Ijfans International Journal Of Food And Nutritional Sciences
ISSN PRINT 2319 1775 Online 2320 7876
 Journal UGC CARE Listed (Group-I) Volume 10 , Issue 12, Dec 2021

Delineating Genetic Algorithm

Rakesh Kumar Giri
Associate Professor, Department of Computer Science & Engineering, Saisha Institutions, Chennai, India
Email: rakeshgiriap@gmail.com

Abstract: A genetic algorithm (GA) is an evolutionary algorithm inspired by the natural selection and biological processes of reproduction of the fittest individual. GA is one of the most popular optimization algorithms that is currently employed in a wide range of real applications. Initially, the GA fills the population with random candidate solutions and develops the optimal solution from one generation to the next. The GA applies a set of genetic operators during the search process: selection, crossover, and mutation. This article aims to review and summarize the recent contributions to the GA research field. In addition, the definitions of the GA essential concepts are reviewed. Furthermore, the article surveys the real-life applications and roles of GA. Finally, future directions are provided to develop the field. Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. It is frequently used to solve optimization problems, in research, and in machine learning.

Keywords: Genetic Algorithm, Page Rank, Web Mining, Selection, Cross Over, Mutation

I. INTRODUCTION

Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as Evolutionary Computation. GAs was developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success. In GAs, we have a pool or a population of possible solutions to the given problem. These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations. Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield fitter individuals. This is in line with the Darwinian Theory of Survival of the Fittest. In this way we keep evolving better individuals or solutions over generations, till we reach a stopping criterion. Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.

II. BACKGROUND

1. Genetic Algorithm Definition:

Genetic algorithm (GA) is a meta heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection.Genetic

algorithms are inspired by Darwin's theory of evolution. The solution to a problem is evolved rather than designed directly. The algorithm starts with a set of solutions (represented by chromosomes) called a population. Solutions from one population are selected and used to form a new population. The idea is that the new population will be better than the old one. Solutions selected to create new solutions (offspring) are chosen according to their fitness: the more suitable they are, the greater their chance of reproducing. This process is repeated until some condition is satisfied, for example a maximum number of generations is reached or the best solution no longer improves.

2. Outline of the Basic Genetic Algorithm

1. [New population] Create a new population by repeating the following steps until the new population is complete
2. [Selection] Select two parent chromosomes from the population according to their fitness (the better the fitness, the greater the chance of being selected)
3. [Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover is performed, the offspring is an exact copy of the parents.
4. [Mutation] With a mutation probability, mutate the new offspring at each locus (position in the chromosome).
5. [Accepting] Place the new offspring in the new population
6. [Replace] Use the newly generated population for the next run of the algorithm

III. CODING

//Genetic Implementation for Data Mining

Breakdown of Headers

- <algorithm>: Provides functions for common operations like sorting, searching, and shuffling data containers.
- <iostream>: Essential for standard input and output (e.g., using std::cout and std::cin).
- <numeric>: Contains generalized numeric operations, such as std::accumulate for summing or std::iota for filling a range with sequential values.
- <random>: Offers modern facilities for generating random numbers using specific engines and distributions.
- <vector>: Defines the std::vector container, a dynamic array that can change size.

In data mining, combining Genetic Algorithms (GA) is a strategy used to optimize information retrieval and web mining. Genetic Algorithms are used to evolve the ranking parameters or refine search results to better match user queries.

1. Genetic Algorithm (GA) Integration

Genetic Algorithms mimic biological evolution (selection, crossover, and mutation) to solve optimization problems. In web mining, they are integrated with PageRank to create a Genetic PageRank Algorithm (GPRA).

- Initial Population: High-quality "seed" links or initial search results from standard

engines.

- Fitness Function: Evaluates how well a page matches a query. It often combines PageRank (link authority) with content features like keyword frequency and synonyms.
- Selection & Crossover: The best-performing pages are selected. "Crossover" in this context might involve combining different ranking factors—such as hyperlink count and keyword frequency—to generate a superior ranking score.
- Mutation: Randomly explores new or less-relevant pages to ensure the search doesn't get stuck on a single "topic drift".

```
int main()
int n=4;  //No. of data nodes
double d=0.85; //Damping Factor
vector <double> pr(n,1.0/n);
vactor<int>links={
{0,1,1,0}   // Node 0 links to 1,2
{0,0,0,1}  // Node 1 links to 3
{1,0,0,1}  //Node 2 links to 0,3
{ 0,0,0,0}  //Node 3 is a dead end (not outgoing)
};
```

Dead Ends: Node 3 has no outgoing links. In a standard PageRank model, you usually treat dead ends as if they link to every node in the graph (including themselves) to ensure the total rank remains 1.0.

//Calculate out degree for each node

The out-degree of a node is the total number of directed edges (links) pointing away from it. In both genetic regulatory networks and PageRank models, this represents the "influence" or "broadcasting" power of a node.

1.. Calculate for Genetic Networks

In genetic networks (like transcription factor networks), the out-degree represents the number of target genes a specific gene regulates.

- Gene A: If Gene A regulates Gene B and Gene C, its out-degree is 2.
- Gene B: If Gene B only regulates Gene C, its out-degree is 1.
- Gene C: If Gene C only regulates Gene A, its out-degree is 1.

2. Calculate for PageRank

In the PageRank algorithm, the out-degree is used to distribute a page's "rank" equally among all the pages it links to.

- Page 1: If Page 1 links only to Page 2, its out-degree is 1.
- Page 2: If Page 2 links to Page 1 and Page 3, its out-degree is 2.
- Page 3: If Page 3 links only back to Page 2, its out-degree is 1.

```
vector<int> outdegree (n,0);
for(int i=0; i<n; ++i){
```

```
for(int j=0;j<n; ++j){
if(links[i][j]) OutDegree[i]++;}}

//Iterative Page Rank Calculation

for(int iter=0; iter<10;++ iter){
vector<double>next_pr(n(1.0-d)/n);
for(int i=0; i<n;++i){
for(int j=0; j<n;++j){
if(links[i][j]) {
next_pr[j]+=d*(pr[i]/Out degree[i])
}
}
}
pr=next_pr;
```

FRONT END

C++ code must include “extern C” to ensure “C” linkage compatibility, which is required for Extern “C”

DEPLOYING AND PL/SQL “Call Space”

Register the library

Sql

```
CREATE OR REPLACE LIBRARY pagerank_lib  AS ‘/path/to/pagerank_lib.so’;
```

DEFINE THE PL/SQL PROCEDURE (CALL SPECIFICATION)

```
CREATE OR REPLACE PROCEDURE compute_pagerank_ext(
from_nodes  IN RAW,
to_nodes  IN RAW,
dqs  IN RAW,
num_edges  IN BINARY_INTEGER,
num_nodes  IN BINARY_INTEGER,
damping IN NUMBER,
ranks  OUT RAW
)
AS LANGUAGE C++
NAME “calculate_pagerank”
LIBRARY pagerank_lib
PARAMETERS (
```



```
from_nodes, to_nodes, dqs, num_edges, num_nodes, damping, ranks
);
/
```

IV. OPERATORS OF GA

As we can see from the genetic algorithm outline, crossover and mutation are among the most important parts of a genetic algorithm. Its performance is influenced mainly by these two operators. Before we explain crossover and mutation in more detail, it is useful to say a few things about chromosomes.

Encoding of a Chromosome

In some way, a chromosome must contain information about the solution it represents. The most commonly used encoding is a binary string. A chromosome could then look like this:

Chromosome 1	1101100100110110
Chromosome 2	1101111000011110

Each chromosome has one binary string. Each bit in this string can represent some characteristic of the solution. Or the whole string can represent a number - this is what was used in the basic GA demonstration. Of course, there are many other ways of encoding. This depends mainly on the problem being solved. For example, one can encode integer or real numbers, sometimes it is useful to encode some permutations and so on.

1. SELECTION

Selection in genetic algorithms (GA) is the process of choosing the fittest individuals from the current population to act as parents for creating the next generation. It balances exploitation of high-quality solutions with exploration of new search spaces, aiming to improve the population's overall fitness over time. Common methods include tournament, roulette wheel, and ranking selection. In genetic algorithms (GAs), selection is the phase where individual solutions (chromosomes) are chosen from the current population to breed and create offspring for the next generation. This process is inspired by the Darwinian principle of "survival of the fittest," where individuals with higher fitness scores have a greater probability of being selected, effectively guiding the algorithm toward more optimal solutions.

```
//Selection
Individual Selection(vector<individual>&pop)
{
int i=rand()% pop.size();
int j=rand()% pop.size();
return(pop[i].fitness>pop[j].fitness)?
pop[i]:pop[j];
```


2. CROSSOVER

After deciding which encoding to use, we can move on to crossover. Crossover selects genes from parent chromosomes and creates new offspring. The simplest way to do this is to choose a random crossover point, copy everything before that point from the first parent, and then copy everything after that point from the second parent.
Crossover can then look like this (| is the crossover point):

Chromosome 1 11011 | 00100110110

Chromosome 2 11011 | 11000011110

Offspring 1 11011 | 11000011110

Offspring 2 11011 | 00100110110

There are other ways to perform crossover. For example, we can choose more than one crossover point. Crossover can become quite complicated, and it depends heavily on the chromosome encoding. A crossover method designed for a specific problem can improve the performance of the genetic algorithm.

```
//Crossover

void crossover(individual p1, individual p2,individual & c1, individual & c2)
{
for(int i=0; i<GENE_SIZE; i++) {
if(i< cp >) {
c1.genes[i]=p1.genes[i];
c2.genes[i]=p2.genes[i];
}
else
c1.genes[i]=p2.genes[i];
c2.genes[i]=p1.genes[i];
}
}
}
```

2. MUTATION

After crossover is performed, mutation takes place. Its purpose is to prevent the entire population from falling into a local optimum of the problem being solved. Mutation changes the new offspring randomly. For binary encoding, we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. Mutation can then look like this:

Original offspring 1 1101111000011110

Original offspring 2 1101100100110110

Mutated offspring 1 1100111000011110

Mutated offspring 2 1101101100110100

Like crossover, mutation depends on the encoding. For example, when we are encoding permutations, mutation could mean exchanging two genes.

```
//Mutate
Void mutate( individual  & ind)
cout<< “Initial best fitness:”<<endl;

//Simple one generation evolution
individual parent1= selection(population);
individual parent2= selection(population);
individual child 1, child 2

crossover(parent 1,parent 2, child 1,child2)
mutate(child 1);
child 1.calculate Fitness();

cout<<”Child Fitness after crossover and mutation: “child1.fitness<<end l;
return 0;
}
```

3. FITNESS FUNCTION

```
// Fitness function combining Precision and Recall

Double calculate_fitness(double precision, double recall)
{
If (precision + recall==0) return 0;
return (2*precision*recall)/(precision + recall);
}
//Display Results

cout<<”Final Page rank (Data Quality Scores);”<<end l;
for(int i=0; i<n;++i){
cout <<”Node”<<end l;
}
return 0;
}
```

V. ADVANTAGES AND DISADVANTAGES

Advantages of GAs

GAs has various advantages which have made them immensely popular. These include −

- Does not require any derivative information (which may not be available for many real-world problems).
- Is faster and more efficient as compared to the traditional methods.

- Has very good parallel capabilities.
- Optimizes both continuous and discrete functions and also multi-objective problems.
- Provides a list of good solutions and not just a single solution.
- Always gets an answer to the problem, which gets better over the time.
- Useful when the search space is very large and there are a large number of parameters involved.

Limitations of GAs

Like any technique, GAs also suffer from a few limitations. These include −

- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.
- Fitness value is calculated repeatedly which might be computationally expensive for some problems.
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.
- If not implemented properly, the GA may not converge to the optimal solution.

VI. SCOPE

Genetic Algorithms have the ability to deliver a good-enough solution fast-enough. This makes genetic algorithms attractive for use in solving optimization problems. The reasons why GAs are needed are as follows −Solving Difficult Problems

In computer science, there is a large set of problems, which are Hard. What this essentially means is that, even the most powerful computing systems take a very long time (even years!) to solve that problem. In such a scenario, GAs prove to be an efficient tool to provide usable near-optimal solutions in a short amount of time.

VII. CONCLUSION

This paper presents the structured and explained view of genetic algorithms. GA and its variants have been discussed with application. Application specific genetic operators are discussed. Some genetic operators are designed for representation. However, they are not applicable to research domains. The role of genetic operators such as crossover, mutation, and selection in alleviating the premature convergence is studied extensively. The applicability of GA and its variants in various research domain has been discussed. Multimedia and wireless network applications were the main attention of this paper. The challenges and issues mentioned in this paper will help the practitioners to carry out their research. There are many advantages of using GAs in other research domains and meta heuristic algorithms.

The intention of this paper is not only provide the source of recent research in GAs, but also provide the information about each component of GA. It will encourage the researchers to understand the fundamentals of GA and use the knowledge in their research problems.

REFERENCES

[1]. Mansi Gangwar, Maiya Din,. K. Jha,“ Comparative Study Of Selection Methods In Genetic Algorithm”, International Journal
of soft Computing and Artificial Intelligence,Volume-5, Issue-1, May-2017, Pp-37-40.
[2]. Khalid Jebari, Mohammed Madiafi , “Selection Methods for Genetic Algorithms” ,International Journal of Emerging Science,
Volume-3, Issue-4, December 2013, Pp:333-344
[3]. Nidhi,”A Comparative Analysis of Genetic Algorithm Selection Techniques”, International Research Journal of Engineering
and Technology (IRJET), Volume: 04 Issue: 07, July -2017, Pp: 2453-2455.
[4]. Nisha Saini, “Review of Selection Methods in Genetic Algorithms”, International Journal Of Engineering And Computer
Science, Volume 6 Issue 12 December 2017, Page No. 22261-22263
[5]. Saneh Lata Yadav, Asha Sohal, “Study of the various selection techniques in Genetic Algorithms” , International Journal of
Engineering, Science and Mathematics, Vol. 6 Issue 3, July 2017, Pp:198-204
[6]. Siew Mooi Lim, et.al., “Crossover and Mutation Operators of Genetic Algorithms”,International Journal of Machine Learning
and Computing, Vol. 7, No. 1, February 2017, Pp:9-12.
[7]. Abdoun Otman, Abouchabaka Jaafar, “A Comparative Study of Adaptive Crossover Operators for Genetic
Algorithms to Resolve the Traveling Salesman Problem” ,International Journal of Computer Applications ,Volume 31– No.11,
October 2011, Pp: 0975 – 8887.
[8]. A.J. Umbarkar and P.D. Sheth, “Crossover Operators in Genetic Algorithms : A Review”, ICTACT Journal on Soft
Computing, October 2015, Volume: 6, Issue:1, Pf:1083-
[9]. Saroj Kaushik, Sunita Tiwari, “ Soft Computing Fundamentals, Techniques and Applications” ,Mc Graw Hill Education.
[10]. https://towardsdatascience.com/introduction- to-genetic-algorithms-including-example-code-e396e98d8bf3