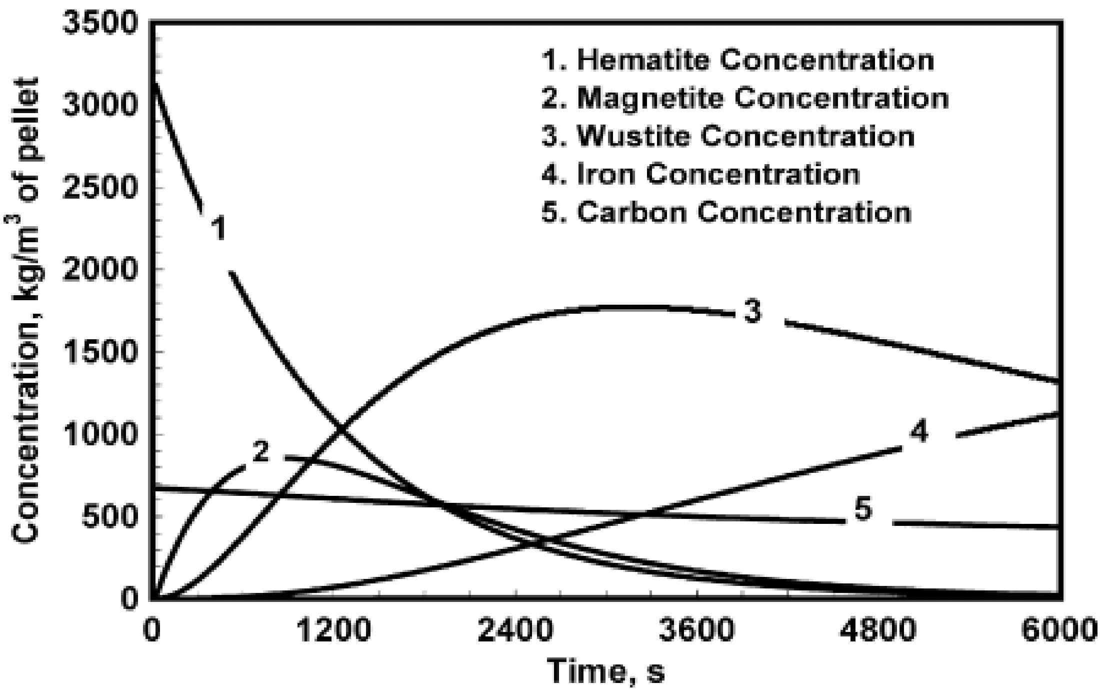
$$\alpha 1 = \frac{S0 + \left(0.033 * \left(\frac{dH}{dt}\right) + 0.068 * \left(\frac{dM}{dt}\right) + 0.222 * \left(\frac{dW}{dt}\right)\right) * \Delta t}{936}$$

Where, S0 gives the total concentration of oxygen consumed in time Δt, and 936 kg/m^3 is the total removable oxygen. H, M and W can be found from the experimental graph of the concentration versus time graph. GA has been used to find the unknown parameters by minimizing the error between the experimental and predicted values of the degree of reduction. We have used a double vector to generate the populations. GA parameters used by us are:

- Fitness function: 1/($\alpha - \alpha 1$)
- Generations: 400
- Mutation Probability: 0.01
- Cross over probability: 0.8
- Selection technique: Tournament selection
- Range at 1000^{o}C: for kh (s^{-1}), km(s^{-1}), kw(s^{-1}), Eh (KJ/mol), Em (KJ/mol), Ew (KJ/mol) is resp:

  [10^14,10^18], [10^12,10^17], [10^10,10^12], [100,600], [150,600], [175,600]

- The total time chosen is 4800 s. and the initial values of H, M and W are 3100, 0 and 0 respectively (from graph):



Evaluation of concentration of various iron oxide phases as well as pure iron during packed bed reduction of iron ore – graphite composite pellets under argon atmosphere at 1000^{0}C

We made the following code to write the fitness function using MATLAB:

```
function z=solve2(y)
b1=y(1)*exp(-y(4)/10583.722);
b2=y(2)*exp(-y(5)/10583.722);
b3=y(3)*exp(-y(6)/10583.722);
h=3100*exp(-b1*1300);
m=0.97*3100*b1*(exp(-b1*4800)-exp(-b2*4800))/(b2-b1);
w=0.93*b2*0.97*3100*((exp(-b1*4800)-exp(-b3*4800))/(b3-b1)-(exp(-b2*4800)-exp(-b3*4800))/(b3-b2))/(b2-
b1);
ht=-h*b1;
mt=0.97*h*b1-m*b2;
wt=0.93*m*b2-w*b3;
S0=0.033*(h-3100)+0.068*m+0.222*w;
z=1/(0.62-((S0+(0.033*ht+0.068*mt+0.222*wt)*4800)/936));
end
```

The code was run using the in-built function of MATLAB – ga. The GUI interface was used and the conditions were inserted and the code was run for 400 generations. The results found were:

	Kh (s^{-1})	**Km (s^{-1})**	**Kw (s^{-1})**	**Eh (KJ/mol)**	**Em (KJ/mol)**	**Ew (KJ/mol)**
GA	3.944*10^17	9.705*10^16	3.973*10^11	188.1611	393.6013	321.0304
literature	6.00*10^17	7.50*10^16	1.70*10^11	380	410	330

The results show quite a good agreement despite of several minor changes in the problem as mentioned below.
Sources of errors in our calculation are:

- The mutation probabilities are not the same as used in the paper.
- The conditions mentioned in the problem is slightly varied like the elitism is not considered, the range is not the same as in the paper, etc.

Applications of Genetic Algorithm

1. Acoustics- GA is used to distinguish between sonar reflections and different types of object. Apart from that it is also used to design Active noise control systems which cancel out undesired sound by producing sound waves that destructively interfere the unwanted noise.

2. Aerospace Engineering- To design wing shape Super Sonic aircraft minimizing aerodynamic drag at supersonic cruising speeds, minimizing drag at subsonic speeds, and minimizing aerodynamic load .

3. Financial Markets – to predict the future performance of publicly traded stocks.

4. Geophysics-locate earthquake hypocenters based on seismological data.

5. Materials Engineering-To design electrically conductive carbon based polymers known as polyanilines. To design exposure pattern for an electron lithography beam.

6. Routing and Scheduling- To find optimal routing paths in telecommunication networks which are used to relay data from sender to recipients .

7. Systems Engineering- To perform multi–objective task of designing wind turbines used to generate electric power.

8. Data mining- GA can also be used in data mining and pattern recognition. Given a large set of data, GA can perform a search for the optimum set of data which satisfies the conditions. Initial population in this case may be single objective conditions and at the end of the GA, we get a combined complex condition which when applied on the large data set can lead to finding the required set of data.

Strengths of GAs

1. The main strength of GA is that it is intrinsically parallel. Whereas most other algorithms are serial and search the solution space to a problem in one direction at a time. So, in such case if solution turns out to be suboptimal then we have to abandon all work previously and start over. However since GA has multiple offspring so they can explore the solution space in multiple direction at once given the condition that some of them would turn out as dead end.

2. GA is better in solving problems where the space of all potential solution is truly huge. Non Linear functions falls into this category, where changing one component may have ripple effects on the entire system and where multiple changes that individually are detrimental may lead to much greater improvements in fitness function.

3. GA performs well in problems for which fitness function is complex or discontinuous or noisy or changes over time or has many local optima. Whereas other search algorithms can become trapped by local optima but GA works well to avoid local optima.

4. One of the main features of GA is its ability to manipulate many parameters simultaneously. Many problems cannot be stated as single value to be maximized or minimized but expressed as multiple objectives.

5. GAs know nothing about the problem that they are deployed to solve. The virtue of this technique is that it allows GA to start out with open mind. Since in GA decisions are based on randomness all possible search pathways are theoretically open to GA.

Other Optimizing Techniques

Different Data Mining Techniques are

1. Neural Networks or pattern Recognition
2. Memory based reasoning
3. Cluster Detection or Market Basket Analysis
4. Link Analysis
5. Visualization
6. Decision Tree or Rule Induction
7. Simulated Anealing
8. Hill Climbing

Neural Network and Genetic Algorithm

Neural networks (NN) fall into the category of *Supervised Models*. That is, our data will be a set of rows, where each row contains an input and a *corresponding output* for that input. Our NN learns by seeing the difference between the correct output and one it predicted, and then adjusting its parameters. So one can't use NN if he/she don't have input-ouput data .

Examples: Graph extrapolation. Facial recognition.

Genetic algorithms (GA) are basically optimizers. Here we will have some set of parameters that we want to optimize for something. We will need an evaluation function that takes these parameters and tells us how good these parameters are. So we keep changing these parameters somehow until we get an acceptable value from our evaluation function, or until we see that things are not improving any more.

Examples: Scheduling airplanes/shipping. Timetables . Finding the best characteristics for a simple agent in an artificial environment. Rendering an approximation of a picture with random polygons.

If we have data that is suitable for supervising a model, then we can use a NN. If we want to optimize some parameters, then use a GA. But most importantly, it is the nature of our data and what we want out of it that should decide what model to use.

Simulated annealing:

Another optimization technique similar to evolutionary algorithms is known as simulated annealing. The idea borrows its name from the industrial process of *annealing* in which a material is heated to above a critical point to soften it, then gradually cooled in order to erase defects in its crystalline structure, producing a more stable and regular lattice arrangement of atoms. In simulated annealing, as in genetic algorithms, there is a fitness function that defines a fitness landscape; however, rather than a population of candidates as in GAs, there is only one candidate solution. Simulated annealing also adds the concept of "temperature", a global numerical quantity which gradually decreases over time. At each step of the algorithm, the solution mutates (which is equivalent to moving to an adjacent point of the fitness landscape).

The fitness of the new solution is then compared to the fitness of the previous solution; if it is higher, the new solution is kept. Otherwise, the algorithm makes a decision whether to keep or discard it based on temperature. If the temperature is high, as it is initially, even changes that cause significant decreases in fitness may be kept and used as the basis for the next round of the algorithm, but as temperature decreases, the algorithm becomes more and more inclined to only accept fitness-increasing changes. Finally, the temperature reaches zero and the system "freezes"; whatever configuration it is in at that point becomes the solution. Simulated annealing is often used for engineering design applications such as determining the physical layout of components on a computer chip.

Important Problems of GA:

There are several problem related to using GA but we would like to stress on 4 major problems of GA:

1. Difficulty in population selection,
2. How to define Fitness Function,
3. Premature or rapid convergence of GA
4. Converge to local optima inspite of global optima

How we can overcome them is always a big question.

Population Selection Problem:

Population selection is a big question but different selections for different problem have been found out. Like for any case of optimization problem in which we have n independent variables and all have certain constraint over them like they are bounded in between the given values. In such problems first of all we are supposed to find out how much accuracy would be required and then if we go for Bit String representation then the corresponding population will be decided by

$2^{m-1} < (b-a)*\text{decimal order accuracy} < 2^{m}$

Defining Fitness Function:

Generally GA Fitness Function value increases with the iteration so we have to define our Fitness Function in such a way so that it increases even with searching for minima or maxima. Like in our main Problem we were supposed to find out the optimal minimum value so we defined our fitness function in such a way so that optimal solution fitness can increase with every iteration.

We defined it like

1/(different of fitness of particular individual to total fitness value)

So we can always find a way to define our fitness function in such a way so that we fit it into the GA.

Rapid Convergence of GA:

Rapid convergence is a very common as well as very general problem in GA. It occurs because the Fitness function changes its value rapidly and that's what results in a premature convergence There are several method to modify our Fitness function in such a way so that Convergence occur slowly which will allow enough time for GA to search in whole space and find the global optima.2 ways are defined below.

1. $F' = a*F + b$;

   F is normal Fitness value of the population String.

Here a is a small no while b is relatively larger number now it can be clearly seen that even if F rapidly increases then after F' will increase slowly.

2. $F' = a*F^k$, this is called power law

Here k is kept low so that even if F increases rapidly it may not create rapid Convergence . There is slightly advanced way to define Fitness function in which Fitness Function is modified in each and every generation.

So,

k = func(t) , t is generation

So in this case Fitness Function is modified in each and every case.

Convergence to Local Optima:

Another important problem is that most of the times GA converges to local optima. We may very well think that we have found the optimal solution but actually we didn't.

Now, it is important to note that GA ability to find the optimal solution lies in the hand of the user. Optimal solution will be achieved only if programmer has written the code so that GA has the ability to search over whole Space to find the optimal solution. So what we should do in such case is that we should modify our crossover and mutation function because they are responsible for changing the population in each and every iteration.

Why does GA work?

There is a concept of Schema defined as

*10101100101

Where * is called as don't care symbol and can be either 1 or 0. So the above schema matches with the string shown below:

110101100101
010101100101

So from the above we can easily see that 1010100011 represent only one string but ********** represents all the string of length 10.

Also a single string (1010100011) is matched by 2^{10} Schema.

1010100011
*010100011
1*10100011

So on…..

Now there are 2 important schema properties that should be noted here:

- Order of the Schema (denoted by o(S)) : it is defined as the total number of 0's and 1's in the Schema

- Defining length of Schema(denoted by Ω(S)): is defined as distance between the 1st and the last fixed string position

For Example –

S1 = ***001*110
S2 = 11101**001

So,
o(S1) = 6, o(S2) = 8 ;
Ω(S1) = 10-4 = 6, Ω(S2) = 10-1 = 9

There are two important things to note here
- The notion of the order of a Schema is useful in calculating survival probabilities of the of the Schema for Mutation.
- The notion of the defining length of a Schema is useful in calculating survival probabilities of the Schema for Crossover.

Now as discussed earlier the GA program consists of four consecutive repeated steps:
t ⟵ t+1

Select P(t) from P(t-1) and recombine P(t). Evaluate P(t) for which we will require some new terms

£(S,t) : it is defined as the number of string in the population which matches with the Schema at time t.

for example

S = ****111***********************
o(S) = 3
Ω(S) = 2

If this is the population

X1 = 111101110000001010101010100
X2 = 011100110010100101010010101
X3 = 010101001010111111111111000
X4 = 000000111111111111000000011

X5   = 10110110000000011010101001101
X6   = 101011111101000000001111111111
X7   = 00000000011111111101101010010
X8   = 010101111111000000000111111111
X9   = 101010100101000001111111111000
X10 = 10101010100100101001010010101
X11 = 100001111010101011111111100000
X12 = 101010111111111111111111111111
X13 = 101011110000000010101001010101
X14 = 100010000000000000011111111111
X15 = 101011111111111110000000000000
X16 = 1000000001010101001010101001001
X17 = 10001110000000010101000010000
X18 = 1000000000000000000000000000000
X19 = 1111000011110101001010101001000
X20 = 000000000000000000011111111111

Then in the above population X13 , X15 ,X17 are the 3 strings that are matched with the Schema.

So £(S,t) = 3

eval (S,t) is defined as the average fitness of the String in the population which matches with the Schema ,i.e

eval(S,t) = $\sum_{i=1}^{P}$ eval(Xi)/P

Xi are the String which matches with the Schema.
where P is the number of the String which matches with the Schema.
The expected number of String which will match in the next iteration with the Schema is given by the relation below:

£(S,t+1) = £(S,t).pop_size .eval(S,t) /F(S,t)

Or

£(S,t+1) = £(S,t) **.**eval(S,t) /$\underline{F(S,t)}$

Here pop_size is the population size and F(S,t) is the sum of the fitness of all the Strings.
Where $\underline{F(S,t)}$ is the average fitness of the entire population.

Now from the above relation it is clear that “above average” Schema receive an increasing number of String in the next generation and “below average” Schema receive decreasing number of String in the next generation.

Now if we assume that Schema S remain above average that means

$\text{eval}(S,t) = \underline{F(S,t)} + \alpha.\underline{F(S,t)}$

So,
$£(S,t) = £(S,0)(1 + \alpha)^{t}$

Now we will consider the effect of **Crossover and Mutation**.
So, first consider Crossover

Consider 2 Strings:

S1 = ***111**************************
S2 = 111***************************01

It can be clearly seen that there are relatively higher chances that S1 will survive after Crossover but on the other hand it is almost sure that the S2 will be destructed until unless it is Crossover by itself or similar kind of Schema.

In general a crossover site is selected uniformly among m-1 possible sites. This implies that that the probability of destruction of Schema S is:

$p_{d}(S) = \Omega(S)/(m-1)$

So probability of survival of single String is

$p_{s}(S) = 1 - \Omega(S)/(m-1)$
$p_{s}(S1) = 30/32$, $p_{s}(S2) = 0$

So,

$p_{s}(S) = 1 - p_{c}(S).\ \Omega(S)/(m-1)$

where $p_{c}(S)$ is the Crossover probability
in fact,

$p_{s}(S) \geq 1 - p_{c}(S).\ \Omega(S)/(m-1)$

So, the modified number of expected number of String which will match in the next iteration with the Schema is given by:

$£(S,t+1) = (£(S,t)\ .\text{eval}(S,t)\ /\underline{F(S,t))}.[\ 1 - p_{c}(S).\ \Omega(S)/(m-1)]$

Now we will consider Mutation
Since the the probability of alteration of single bit is p_{m} the probability of survival of bit is 1-p_{m} . So

$p_{s}(S) = (1 - p_{m})^{o(S)}$
since $p_{m} << 1$. So
$p_{s}(S) \sim 1 - o(S).P_{m}$

hence,

$£(S,t+1) = (£(S,t) . eval(S,t) / \underline{F(S,t)}) . [1 - p_c(S) . \Omega(S)/(m-1) - o(S) . p_m]$
$£(S,t+1) = £(S,0) . [eval(S,t) / \underline{F(S,t)} . [1 - p_c(S) . \Omega(S)/(m-1) - o(S) . p_m]]^t$

So this is the final expected number of String which will be same as that of Schema.
If $[eval(S,t) / \underline{F(S,t)} . [1 - p_c(S) . \Omega(S)/(m-1) - o(S) . p_m]] > 1$

Then, the number of String similar to Schema will increase exponentially otherwise decrease exponentially. So this is how GA discriminate between the above average Schema and below average Schema.

Conclusion:

We have tried to explore the Genetic Algorithm as an optimizing technique and studied its advantages over the other optimizing techniques. The comparison was made with other optimization techniques and we found that GA due to its stochastic nature, parallelism and natural selection technique is better than others in many ways. GA's can be applied to complex non-linear functions and problems involving huge solution space search where other optimizing techniques may not work. For simple problems, GA may be quite expensive with respect to time.

We also explained the algorithm with the help of an example showing all major steps of GA including reproduction and selection, cross over and mutation. We found that the total fitness of the population increases with iterations and hence, we approach to optimum values. We discussed varied applications of GA in many fields and finally, showed the use of GA in finding the rate parameters of reduction of iron ore in presence of graphite.

References:

1. Golap Md. Chowdhury, Gour G. Roy, “Application of Genetic Algorithm (GA) to estimate the rate parameters for solid state reduction of iron ore in presence of graphite”, Computational Materials Science 45 (2009) 176–180

2. Dorit Wolf and Ralf Moros, “Estimating rate constants of heterogeneous catalytic reactions without supposition of rate determining surface steps- an application of a genetic algorithm”, Chemical Engineering Science, Vol. 52, No. 7, pp-1189-1199, 1997

3. David E. Goldberg, “Genetic Algorithms in search, optimization and machine learning”

4. Adam Marczyk, “Genetic Algorithms and Evolutionary Computation”

5. www.rennard.org/alife/english/gavintrgb.html

6. www.ai-junkie.com/ga/intro/gat1.html

7. www.genetic-programming.org/

8. en.wikipedia.org/wiki/Genetic algorithm