Selective Methods-Genetic operators, Fitness Scaling, GA applications

Evolution, search spaces and fitness landscapes, Elements of Genetic Algorithms, Data structures, Adaptive Encoding.

INTRODUCTION

Genetic Algorithms were invented to mimic some of the processes observed in natural evolution. The idea with GA is to use this power of evolution to solve optimization problems. The father of the original Genetic Algorithm was John Holland who invented it in the early 1970's. Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics.

They are based on the Darwin's theory of evolution which stressed the fact that the existence of all living things is based on the rule of 'survival of the fittest.' Darwin also postulated that new breeds or classes living things come into existence through the process of reproduction, crossover and mutation among existing organisms.

The following table presents a list of different expressions, which are common in genetics, along with their equivalent in the framework of GAs:

Genetics	Genetic algorithm -
Genotype genotype phenotype	coded string uncoded point
chromosome gene allele fitness	string string position value at a certain position objective function value

The basic structure of a genetic algorithm

```
t := 0;

Compute initial population B0;

WHILE stopping condition not fulfilled DO

BEGIN

select individuals for
reproduction; create
offsprings by crossing
individuals; eventually
mutate some individuals;
compute new generation

END
```

As obvious from the above algorithm, the transition from one generation to the next consists of four basic components:

Selection: Mechanism for selecting individuals (strings) for reproduction according to their fitness (objective function value).

Crossover: Method of merging the genetic information of two individuals; if the coding is chosen properly, two good parents produce good children.

Mutation: In real evolution, the genetic material can by changed randomly by erroneous reproduction or other deformations of genes,

e.g. by gamma radiation. In genetic algorithms, mutation can be realized as a random deformation of the strings with a certain probability. The positive effect is preservation of genetic diversity and, as an effect, that local maxima can be avoided.

Sampling: Procedure which computes a new generation from the previous one and its offsprings.

The steps involved in GAs

The steps involved in creating and implementing a genetic algorithm are:

1. Generate an initial, random population of individuals for a fixed size.

2. Evaluate their fitness.

3. Select the fittest members of the population.

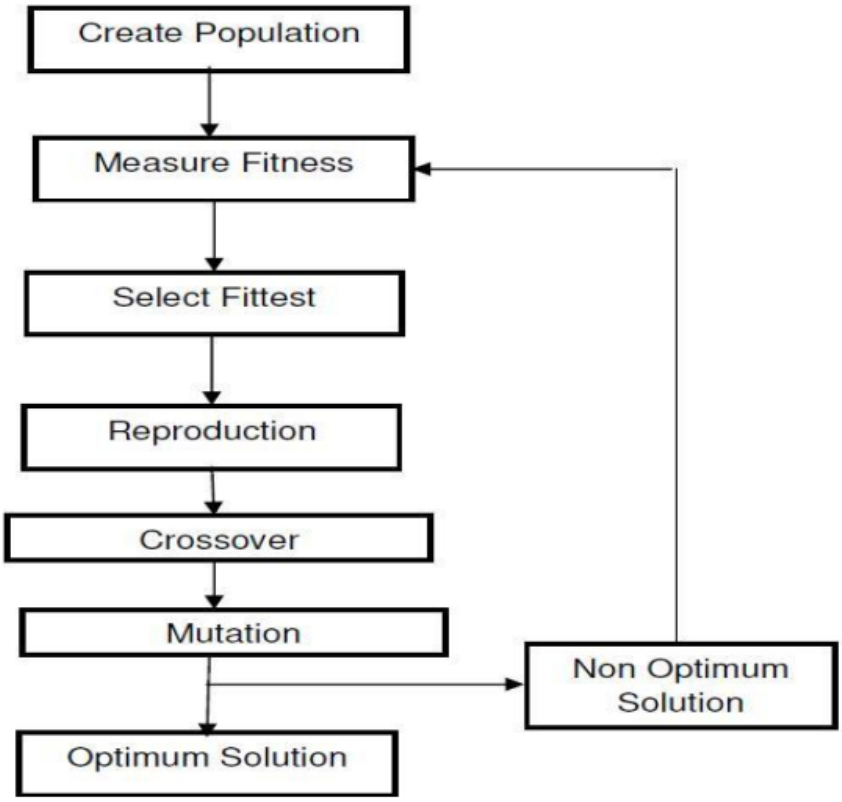
4. Reproduce using a probabilistic method (e.g., roulette wheel).

5. Implement crossover operation on the reproduced chromosomes

6. Execute mutation operation with low probability.

7. Repeat step 2-6 until a predefined convergence criterion is met.

Flow chart



fitness Function

A fitness function is a particular type of objective function that is used to summarize how close a given design solution is to achieving the set aims. After each round of testing, or simulation, the idea is to delete the 'n' worst design solutions, and to breed 'n' new ones from the best design solutions.

The fitness function must not only correlate closely with the designer's goal, it must also be computed quickly. Speed of execution is very important, as a typical genetic algorithm must be iterated many times in order to produce a usable result for a non-trivial problem.

Fitness approximation may be appropriate, especially in the following cases:

- Fitness computation time of a single solution is extremely high
- Precise model for fitness computation is missing
- The fitness function is uncertain or noisy.

Reproduction

During the reproduction phase, the fitness value of each chromosome is assessed. This value is used in the selection process to provide bias towards fitter individuals. Just like in natural evolution, a fit chromosome has a higher probability of being selected for reproduction.

There are four common methods for selection are:

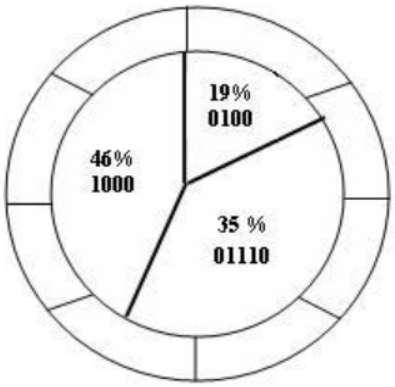
1. Roulette Wheel selection

2. Stochastic Universal sampling

3. Normalized geometric selection

4. Tournament selection

An example of a common selection technique is the ‘Roulette Wheel’ selection method, as shown in Figure. Each individual in the population is allocated a section of a roulette wheel; the size of the section is proportional to the fitness of the individual. A pointer is spun and the individual to whom it points is selected. This continues until the selection criterion has been met. The probability of an individual being selected is thus related to its fitness, ensuring that fitter individuals are more likely to leave offspring. Multiple copies of the same string may be selected for reproduction and the fitter strings should begin to dominate. However, for the situation illustrated in Figure it is not implausible for the weakest string (01001) to dominate the selection process.

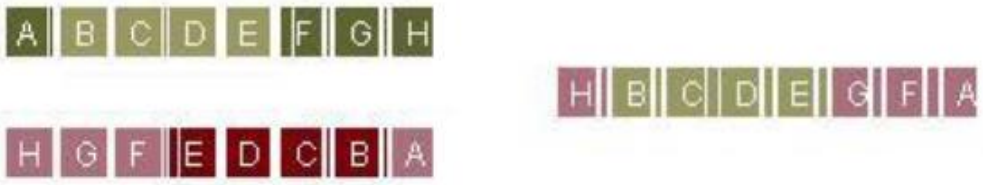


Crossover

Once the selection process is complete, the crossover algorithm is initiated. The crossover operations swap certain parts of the two selected strings in a bid to capture the good parts of old

chromosomes and create better new ones. Genetic operators manipulate the characters of a chromosome directly, using the assumption that certain individual’s gene codes, on average, produce fitter individuals. The crossover probability indicates how often crossover is performed.

A probability of 0% means that the ‘offspring’ will be exact replicas of their ‘parents’ and a probability of 100% means that each generation will be composed of entirely new offspring.



Single point crossover:

The simplest crossover technique is the Single Point Crossover. There are two stages involved in single point crossover:

1. Members of the newly reproduced strings in the mating pool are 'mated' (paired) at random.

2. Each pair of strings undergoes a crossover as follows: An integer k is randomly selected between one and the length of the string less one, [1, L-1]. Swapping all the characters between

positions k+1 and L inclusively creates two new strings.

Example: If the strings 10000 and 01110 are selected for crossover and the value of k is

randomly set to 3 then the newly created strings will be 10010 and 01100 as shown in figure.

More complex crossover techniques exist in the form of Multi-point and Uniform Crossover Algorithms.

Multi-point crossover:

Multi-point crossover is an extension of the single point crossover algorithm and

operates on the principle that the parts of a chromosome that contribute most to its fitness might not be adjacent.

There are three main stages involved in a Multi-point crossover.

1. Members of the newly reproduced strings in the mating pool are ‘mated’ (paired) at random.

2. Multiple positions are selected randomly with no duplicates and sorted into ascending order.

3. The bits between successive crossover points are exchanged to produce new offspring.

Example: If the string 11111 and 00000 were selected for crossover and the multipoint crossover

positions were selected to be 2 and 4 then the newly created strings will be 11001 and 00110 as shown in Figure.

Uniform crossover:

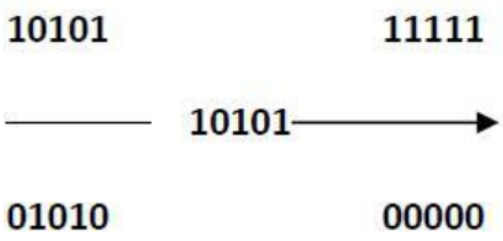
In uniform crossover, a random mask of ones and zeros of the same length as the parent strings is used in a procedure as follows.

1. Members of the newly reproduced strings in the mating pool are ‘mated’ (paired) at random.

2. A mask is placed over each string. If the mask bit is a one, the underlying bit is kept. If the mask bit is a zero then the corresponding bit from the other string is placed in this position.

Example: If the string 10101 and 01010 were selected for crossover with the mask 10101 then

newly created strings would be 11111 and 00000 as shown in Fig.



Uniform crossover is the most disruptive of the crossover algorithms and has the capability to completely dismantle a fit string, rendering it useless in the next generation.

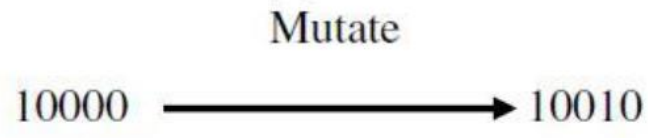
Mutation

Using selection and crossover on their own will generate a large amount of different strings. However there are two main problems with this:

1. Depending on the initial population chosen, there may not be enough diversity in the initial strings to ensure the GA searches the entire problem space.

2. The GA may converge on sub-optimum strings due to a bad choice of initial population.

These problems may be overcome by the introduction of a mutation operator into the GA. Mutation is the occasional random alteration of a value of a string position. It is considered a background operator in the genetic algorithm. The probability of mutation is normally low because a high mutation rate would destroy fit strings and degenerate the genetic algorithm into a random search. Mutation probability values of around 0.1% or 0.01% are common, these values represent the probability that a certain string will be selected for mutation i.e. for a probability of 0.1%; one string in one thousand will be selected for mutation. Once a string is selected for mutation, a randomly chosen element of the string is changed or ‘mutated’. For example, if the GA chooses bit position 4 for mutation in the binary string 10000, the resulting string is 10010 as the fourth bit in the string is flipped as shown in Figure.



Simple C Program

Genetic Algorithm is a program for converging to a required point through a given set of data. This program is written to converge to a population consisting of fittest members.

```
#include<conio.h>

#include<stdio.h>

#include<stdlib.h>

#include<math.h>


void initialize(int *a)

{

        int i;
        for(i=0;i<4;
        i++)

        {

                a[i] = rand()%64;

        }

        printf("\n\tThe present generation
        is:"); for(i=0;i<4;i++)
```

```
        {

                printf(" %d ",a[i]);

        }

}



#include<stdio.h>

#include<stdlib.h>

int fitness(bool*
chromosome); int main()

{

        int i,j,g;
        //counters int
        population=10
        ;

        bool chromosome[10][7]; //population

        //initializing
        population
        for(i=0;i<population

;i++)

{

          for(j=0;j<7;j++)

          {
```

```
                //randomize the
                chromosome
                chromosome[i][j]=rand()
                %2;

        }

}

        for(g=0;g<100;g++)

{

        printf("generation
        %d\n",g); //Evaluation

        int best=0;         //will store the index for the best in the population


        for(i=1;i<population;i++)

        {

                if(fitness(chromosome[best])<fitness(chromosome[i]))

                        best=i;

      }


      //Reproduction

      for(i=0;i<population;i++)
```

```
{

        //to not reproduct with
        itself //it would be a
        waste of time

    ///(although this if could introduce a break in the pipeline, well..

forget about this)

        if(i!=best)

        {
        for(j=0;j<7;j++)

            {

                //either the gene will be kept the same or it

                    will be changed

                    //to be what the best
                    chromosome has if(rand()%2)


chromosome[i][j]=chromosome[best][j];
else

chromosome[i][j]=chromosome[i][j];


                                                    //mutation

    if(rand()%100<4)

      chromosome[i][j]=rand()%2;
```

```
                                }

                            }

                        }

                        printf("best fitness %d\n",fitness(chromosome[best]));


                }
                return 0;

}

int fitness(bool* chromosome)
{
        // the ingredients are: 0   1          2          3  45        6

        // salt sugar lemon egg water onion apple return ( -
        chromosome[0] + chromosome[1] + chromosome[2]

                -chromosome[3] + chromosome[4] -
                chromosome[5] -chromosome[6] );

}
```

Coding the GA:

Various softwares are available for coding Genetic Algorithms. Some of the available softwares are listed as follows:

JGAP

- jMetal
- Jenetics: Java Genetic Algorithm Library
- Java Graticule 3D

Fitness Scaling

- **Linear scaling** : The fitness of each individual in the population is scaled such that the scaled fitness is linearly related to the unscaled fitness

$$f' = af + b$$

- f' is the scaled fitness value,
- f is the actual fitness value
- a' and 'b' are linear co-efficients
- Relationship between the maximum fitness individual in the population and the average population fitness

$$f'max = f_{avg} \times C_s$$

$$f'_{avg} = f_{avg}$$

- f'_{max} is the scaled maximum fitness
- f_{avg} is the average fitness of the population
- C_s and is a scaling constant (specifies the expected number of copies of the best individual in the next generation).

Applications of GA

i. Optimization

ii. Automatic Programming

iii. Machine and robot learning

iv. Economic models

v. Immune system models

vi. Ecological models

vii. Population genetics models

viii. Models of social systems

i. Optimization

- numerical optimization
- combinatorial optimization
- Example problems:
  - traveling salesman problem (TSP)
  - circuit design
  - job shop scheduling
  - video & sound quality optimization.

ii. Automatic Programming

- To design computational structures
- For example

– cellular automata (A cellular automaton consists of a regular grid of *cells*, each in one of a finite number of *states*, such as *on* and *off*)

– sorting networks

iii. Machine and robot learning

- Classification and prediction
  - Example protein structure prediction
- To design neural networks
- To evolve rules for learning classifier systems or symbolic production systems
- To design and control robots.

iv. Economic models

- To model processes of innovation
- Development of bidding strategies
- Emergence of economic markets

v. Immune system models

- To model various aspects of the natural immune system
  - Somatic mutation during an individual's lifetime
  - Discovery of multi-gene families during evolutionary time.

vi. Ecological models

- biological arms races(an evolutionary struggle between competing sets of co-evolving genes that develop adaptations and counter-adaptations against each other)

- host-parasite co-evolutions
- symbiosis (individual interactions)
- resource flow in ecologies.

vii. Population genetics models

- To study questions in population genetics, such as "under what conditions will a gene for recombination be evolutionarily viable?"
- GAs have been used to study how individual learning and species evolution affect one another.

viii. Models of social systems:

- Evolution of cooperation
- Evolution of communication
- Trail-following behavior in ants.

Reference

Eiben, A. E. et al (1994). "Genetic algorithms with multi-parent recombination". PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: 78–87. ISBN 3-540-58484-6.

Ting, Chuan-Kang (2005). "On the Mean Convergence Time of Multi-parent Genetic Algorithms Without Selection". Advances in Artificial Life: 403–412. ISBN 978-3-540-28848-0.

Shir, Ofer M. (2012). "Niching in Evolutionary Algorithms". In Rozenberg, Grzegorz; Bäck, Thomas; Kok, Joost N. (eds.). Handbook of Natural Computing. Springer Berlin Heidelberg. pp. 1035–1069. doi:10.1007/978-3-540-92910-9_32. ISBN 9783540929093.

Harik, Georges R.; Lobo, Fernando G.; Sastry, Kumara (1 January 2006). Linkage Learning via Probabilistic Modeling in the Extended Compact Genetic Algorithm (ECGA). Scalable Optimization Via Probabilistic Modeling. Studies in Computational Intelligence. 33. pp. 39–61

Pelikan, Martin; Goldberg, David E.; Cantú-Paz, Erick (1 January 1999). BOA: The Bayesian Optimization Algorithm. Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1. Gecco'99. pp. 525–532. ISBN 9781558606111.

Sadowski, Krzysztof L.; Bosman, Peter A.N.; Thierens, Dirk (1 January 2013). On the Usefulness of Linkage Processing for Solving MAX-SAT. Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation. Gecco '13. pp. 853–860.

Goldberg, David E. (1991). "The theory of virtual alphabets". Parallel Problem Solving from Nature. Parallel Problem Solving from Nature, Lecture Notes in Computer Science. Lecture Notes in Computer Science.

Zhang, J.; Chung, H.; Lo, W. L. (2007). "Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms". IEEE Transactions on Evolutionary Computation