Genetic Algorithms and Evolutionary Computation

Matteo Matteucci and Andrea Bonarini

{matteucci,bonarini}@elet.polimi.it

Department of Electronics and Information

Politecnico di Milano

Genetic Algorithms

– Introduction –

Terminology (From Biology)

Evolution is a long time scale process that changes a population of organisms by generating better offsprings trough reproduction

- Chromosome: DNA-coded information characterizing an organism
- Gene: Elementary DNA block of information (e.g., eyes color)
- Allele: One of the possible values for a gene (e.g., brown, blue, . . .)
- Trait: The physical characteristic encoded by a gene
- Genotype: A particular set of genes
- Phenotype: The physical realization of a genotype (e.g., a person)
- Fitness: A measure of success in life for an organism

- Crossover: Chromosomes from the parents exchange genetic materials to generate a new offspring
- Mutation: Error occurring during DNA replication from parents

From Biology to Genetic Algorithms

We can borrow some terms (and ideas) from biology . . .

- Chromosome: The coding of a possible solution for a given problem, usually represented with an array of bits or characters
- Gene: A single bit or a set of bits coding part of the solution
- Allele: One of the elements used to code the genes
- Fitness: Evaluation of the actual solution

. . . to model a learning process as evolution:

- Crossover: Generate new solution by “mixing” two existing solutions
- Mutation: Random change in the solution

Genetic Algorithms: A Powerful Idea from Nature

Genetic Algorithms are a part of evolutionary computing, and they are inspired by Darwin's theory of evolution:

Problems are solved by an evolutionary process that mimics natural evolution in looking for a best (fittest) solution (survivor)

We can trace a brief history of evolutionary computation:

1. 1960: Ingo Rechenberg introduces the idea of evolutionary computing in his work "Evolution strategies"
2. 1975: John Holland invents Genetic Algorithms and publish his book "Adaption in Natural and Artificial Systems"
3. 1992: John Koza uses genetic algorithm to evolve programs to perform certain tasks. He called his method Genetic Programming
4. 1995: Stewart Wilson re-invents learning classifier systems with XCS: GA to learn rules
5. . . .

Genetic Algorithm Applications

They have been used for many applications:

- Optimization (e.g., circuits layout, job shop scheduling, . . .)
- Prediction (e.g., weather forecast, protein folding, . . .)
- Classification (e.g., fraud detection, quality assessment, . . .)
- Economy (e.g., bidding strategies, market evaluation, . . .)
- Ecology (e.g., biological arm races, host-parasite coevolution, . . .)
- Automatic programming
- . . .

In general they are best suited for

- Big search space, non unimodal, non smooth
- Noisy fitness function, usually not analytic
- We do not want to spend years looking for the global optimum, but we just want a good sub-optimum in a reasonable time

Outline of the Basic Genetic Algorithm

1. [Start] Generate random population of n chromosomes (suitable solutions)
2. [Fitness] Evaluate the fitness f(x) of each chromosome x in the population
3. [New population] Create a new population by repeating the following steps until the new population is complete
   (a) [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
   (b) [Crossover] With a crossover probability cross over the parents to form new offspring. If no crossover was performed, offspring is the exact copy of parents.
   (c) [Mutation] With a mutation probability mutate new offsprings at each locus
   (d) [Accepting] Place new offsprings in the new population
4. [Replace] Use new generated population for a further run of the algorithm
5. [Test] If the end condition is satisfied, return the best solution in current population
6. [Loop] Go to step 2

Two Comments to the Basic Genetic Algorithm

There are many parameters and settings that can be implemented differently in various problems:

- How to create chromosomes and what type of encoding choose
- How to select parents for crossover in the hope that the better parents will produce better offspring
- How to define crossover and mutation, the two basic operators of GA

It seems there is some “black magic” behind genetic algorithms . . . Why do genetic algorithms work?

- It can be partially explained by the Schema Theorem
- They have been modeled using Monte Carlo Markov Chains

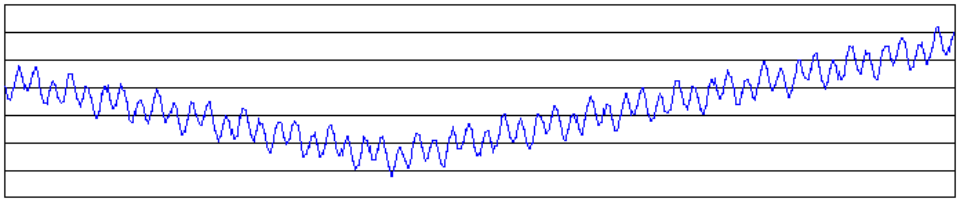
But first of all we’ll have a look to a simple example . . .

Example: Minimum of a Function

In this simple example we are looking for the extreme of a function defined over a search space.

1. Search Space: An interval of the real line
2. Fitness Function: The value of the function we are “exploring”

Why should we use genetic algorithms for this?



Functions can get quite nasty ;)

Minimization Example: Chromosome Encoding

The first step in developing a genetic algorithm is defining a solution encoding:

- A chromosome should in some way contain information about the solution that it represents
- The encoding depends mainly on the solved problem (e.g., integer or real numbers, permutations, parsing trees, . . .)
- The most used way of encoding is a binary string; each bit in the string can represent some characteristics of the solution

Our chromosome then could look like this:

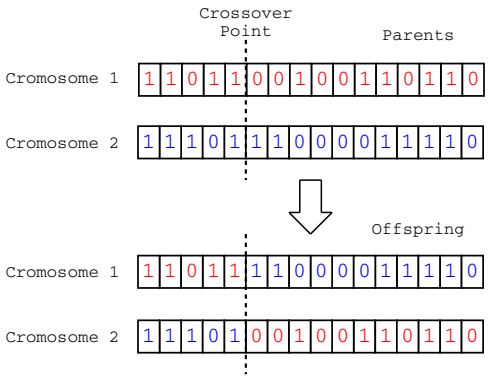
Cromosome 1	1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	0
Cromosome 2	1	1	1	0	1	1	1	0	0	0	0	1	1	1	1	0

Each chromosome is represented by the binary code of a real number

Minimization Example: Crossover Operator

Crossover operates on selected genes from parent chromosomes and creates new offspring, the simplest way to do that is:

1. Choose randomly some crossover point in the chromosome
2. Copy everything before this point from the first parent and then copy everything after the crossover point from the other parent

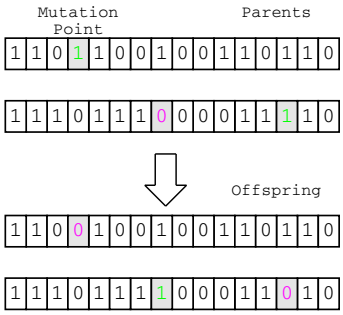


Note: Crossover depends mainly on the encoding of chromosomes. Specific crossover for a specific problem can improve performance.

Minimization Example: Mutation Operator

After a crossover is performed, mutation takes place:

- Switch a few randomly chosen bits from 1 to 0 or from 0 to 1



Few notes on mutation:

- Mutation is intended to prevent falling of all solutions in the population into a local optimum
- Also mutation depends on the encoding of chromosomes (e.g., when we are encoding permutations, mutation could be performed as an exchange of two genes)

Minimization Example: Demo

Stolen from:
http://www.obitko.com/tutorials/genetic-algorithms/example-function-minimum.php

Genetic Algorithms

– Into the groove –

Genetic Algorithms Explained: Selection

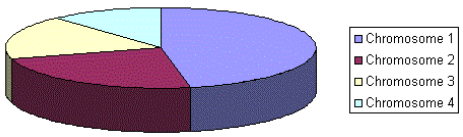
According to Darwin's theory of evolution the best chromosome survive to create new offspring. There are many methods to select the best chromosomes:

- Roulette wheel selection,
- Rank selection
- Tournament selection
- Boltzmann selection
- . . .

Roulette Wheel Selection

Roulette Wheel Selection: Parents are selected proportionally to their fitness. The better they are, the more chances to be selected they have.

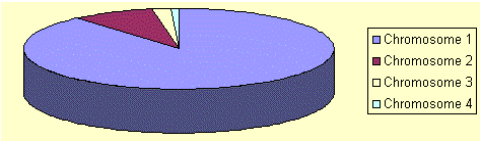
1. Imagine a roulette wheel where all the chromosomes in the population are placed
2. The size of the section in the roulette wheel is proportional to the value of the fitness function of every chromosome - the bigger the value is, the larger the section is
3. A marble is thrown in the roulette wheel and the chromosome where it stops is selected



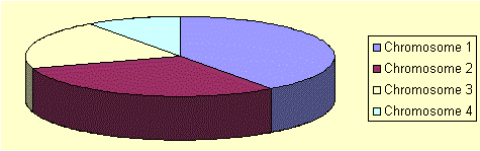
Possible problems if there is a large difference of fitness from the best to the worst, which could hardly be selected.

Rank Selection

Rank Selection: Parents are ranked and the selection probability is proportional to the rank.



Roulette before ranking



Roulette after ranking

Possible problems: slow convergence, due to small difference between best and worst parents.

Tournament Selection

Tournament Selection: Parents are pooled and a tournament is held within the pool(s).

Pseudocode:

- choose k (the tournament size) individuals from the population at random
- choose the best individual from pool/tournament with probability p
- choose the second best individual with probability $p * (1 - p)$
- choose the third best individual with probability $p * ((1 - p)^2)$
- and so on...

Efficient implementation, easy to adjust.

Boltzmann Selection

Boltzmann Selection: Parents are selected with a probability that favors exploration at the beginning of learning and tends to stabilize and select the best solutions as generations proceed.

$P = e^{-f_{Max} - f(X_i)/T}$

$T = T_0(1-\alpha)^k$, with $\alpha \in [0,1]$, and $T_0 \in [5,100]$

$k = 1 + 100 * g/G$,

where g is the generation number and G the maximum number of generations

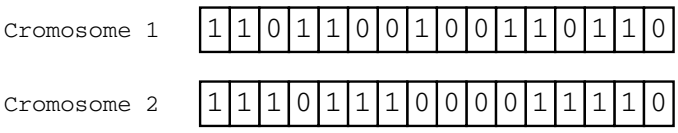
Genetic Algorithms Explained: Elitism

When creating a new population by crossover and mutation, we have a big chance that we will lose the best chromosome.
Elitism is the name of the method that first copies the best chromosome (or few best chromosomes) to the new population. It can rapidly increase the performance, because it prevents a loss of the so–far best found solution.

Genetic Algorithms Explained: Encoding (I)

Binary encoding is the most common one, mainly because, when all configurations are used, it guarantees the maximum exploitation of the information representation.

- In binary encoding, every chromosome is a string of bits (0 or 1)
- Simple Implementation of the genetic operators
- Not natural for many problems



Example of Problem: Knapsack problem

- There are things with given value and size. The knapsack has given capacity. Select things to maximize the value of things in knapsack, but do not extend knapsack capacity.
- Each bit says whether the corresponding thing is in the knapsack.

Genetic Algorithms Explained: Crossover/Mutation (I)

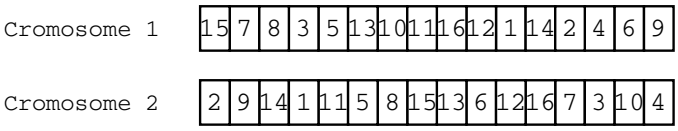
For binary encoding we have many operators:

- Single point crossover: one crossover point is selected, binary string from the beginning of the chromosome to the crossover point is copied from the first parent, the rest is copied from the other parent
- Two point crossover: two crossover points are selected, binary string from the beginning of the chromosome to the first crossover point is copied from the first parent, the part from the first to the second crossover point is copied from the other parent and the rest is copied from the first parent again
- Uniform crossover: bits are randomly copied from the first or from the second parent
- Arithmetic crossover: some arithmetic operation is performed to make a new offspring (e.g., logic AND)
- Mutation: inversion of bits selected with a given probability

Genetic Algorithms Explained: Encoding (II)

Permutation encoding can be used in ordering problems

- Every chromosome is a string of numbers that represent a position in a sequence
- Crossover and mutation must be designed to leave the chromosome consistent (i.e., have real sequence in it)



Example of Problem: Traveling salesman problem (TSP)

- There are cities and given distances between them. Traveling salesman has to visit all of them, but he does not want to travel more than necessary. Find a sequence of cities with a minimal traveled distance.
- Chromosome describes the order of cities

Genetic Algorithms Explained: Crossover/Mutation (II)

For permutation encoding we have to preserve consistency:

- Single point crossover: one crossover point is selected, the permutation is copied from the first parent till the crossover point, then the other parent is scanned looking the other numbers

  (1 2 3 4 5 6 7 8 9) + (4 5 3 6 8 9 7 2 1) = (1 2 3 4 5 6 8 9 7)

- Order changing mutation: two numbers are selected and exchanged

  (1 2 3 4 5 6 8 9 7) $\Rightarrow$ (1 8 3 4 5 6 2 9 7)

Genetic Algorithms Explained: Encoding (III)

Direct value encoding can be used in problems where some more complicated values are required

- Every chromosome is a sequence of some values connected to the problem, such as (real) numbers, chars or any objects
- Good choice for some special problems, but necessary to develop some specific crossover and mutation

Cromosome	A	B	D	H	Y	V	S	V

Cromosome	2.5678	1.4361	3.3426	7.8761

Cromosome	open	walk	back	close

Example of Problem: Finding weights for a neural network

- A neural network is given with defined architecture. Find weights between neurons to get the desired output from the network
- Real values in chromosomes represent weights in the neural network

Genetic Algorithms Explained: Crossover/Mutation (III)

For real value encoding we can reuse crossover from binary encoding:

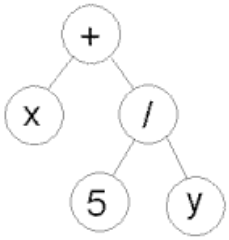
- Mutation: a small number is added (or subtracted) to selected values

$$(1.29\ 5.68\ 2.86\ 4.11\ 5.55) \Rightarrow (1.29\ 5.68\ 2.73\ 4.22\ 5.55)$$

Genetic Algorithms Explained: Encoding (IV)

Tree encoding is used mainly for evolving programs or expressions (i.e., genetic programming)

- Every chromosome is a tree of some objects, such as functions or commands in programming language.
- Programming language LISP is often used for this purpose, so crossover and mutation can be done relatively easily.



`Cromosome (+ X (/ 5 y))`

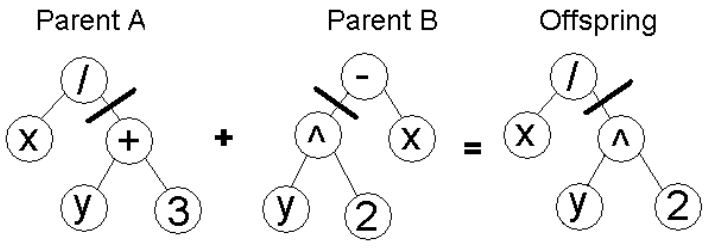
Example of Problem: Finding a function that would best match given pairs of values (approximant function)

- Input and output values are given. The task is to find a function that will give the best outputs for all inputs.
- Chromosome are functions represented in a tree

Genetic Algorithms Explained: Crossover/Mutation (IV)

Specific operators have to be selected also for tree encoding

- Tree crossover: one crossover point is selected in both parents, parents are divided in that point and the parts below crossover points are exchanged to produce new offspring



- Changing mutation: the operator, number, or variable in a randomly selected nodes is changed

Time for Another Demo

Stolen again from:
http://www.obitko.com/tutorials/genetic-algorithms/tsp-example.php

Tips & Tricks

These “rules of thumb” are often results of empiric studies performed on binary encoding only, but they usually work fine . . .

- Crossover rate should be high, generally about $80\% - 95\%$ (However some results show that for some problems crossover rate about 60% is the best.)
- Mutation rate should be very low. Good rates could be about $0.5\% - 1\%$ per allele
- Very big population size usually does not improve performance (in the sense of speed of finding solution). Good population size is about $20 - 30$, however sometimes sizes $50 - 100$ are reported as the best
- Basic roulette wheel selection can be used, but sometimes rank selection can be better. Elitism should be used for sure if you do not use other method for saving the best found solution

Understanding Genetic Algorithms: Building Blocks

Holland (1975) formalized the idea of building block and schema:

A string S will contain sub-strings. We can represent this as a 'similarity template string' (i.e., the schema) H which uses a 'wild card' symbol to mark positions not belonging to the sub-strings (they can take 1 or 0).

Example: schema **11 represents strings 1111, 0011, 1011 and 0111. Alternatively, 1011 and 1101 contain common schemata 1***, ***1 and 1**1.

Note: Since 2 strings have 3 common schemata each, making a total of 6 schemata; processing a string implicitly processes many more schemata. In general, for a string of length l, there are 3^l schemata.

This property is usually referred as implicit parallelism.

Understanding Genetic Algorithms: Schema Theorem

By evaluating many strings we implicitly estimate the expected value of schemata, and working out a "little bit of math" we can approximate the expected number of schemata at next step:

$$M(H_i, t+1) \geq M(H_i, t) \cdot \left[\frac{f(H_i)}{\bar{f}}\right] \cdot \left[1 - p_c \cdot \frac{\delta(H_i)}{l-1}\right] \cdot \left[(1-p_m)^{o(H_i)}\right].$$

- $f(H_i)$: Mean fitness of the i^{th} schema
- $\bar{f}$: Mean fitness of population
- $\delta(H_i)$: Length of the i^{th} schema
- l: Length of strings in the search space
- $o(H_i)$: Number of known bits in the schema
- p_m and p_c: Mutation and Crossover probabilities

Understanding Genetic Algorithms: What's going on?

The previous formula states two important concepts for the understanding of genetics algorithm:

Schema Theorem

Above average fitness, short, low-order schemata will have a large survival probability. They will grow at least exponentially in the population.

Building Block Hypothesis

Short, highly fit, low order schemata are called building blocks; it is thought that they represent partial solutions to the problem and that processing them will build up the full solution.