Genetic Algorithm

Basic GA Terminology

Since GA is inspired by the concepts of natural genetics and the Darwinian theory of evolution, they use lots of terms from biology and genetics. Since these terms are prevalent and ubiquitous across the literature on evolutionary computation, we briefly present them here in below.

individual

> A candidate solution to the problem at hand. Also called ‘Chromosome’ or ‘Genome’.

population

> A set of individuals (i.e., candidate solutions) maintained by an GA

genome

> Often used interchangeably with the term individual. Actually genome is the data structure of an individual, which is used during genetic operations, such as crossover and mutation

genotype

> same as the term genome

chromosome

> same as the genotype or genome, but represented as a fixed length vector

parent and child (offspring)

> A parent (i.e., existing individual or candidate solution) is perturbed by genetic operations (i.e., mutation, crossover/recombination) to produce a new candidate solution (child or offspring)

gene

> A gene is a particular slot position in a chromosome. In other words, each chromosome (individual) is a list (sequence) of gene values.

allele

> a particular setting of a gene

phenotype

> The phenotype of an individual is a measurement of how it performs during its fitness assessment

fitness

> The fitness of an individual is a measurement of its quality as a candidate solution to the problem at hand

fitness assessment

> The task of computing the fitness of an individual

fitness landscape

> quality function; shows the fitness of every possible individual

generation

> One complete cycle (iteration) of fitness assessment, breeding and re insertion to form the population for the next cycle. The term generation may also refer to the population produced in each such cycle

selection

> choosing individuals for breeding, usually based on their fitness

breeding or mating

> The procedure of producing one or more children (new candidate solutions) from a pool of parents (i.e., existing candidate solutions) through a process of selection and genetic operations (crossover/recombination and/or mutation)

recombination or crossover

> A form of genetic operation related to sexual breeding. A typical recombination or crossover operation selects two individuals (parents), combines or swaps their genetic information in some way to produce (usually) two new individuals (offspring).

mutation

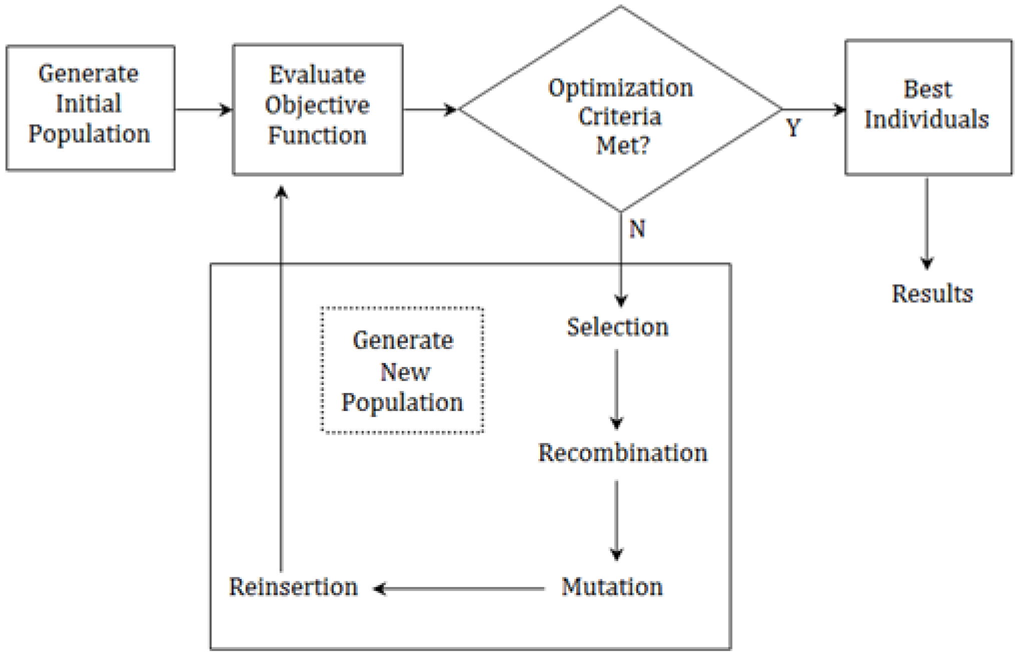
> A form of genetic operation that selects an individual (candidate solution) and randomly perturbs it to form a new individual.

re insertion

> Constructing the population of the next generation from the union of the parent and offspring solutions.

Genetic (or, Evolutionary) algorithm is an iterative, stochastic search and optimization technique based on the concepts of the Darwinian theory of evolution. GA maintains a population of individuals or chromosomes (i.e., candidate solutions) that are selected using Darwinian 'laws of natural selection' with 'survival of the fittest' and updated using operators borrowed from natural genetics like crossover, recombination and mutation. GA is an iterative algorithm — it runs generation by generation. In each generation, a typical GA employs selection, crossover/recombination and/or mutation operations to produce a pool of new candidate solutions (i.e., offspring) from the existing solutions (i.e., parents). From the union of the parents and offspring, the selection and/or reinsertion policy tries to keep the better candidate solutions and wipe out the low quality individuals during constituting the next generation population. Crossover and recombination are like 'sexual breeding' operations that produce new candidate solutions by combining information from two (or, more) different existing solutions, while mutation is like an 'asexual' genetic operation that randomly alters 'gene' value(s) of an individual.

Block diagram of a typical GA [Figure 1]



Algorithm 2.1: **A typical GA**

1. **Initializatio**n. Generate a random population of n chromosomes (individuals). Each chromosome is a candidate solution for the problem.

2. **Fitness Evaluation**. Evaluate the fitness f(x) of each chromosome x of the population.
3. **Generate new population**. Create a new population by repeating the following steps until the new population is complete.

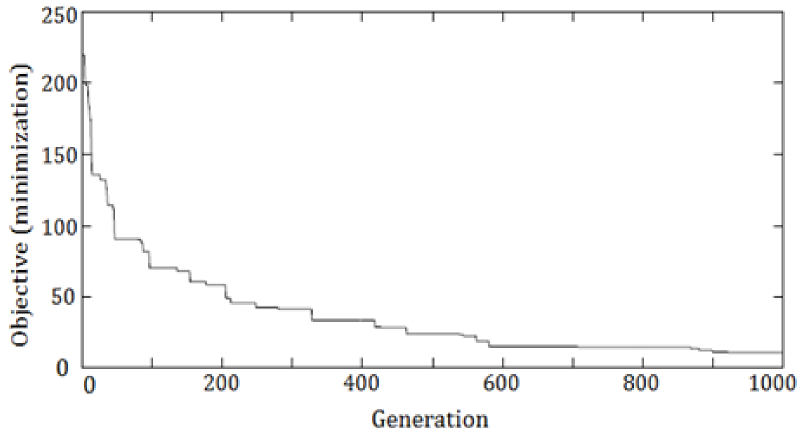
   3.1. **Selection**. Select two parent chromosomes from the population according to their fitness values (the better the fitness, the bigger chance to be selected).

   3.2 **Recombination (or, Crossover)**. Combine the gene values of the two parents to produce new offspring.

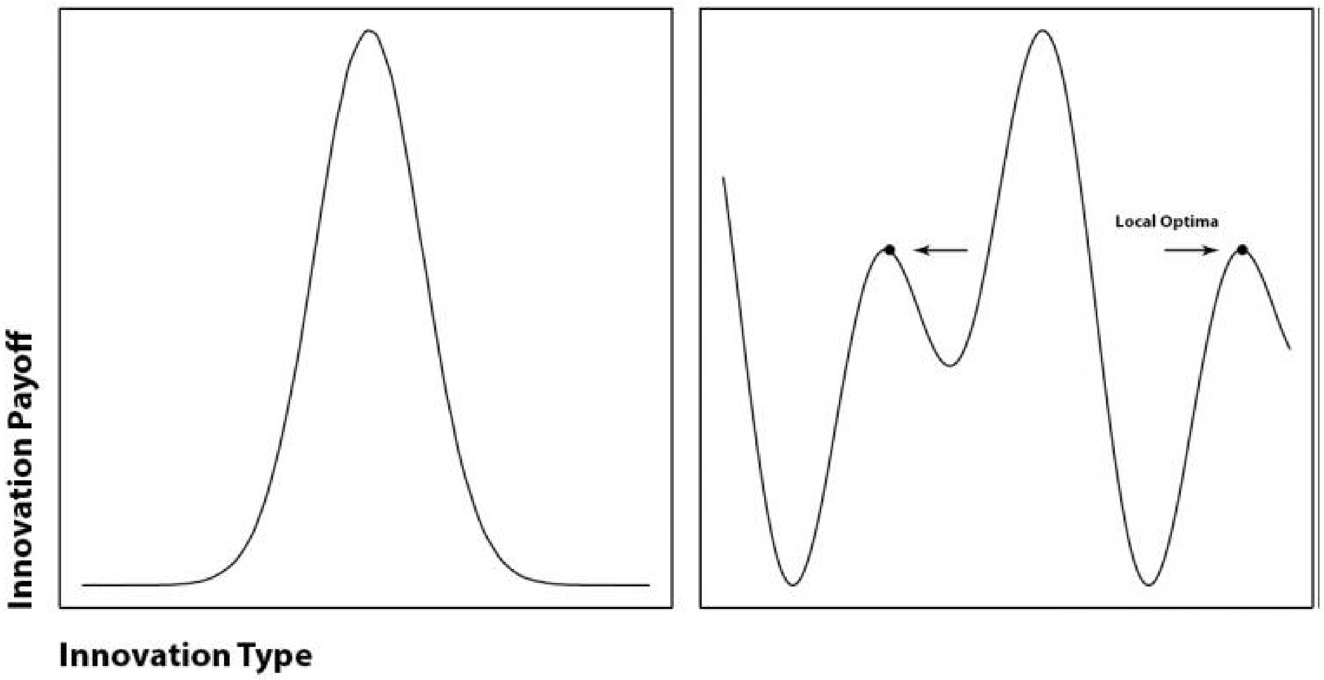
   3.3 **Mutatio**n. With a mutation probability, mutate the new offspring at each gene/locus (position in its chromosome).

   3.4 **Reinsertion**. Either accept or reject the new offspring in a new population.
4. **Test for termination**. If the stopping criteria are satisfied, then stop and return the best solution in the current generation population. Otherwise, continue.
5. Loop back. Go to the step 2 to continue the evolution with the new population of candidate solutions.

[Figure 2]



Gradual improvement of the objective value during the run of a typical GA. Here, the objective was the minimization of a continuous function.



Exploration vs Exploitation

In each generation of a typical GA, there are four operations — selection, crossover (or, recombination), mutation and re insertion. Selection picks the individuals that go through the genetic perturbations by crossover (or, recombination) and mutation operations, followed by the re insertion scheme that selects high quality individuals for the next generation and weeds out the low quality solutions. The selection and re insertion operations are usually based on fitness and hence responsible for search space exploitations, while the perturbation operations (crossover/recombination and mutation) usually perform the explorations of the search regions. Fig. 1 shows the structure of a typical Genetic/evolutionary algorithm, while algorithm 2.1 presents the pseudo code of a typical GA. The algorithm starts with a population of individuals, each representing a candidate solution of the problem at hand. Each generation of solutions passes through the evolutionary process of selection, crossover/recombination and/or mutation operations to generate new offspring solutions. The next generation is constructed by selecting good quality individuals from the union of the parents and offspring. This process continues again and again until some stopping criteria are met, which may be based on some predefined maximum number of generations, some maximal number of fitness evaluations or reaching some desired fitness or objective function value. During the run, the fitness value of the best found individual usually improves over time, generation by generation. This fitness improvement may stagnate at the intermediate locally optimal points or during the end of the run, as demonstrated in Fig. 2. In the ideal case, the fitness stagnation would happen with the successful finding of the global optimum. However, stagnation may also occur around a strong locally optimal point, which can lead to premature convergence of the optimization procedure with insufficient solution quality, which is one of the major problems of EAs (and SIAs, too).

Components of GA

GA/EA-s have a number of component procedures and operators that must be described to have a proper comprehension of the algorithm. When solving a particular problem using GA, each of these components is required to be specified precisely in order to describe the GA. The essential **SIX components of a GA** are as follows.

- Representation (Encoding)
- Fitness (or, evaluation) function
- Selection
- Recombination/Crossover
- Mutation
- Reinsertion

Before explaining these terms further, we will see a concrete example — How an optimization Problem can readily be solved by the Genetic Algorithm.

Traveling Salesman Problem

Given a list of cities and the distances between each pair of cities, what is the `shortest possible route` that visits each city exactly once and returns to the origin city?

Note:

- n cities … pairwise distance known
- Visit Each City Exactly Once (You Can't Miss any city, and Can't visit any city more than once)
- Start and End at the same city (so, it's a `cycle` or loop, not open ended path)

- The total distance travelled will be minimum over all possible such routes

Search Space Size for TSP

Each ordered list of n-cities represent a solution. This means, any permutation of the n-cities is a possible solution (candidate solution). Total no. of possible permutations over n cities is n! (Which is a huge number, even for small n. For example, `for n=20 cities, total no. of permutations is 20! = 2432902008176640000`, which would take forever to compute even by Today's modern computers!) The Brute Force Algorithm will try to produce all possible permutations of n cities, compute the total tour length and choose the minimum-length tour, which is not practical even for small n — as small as n=20.

Solving TSP by a Concrete Genetic Algorithm (CGA):

The Following GA will be referred as "Concrete Genetic Algorithm" for future reference

• Representation: An individual (i.e., chromosome or candidate solution) is represented as an Ordered-List of n-cities. Example: An 8-city problem (see in PDF, page/slides 33-38)

> Note: This is an individual Only, The Population contains Many such individuals, each one randomly generated at the initial generation (Generation counter gen = 0). However, for the later generations, the new offspring solutions are produced by crossover and mutation operation, as described below.

> For this concrete GA example, select population size NP = 100 (i.e., the population contains 100 individuals, i.e., candidate solutions)

• Fitness Function Evaluation:

> Calculate the Fitness of Every individual of the current population. The fitness of an individual is inversely proportional to its Tour length. The smaller the TSP tour length, the Higher the fitness Value (because, this is a minimization problem, where the objective is to Find the minimum length TSP Tour). Hence, the fitness of an individual x_i can be estimated as: $fitness(x_i) = \frac{1}{TourLength(x_i)}$.

• Selection:

> Select One Parent by using Roulette Wheel Selection Scheme which ensures Fitness Proportional Selection (i.e., shorter Tours get higher probability to be selected). The Another Parent by using Tournament Selection (tournament size = 3). This parent becomes the partner of the first parent selected above Repeat the above Two steps NP/2 (=population size= 100) times to get NP/2 pairs of parents

• Crossover :

> Apply Davis' Order Crossover (OX1) Crossover between Each Pair of Parents to produce a Pair of Offspring (i.e., new candidate solutions). NP/2 pairs of Parents produce NP/2 pairs offspring in total. Details: page 35 in TSP.pdf

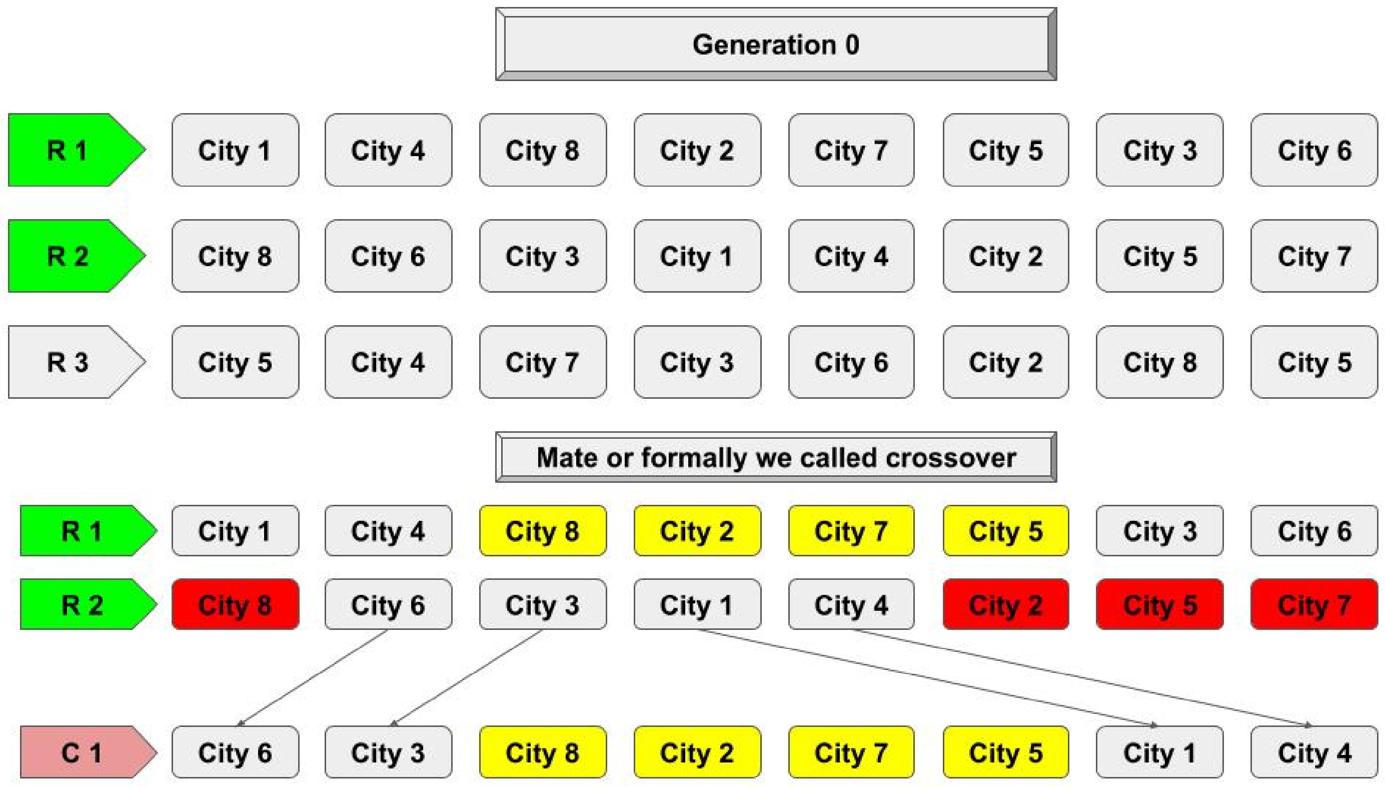
• Mutation:

> Each Offspring is Mutated by using Swap Mutation (with mutation probability pm = 0.1). Details: page 36 in TSP.pdf

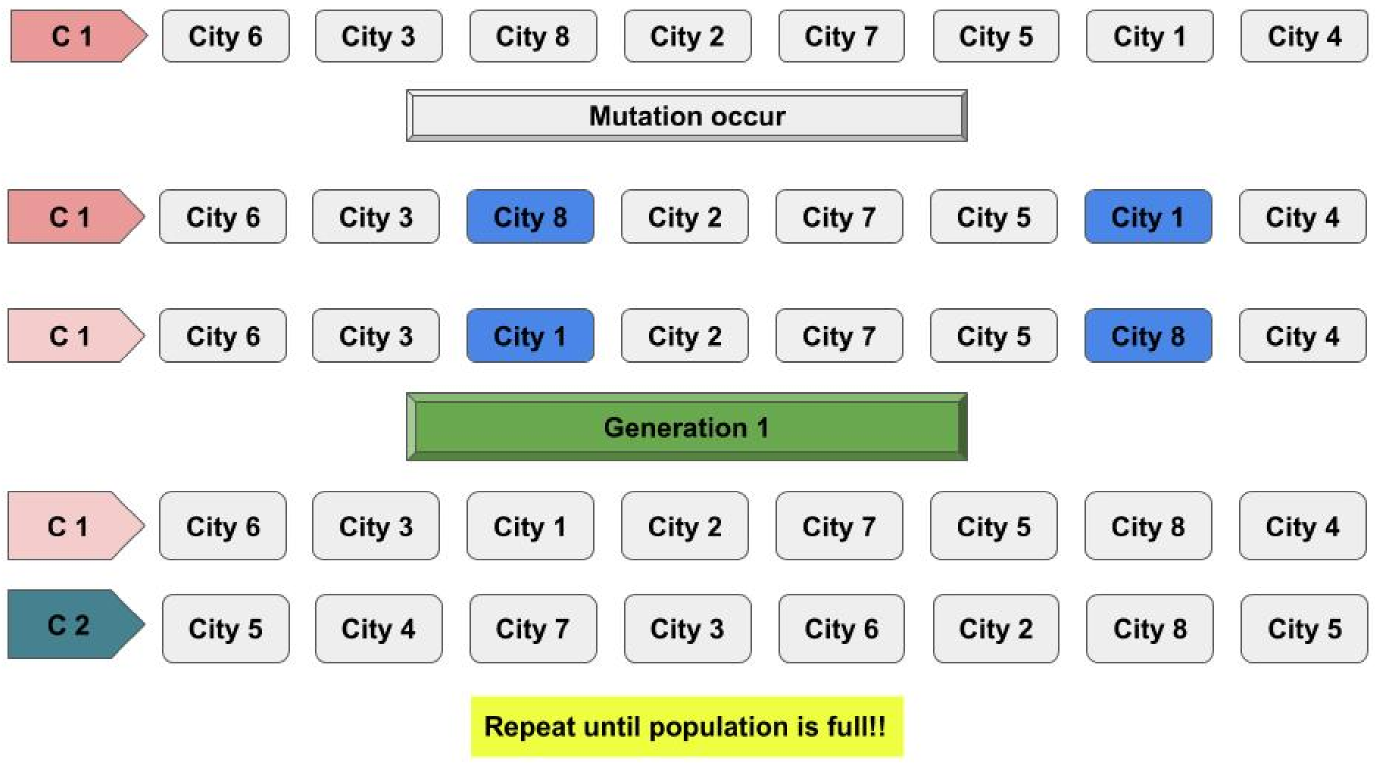
• Re-Insertion:

> Over the combination of (NP+NP) Parents and Offspring individuals, perform the following Tournament based selection scheme:
>
> > For each individual, q (=10) opponents are chosen uniformly at random from all the parents and offspring. Perform a pairwise comparison between each individual and its opponents. In each comparison, if the individual's fitness is no smaller than the opponent's, it receives a "win."

> Sort all the individuals (2*NP in total) based on their no. of wins and select first NP individuals who received the highest no. of wins. They are the survivors who form the next Generation of Population.
> Repeat the Above Procedure (Fitness evaluation, Selection, CrossOver, Mutation and Re-Insertion) repeatedly, until we are run out of time. Then, return the best individual Found so far.


Generation 0
R 1 City 1 City 4 City 8 City 2 City 7 City 5 City 3 City 6
R 2 City 8 City 6 City 3 City 1 City 4 City 2 City 5 City 7
R 3 City 5 City 4 City 7 City 3 City 6 City 2 City 8 City 5
Mate or formally we called crossover
R 1 City 1 City 4 City 8 City 2 City 7 City 5 City 3 City 6
R 2 City 8 City 6 City 3 City 1 City 4 City 2 City 5 City 7
C 1 City 6 City 3 City 8 City 2 City 7 City 5 City 1 City 4

Before Mutation


C 1 City 6 City 3 City 8 City 2 City 7 City 5 City 1 City 4
Mutation occur
C 1 City 6 City 3 City 8 City 2 City 7 City 5 City 1 City 4
C 1 City 6 City 3 City 1 City 2 City 7 City 5 City 8 City 4
Generation 1
C 1 City 6 City 3 City 1 City 2 City 7 City 5 City 8 City 4
C 2 City 5 City 4 City 7 City 3 City 6 City 2 City 8 City 5
Repeat until population is full!!