**Genetic Algorithm**

**Tabish, CSE-09/16**

**Department of Computer Science & Engineering, NIT Srinagar**

**Introduction to the Algorithm:**

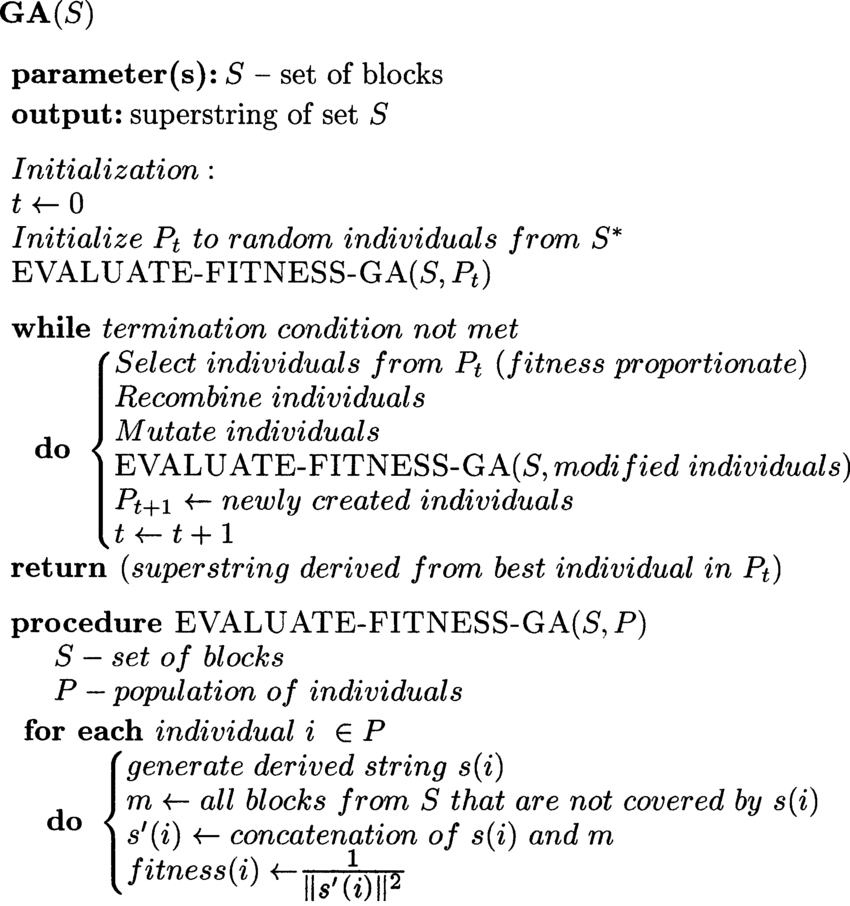
Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate “survival of the fittest” among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

Genetic algorithms are based on an analogy with genetic structure and behavior of chromosome of the population. Following is the foundation of GAs based on this analogy –

1. Individual in population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring than others
3. Genes from “fittest” parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
4. Thus each successive generation is more suited for their environment.

**Algorithm:**



**Code:**

**POPULATION\_SIZE = 100**

**GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP**

**QRSTUVWXYZ 1234567890, .-;:\_!"#%&/()=?@${[]}'''**

**# Target string to be generated**

**TARGET = "I love GeeksforGeeks"**

**class Individual(object):**

**'''**

**Class representing individual in population**

**'''**

**def \_\_init\_\_(self, chromosome):**

**self.chromosome = chromosome**

**self.fitness = self.cal\_fitness()**

**@classmethod**

**def mutated\_genes(self):**

**global GENES**

**gene = random.choice(GENES)**

**return gene**

**@classmethod**

**def create\_gnome(self):**

**'''**

**create chromosome or string of genes**

**'''**

**global TARGET**

**gnome\_len = len(TARGET)**

**return [self.mutated\_genes() for \_ in range(gnome\_len)]**

**def mate(self, par2):**

**# chromosome for offspring**

**child\_chromosome = []**

**for gp1, gp2 in zip(self.chromosome, par2.chromosome):**

**# random probability**

**prob = random.random()**

**# if prob is less than 0.45, insert gene**

**# from parent 1**

**if prob < 0.45:**

**child\_chromosome.append(gp1)**

**# if prob is between 0.45 and 0.90, insert**

**# gene from parent 2**

**elif prob < 0.90:**

**child\_chromosome.append(gp2)**

**# otherwise insert random gene(mutate),**

**# for maintaining diversity**

**else:**

**child\_chromosome.append(self.mutated\_genes())**

**# create new Individual(offspring) using**

**# generated chromosome for offspring**

**return Individual(child\_chromosome)**

**population = new\_generation**

**print("Generation: {}\tString: {}\tFitness: {}".\**

**format(generation,**

**"".join(population[0].chromosome),**

**population[0].fitness))**

**generation += 1**

**print("Generation: {}\tString: {}\tFitness: {}".\**

**format(generation,**

**"".join(population[0].chromosome),**

**population[0].fitness))**

**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**

**def cal\_fitness(self):**

**'''**

**Calculate fittness score, it is the number of**

**characters in string which differ from target**

**string.**

**'''**

**global TARGET**

**fitness = 0**

**for gs, gt in zip(self.chromosome, TARGET):**

**if gs != gt: fitness+= 1**

**return fitness**

**def main():**

**global POPULATION\_SIZE**

**#current generation**

**generation = 1**

**found = False**

**population = []**

**# create initial population**

**for \_ in range(POPULATION\_SIZE):**

**gnome = Individual.create\_gnome()**

**population.append(Individual(gnome))**

**while not found:**

**population = sorted(population, key = lambda x:x.fitness)**

**]**

**if population[0].fitness <= 0:**

**found = True**

**break**

**# Otherwise generate new offsprings for new generation**

**new\_generation = []**

**s = int((10\*POPULATION\_SIZE)/100)**

**new\_generation.extend(population[:s])**

**# From 50% of fittest population, Individuals**

**# will mate to produce offspring**

**s = int((90\*POPULATION\_SIZE)/100)**

**for \_ in range(s):**

**parent1 = random.choice(population[:50])**

**parent2 = random.choice(population[:50])**

**child = parent1.mate(parent2)**

**new\_generation.append(child)**

**Output Observed for different Inputs:**

* **Input passed:**

See Code

* **Output Obtained:**

Generation: 1 String: tO{"-?=jH[k8=B4]Oe@} Fitness: 18

Generation: 2 String: tO{"-?=jH[k8=B4]Oe@} Fitness: 18

Generation: 3 String: .#lRWf9k\_Ifslw #O$k\_ Fitness: 17

Generation: 4 String: .-1Rq?9mHqk3Wo]3rek\_ Fitness: 16

Generation: 5 String: .-1Rq?9mHqk3Wo]3rek\_ Fitness: 16

Generation: 6 String: A#ldW) #lIkslw cVek) Fitness: 14

Generation: 7 String: A#ldW) #lIkslw cVek) Fitness: 14

Generation: 8 String: (, o x \_x%Rs=, 6Peek3 Fitness: 13

.

.

.

Generation: 29 String: I lope Geeks#o, Geeks Fitness: 3

Generation: 30 String: I loMe GeeksfoBGeeks Fitness: 2

Generation: 31 String: I love Geeksfo0Geeks Fitness: 1

Generation: 32 String: I love Geeksfo0Geeks Fitness: 1

Generation: 33 String: I love Geeksfo0Geeks Fitness: 1

Generation: 34 String: I love GeeksforGeeks Fitness: 0