Introducción a la Bioinformática:

Genetic Algorithms

Luis Garreta

Doctorado en Ingeniería
Pontificia Universidad Javeriana – Cali

April 19, 2017

Genetic Algorithms History

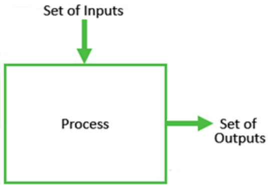
- (1962) GAs were developed by John Holland and his students and colleagues at the University of Michigan
- (1975) John Holland published Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence:
  - Considered by most to be the seminal work in the field
  - Established formal, theoretical basis for evolutionary optimization with introduction of schemata (building blocks, partial solutions)

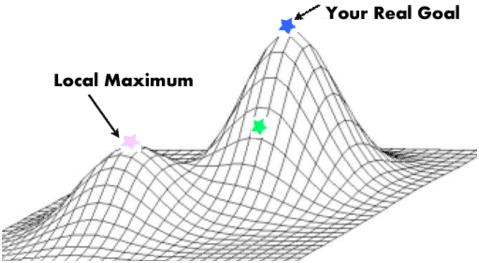
What are Genetic Algorithms?

- A way to employ evolution in the computer
- Based on the principles of **Genetics and Natural Selection**.
- Search and optimization technique based on variation and selection
- Used **to find optimal or near-optimal solutions:**
  - difficult problems (NP-complete) which otherwise would take a lifetime to solve.

What is Optimization?

- Process of making something better
- Finding the values of inputs causing the “best” output values.
- In mathematical terms, "best":
  - maximizing or minimizing one objective function,
  - by varying the input parameters.
- Search space:
  - Possible solutions or values which the inputs can take

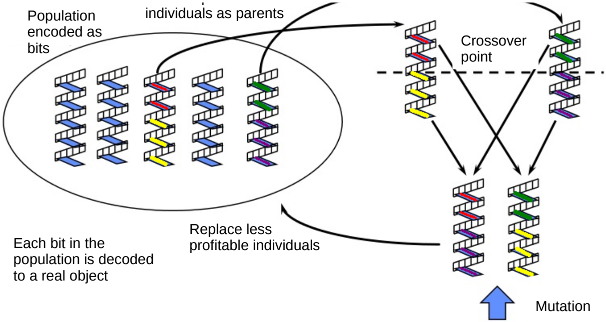




Genetic Algorithm Model

Darwinian Theory of "Survival of the Fittest"

A pool or a population of possible solutions to the given problem undergo recombination and mutation, 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 more "fitter" individuals.

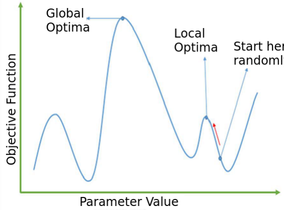


Random but Exploiting Historical Information

- In this way we keep "evolving" better individuals or solutions over generations, till we reach a stopping criterion.
- GAs 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 GAs **exploit historical information** as well.

Advantages of GAs

- Solving Difficult Problems (e.g. NP-Hard problems):
  - Provide usable near-optimal solutions in a short amount of time
- It is faster and more efficient as compared to the traditional methods:
  - High climbing
  - Gradient descent



- Has very good parallel capabilities.
- Always gets an answer to the problem, which gets better over the time.
  - Provides a list of “good” solutions and not just a single solution.
- Useful when the search space is very large and there are a large number of parameters involved.

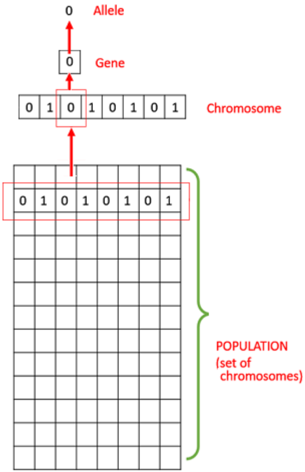
Limitations of GAs

Like any technique, GAs also suffer from a few limitations:

- **Fitness value is calculated repeatedly:**
  - It might be computationally expensive for some problems.
- **Being stochastic:**
  - There are no guarantees on the optimality or the quality of the solution.
- **Depends of the Implementation** (Codification):
  - Improperly implementations may not converge to the optimal solution.

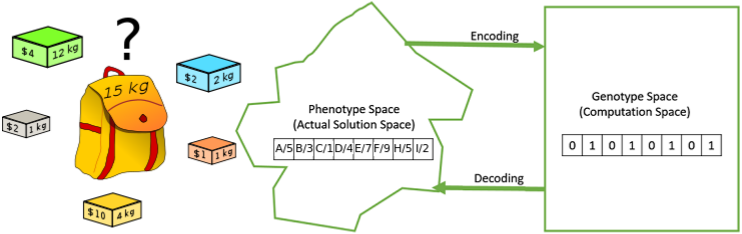
Terminology

- **Population**: subset of all the possible (encoded) solutions to the given problem.
- **Chromosome:** one such solution to the given problem.
- **Gene**: one element position of a chromosome.
- **Allele:** The value of a gene.
- **Genotype:** Population in the computation space
- **Phenotype:** Population in the actual real world solution space
- **Decoding:** Process of transforming a solution from the genotype to the phenotype space,
- **Encoding:** Process of transforming from the phenotype to genotype space.
- **Fitness Function**: function which takes the solution as input and produces the suitability of the solution as the output.
- **Genetic Operators:** Alter the genetic composition of the offspring. These include crossover, mutation, selection, etc.



GAs Considerations

- Phenotype and Genotype:
  - For **simple problems**, the phenotype and genotype **spaces are the same**.
  - But, in most of the cases, the **phenotype and genotype are different**.
  - **Decoding should be fast** as it is carried out **repeatedly** in a GA during the fitness value calculation.
  - Example: Knapsack Problem



- Fitness Function:
  - In some cases, the fitness function and the objective function may be the same, while in others it might be different based on the problem.

Simple Genetic Algorithm

```
produce an initial population of individuals
evaluate the fitness of all individuals
while termination condition not met do
  select fitter individuals for reproduction
  recombine between individuals
  mutate individuals
  evaluate the fitness of the modified individuals
  generate a new population
End while
Return Best
```

Genotype Representation

- It is crucial for GAs **how to represent the solutions**.
- **Improper representation** leads to **poor** performance of the GA.
- A **proper mapping** between the phenotype and genotype is essential for **success of GA**.
- There are common representations for GAs:
  - However, representation is **highly problem specific**
  - **Another representation** may be better, or
  - a **mix of the representations**.

Binary Representation

- One of the simplest and most used representations.
- Genotype consists of bit strings:

0	0	1	0	1	1	1	0	0	1

- Ideal for problems when solution space consists of YES/NO decisions:
  - e.g: Knapsack problem:
    - 0: Xth Element is not picked
    - 1: Xth Element is picked
- Also, it is used for problems involving numbers, but:
  - Problems as positions of bits have significance,
  - Crossover and mutation affect the representation

Number Value Representation

- Real Representation:
  - For problems where we want defines genes using continuous than discrete variables.

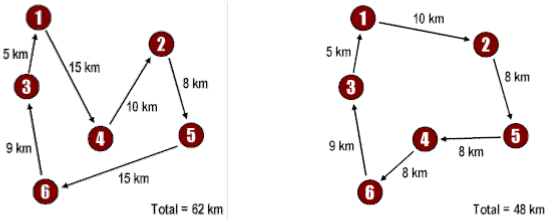
0.5	0.2	0.6	0.8	0.7	0.4	0.3	0.2	0.1	0.9

- Integer Representation:
  - For discrete values genes not limited to binary YES/NO.
  - e.g. Nucleotide Alignments with alphabet: ACGT –> 1234

1	2	3	4	3	2	4	1	2	1

Permutation Representation

- In many problems, the solution is represented by an **order of elements**.
- In such cases **permutation representation is the most suited**.
- e.g. TSP Problem (Travelling Salesman Problem):

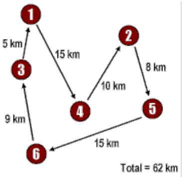


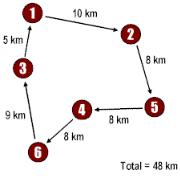
- The solution: an ordering or permutation of all the cities,
- Using a permutation representation makes sense for this problem:

1	4	2	5	6	3	**62**
1	2	5	4	6	3	**48**

Population

- Population is a **subset of solutions** in the current generation.
  - It can also be defined as a **set of chromosomes**.
- Several things to be **kept in mind** when dealing with GA population:
  - **The diversity** should be maintained otherwise it might lead to **premature convergence**.
  - **The Size** should not be kept very large (Causes GA to slow down)
  - **Smaller population** might **not be good** mating pool.
  - **An optimal population** size needs to be decided by **trial and error**.
- Usually defined as a **2D array** of size population by chromosome size.
- e.g TSP:





1	4	2	5	6	3
1	2	5	3	6	4
.					
.					
.					
5	2	1	4	6	3
3	2	5	6	4	1
1	4	5	2	6	3

Population Initialization

- Random Initialization:
  - *Populate the initial population with completely random solutions.*
  - Good as it drives the population to the optimality (**more diversity**)
  - But, take **more time**.
- Heuristic initialization:
  - *Populate the initial population using a known heuristic for the problem*.
  - Good, as it seeds the solution with **good individuals**,
  - But, bad if all are "goog" individuals (**loss of diversity**)
  - Better, **half and half**.

Next: *Population Models*

Population Models

Two population models widely in use:

- **Steady State:**
  - Generate **one or two off-springs** in each iteration, and
  - they **replace one or two individuals** from the population.
  - A steady state GA is also known as **Incremental GA**.
- **Generational:**
  - Generate **'n' off-springs**, where **n is the population size**,
  - The **entire population is replaced** by the new one.

Next: *Fitness Function*

Fitness Function

A function taking as **input a candidate solution** to the problem, and producing as **output how "fit" our how "good" the solution** is with respect to the problem.

Next: *Fitness Function Considerations*

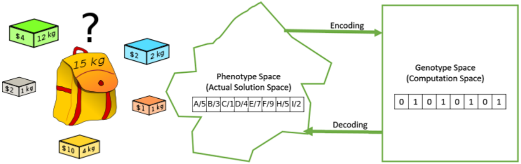
Fitness Function Considerations

- The fitness function should be sufficiently **fast to compute.**
  - as calculation of fitness value is done **repeatedly**
- A **slow computation** of the fitness value:
  - Can adversely affect a GA and make it **exceptionally slow**.
- Mostly the **fitness function and the objective function are the same**
  - The objective is to either maximize or minimize the objective function.

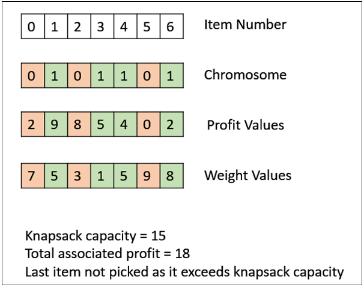
It must quantitatively **measure how fit a given solution is**

Next: *Fitness Function For Ksnapsack Problem*

Fitness Function for Knapsack Problem



The fitness calculation for a solution is a simple fitness function which just sums the profit values of the items being picked (which have a 1), scanning the elements from left to right till the knapsack is full.



Next: *Parent Selection*

Parent Selection

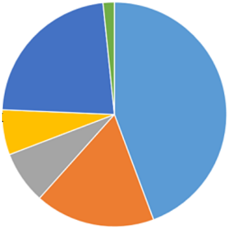
Parent Selection is the process of selecting parents which mate and recombine to create off-springs for the next generation

- Parent selection **is very crucial to the convergence** rate of the GA:
  - as good parents drive individuals to a **better and fitter solutions**.
- Care should be taken to prevent **one extremely fit solution**
  - It can take over the entire population **in a few generations**
  - **Solutions being close to one another** in the solution space thereby leading to a **loss of diversity**.
  - Maintaining **good diversity** in the population is extremely **crucial** for the success of a GA.
  - Known as **Premature Convergence**, an undesirable condition in a GA.

Next: *Fitness Proportionate Selection*

Fitness Proportionate Selection

- One of the most popular ways of parent selection.
- Every individual can become a parent:
  - with a **probability which is proportional to its fitness**.
  - **Fitter individuals have a higher chance**
  - This strategy applies **a selection pressure** to the more fit individuals in the population, **evolving better individuals over time**.
- Consider a circular wheel divided into n pies:
  - n individuals in the population.
  - Each on gets a portion of the circle proportional to its fitness value.

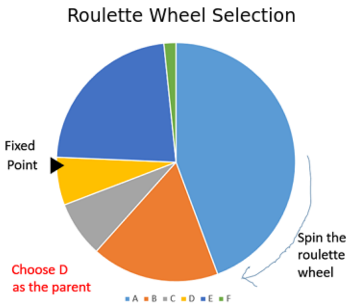


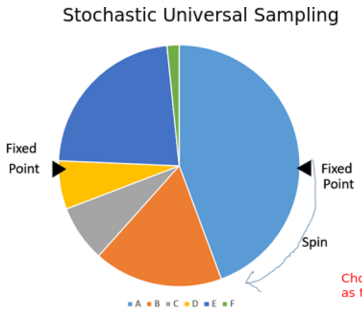
Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Next: *Implementing Fitness Proportionate Selection*

Implementing *Fitness Proportionate Selection*

Two implementations of fitness proportionate selection are possible:





Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Next: *Other Methods for Parent Selection*

Others Methods for Parent Selection

- Tournament Selection:
  - We select K individuals from the population at random and select the best out of these to become a parent.
- Random Selection:
  - In this strategy we randomly select parents from the existing population.

Next: *Crossover*

Crossover

- The crossover operator is analogous to **reproduction and biological crossover**.
  - In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents.
- Crossover is usually applied in a GA with a high probability – pc .

Next: *Crossover Operators*

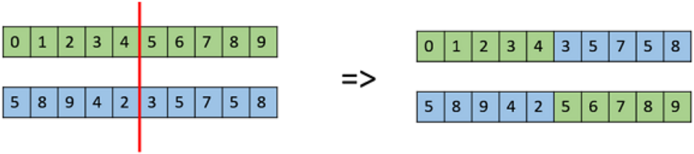
Crossover Operators

- There are popularly used crossover operators
- These crossover operators are very generic
- GA Designer might choose to implement a **problem-specific crossover operator** as well.

Next: *One Point Crossover*

One Point Crossover

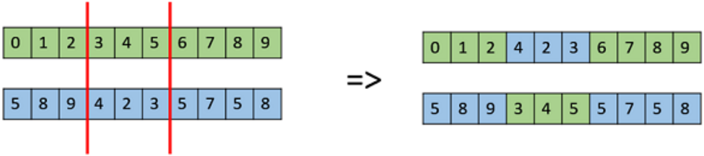
A random crossover point is selected and the tails of its two parents are swapped to get new off-springs



Next: *Multi Point Crossover*

Multi Point Crossover

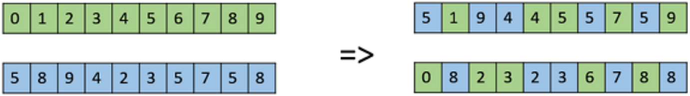
A generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.



Next: *Uniform Crossover*

Uniform Crossover

- We don't divide the chromosome into segments, rather **we treat each gene separately**
- The uniform crossover evaluates each bit in the parent strings for exchange with a probability of ***p***
- We can also bias the coin **to one paren**t, to have more genetic material in the child from that parent.



Next: *Mutation*

Mutation

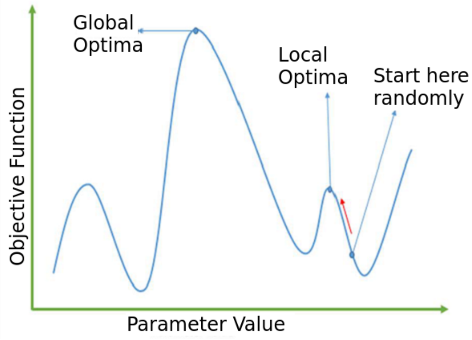
Mutation may be defined as a **small random tweak in the chromosome**, to get a new solution.

- It is used to **maintain and introduce diversity** in the genetic population, and
- it is usually applied with a **low probability – pm**.
- **If the probability is high**, the GA gets reduced to a **random search**.

Next: *Mutation for exploring the Search Space*

Mutation for exploring the Search Space

- Mutation is the part of the GA which is related to the "exploration" of the search space.
- It has been observed that mutation is essential to the convergence of the GA while crossover is not.



Next: *Mutation Operators*

Mutation Operators

- There are many commonly used mutation operators.
- Like the crossover operators,
  - the GA designer might find a **combination** of these approaches or
  - **a problem-specific** mutation operator more useful.

Bit Flip Mutation

- In this bit flip mutation, we select one or more random bits and flip them.
- This is used for binary encoded GAs.

0	0	1	1	0	1	0	0	1	0	=>	0	0	1	0	0	1	0	0	1	0

Random Resetting

- Random Resetting is an extension of the bit flip for the integer representation.
- In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

Swap Mutation

- Scramble mutation is also popular with **permutation representations**.
- A subset of genes is chosen and their **values are scrambled or shuffled randomly**.

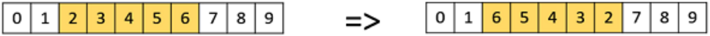
0	1	2	3	4	5	6	7	8	9

=>

0	1	3	6	4	2	5	7	8	9

Inversion Mutation

- we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.



Next: *Survivor Selection*

Survivor Selection

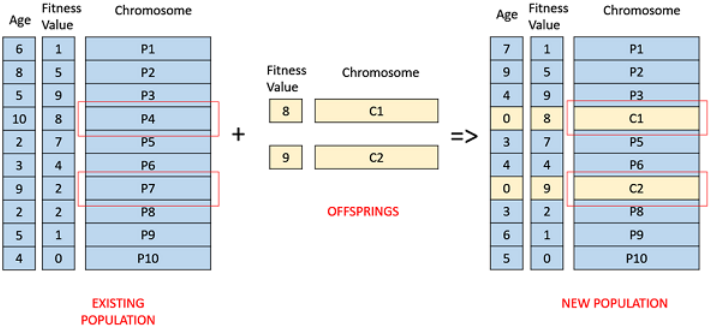
- The Survivor Selection Policy determines:
  - **which individuals are to be kicked out** and
  - **which are to be kept in** the next generation.
- It is crucial as it **should ensure:**
  - that the **fitter individuals are not kicked out** of the population,
  - while at the same time **diversity should be maintained** in the population.

Elitism and Random

- Some GAs employ Elitism.
  - It means the **current fittest member of the population is always propagated** to the next generation.
  - Therefore, under no circumstance can the fittest member of the current population be replaced.
- The easiest policy is to kick random members out of the population:
  - but such an approach frequently has convergence issues, therefore the following strategies are widely used.

Age Based Selection

- In Age-Based Selection, we don't have a notion of a fitness.
- It is based on the premise that each individual is allowed in the population for a finite generation where it is allowed to reproduce,
- After that, it is kicked out of the population no matter how good its fitness is.



Fitness Based Selection

- The children tend to replace the least fit individuals in the population.
- The selection of the least fit individuals may be done using a variation of any of the selection policies described before – tournament selection, fitness proportionate selection, etc.