Index

Hemin Joshi
joshi.he@husky.neu.edu

Juhi Pareek
pareek.j@husky.neu.edu

Aditya Soni
soni.ad@husky.neu.edu

Introduction

Overview

This application simply evolves random strings from a finite population to a target string using Genetic Algorithm. The program itself is really quite simple and more complex concepts like crossover using roulette wheel algorithms, insertion/deletion mutation aren't included. The random password is generated as a result of an early evolution with a greater fitness than a late evolution with a lower fitness.

Problem Statement

A brute-force password cracking attack might well suss out a password like "Superman", but it will fail to guess in finite time when the password is something like jio9-md@5-7$ol-qma8. Using a random password generator isn't completely secure. Most of the random password generators use a pseudo-random algorithm. In theory, a hacker who knows the algorithm and has access to one of the previously generated passwords can predict all of the subsequent passwords (quite difficult). Our goal is to build a password generator that doesn't let anyone predict the future password even if they have access to any previous randomly generated passwords.

Solution

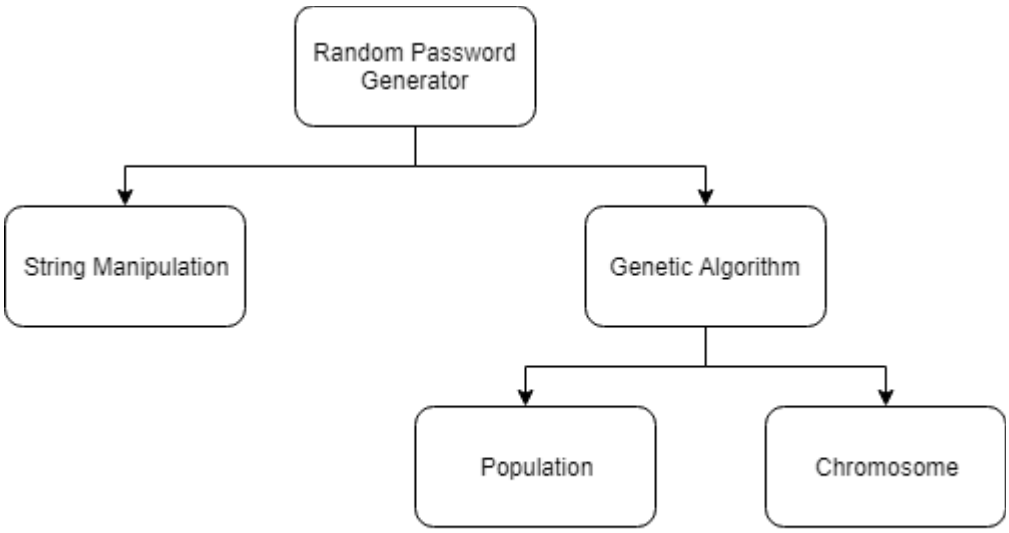


Fig. 1.1 Basic Structure

The overall architecture for the application is divided into two parts - String manipulation and Genetic Algorithm. The Genetic Algorithm consists of two units - Population and Chromosome.

Hemin Joshi joshi.he@husky.neu.edu
Juhi Pareek pareek.j@husky.neu.edu
Aditya Soni soni.ad@husky.neu.edu

Theory

Population

A population is simply a sorted collection of Chromosome (sorted by fitness) that has a method for evolution (evolve()). This implementation of a population uses a tournament selection algorithm for selecting parents for crossover during each generation's evolution. Along with a collection of Chromosome instances, the Population consists of three key attributes namely the crossover ratio, the mutation ratio and an elitism ratio (0 <= crossover ratio, mutation ratio, elitism ratio <= 1). The problem with elitism is that it causes the GA to converge on local maxima instead of the global maximum, so pure elitism is just a race to the nearest local maximum and you'll get little improvement from there. You can fix this in a few ways, two of which are (a) mutate the elite randomly, i.e. introduce entropy (b) go with second or third decile alleles/chromosomes also randomly.

Evolution

The evolve() method uses the crossover ratio, the mutation ratio and an elitism ratio during the evolution process. Elitism ratio involves copying a small proportion of the fittest candidates, unchanged, into the next generation. Based on the crossover ratio, the remaining chromosomes are either copied directly or mated with other chromosomes in the population. The mutation ratio enables random mutation in each of these chromosomes.

Chromosome

A gene represents a possible solution to a given problem. Every chromosome has the gene. In this application, each gene strives to match the target string. The fitness attribute is computed as the measure of proximity of the gene to the target string. Fitness is a sum of the absolute difference of each character in the gene to the corresponding character in the target string. In other words, fitness is defined as being the sum of the absolute value of the difference between the current gene and the target gene. Each gene is a string of ASCII characters from ASCII 32 to ASCII 121. The mutate() method randomly replaces one character from the gene while the mate() takes a chromosome instance to return two new chromosomes. This method randomly selects one character in the Chromosome's gene and then replaces it with another random (but valid) character. This method returns a new Chromosome, it does not modify the existing Chromosome. The mate() method takes parent chromosome instances and returns two new child chromosome instances.

The program outputs the generated string near to the target string in the console to demonstrate how genetic algorithms can be used to reach near optimum solutions.

Hemin Joshi joshi.he@husky.neu.edu
Juhi Pareek pareek.j@husky.neu.edu
Aditya Soni soni.ad@husky.neu.edu

Random Password Generation

Random Password Generators use random numbers to select a character. Some random password generators use an input file to generate characters by reading lines. The developed system uses Genetic Algorithm to generate random passwords.
User Inputs like First Name, Last Name, Date of Birth, Email and Phone Number are taken and concatenated to a line which is taken from a file. The line is randomly selected from the file. The resultant string is our target string which the genetic algorithm tries to achieve in *n* generations. This is done by the genetic algorithm using selection, crossover, mutation.

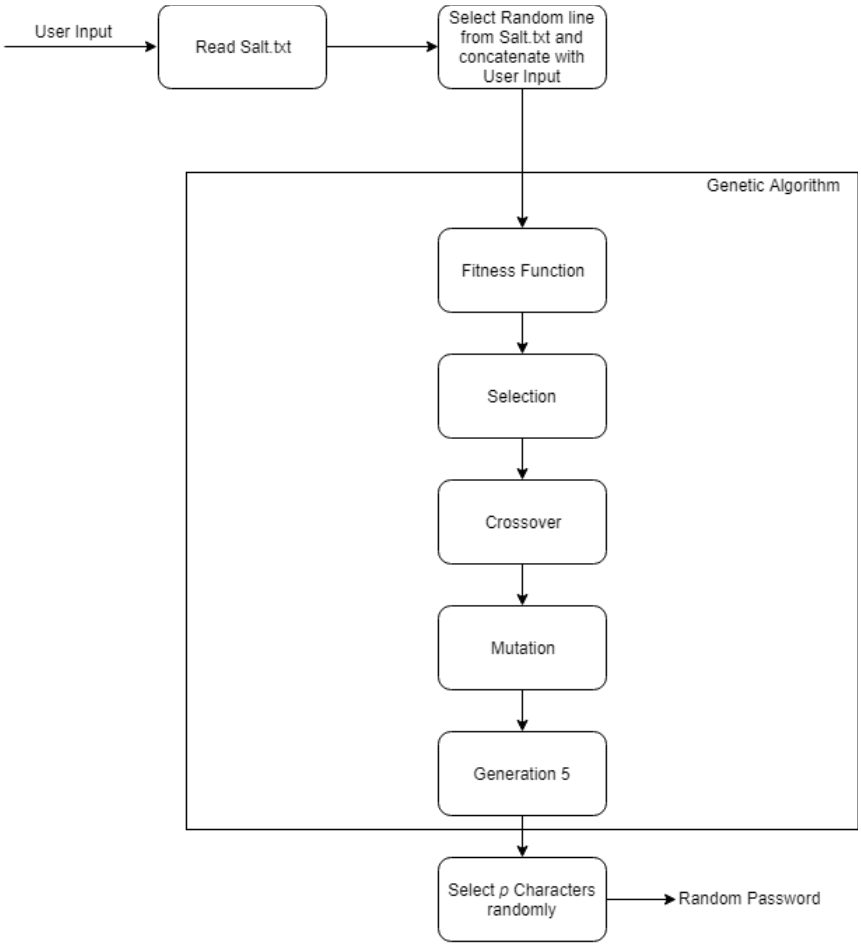
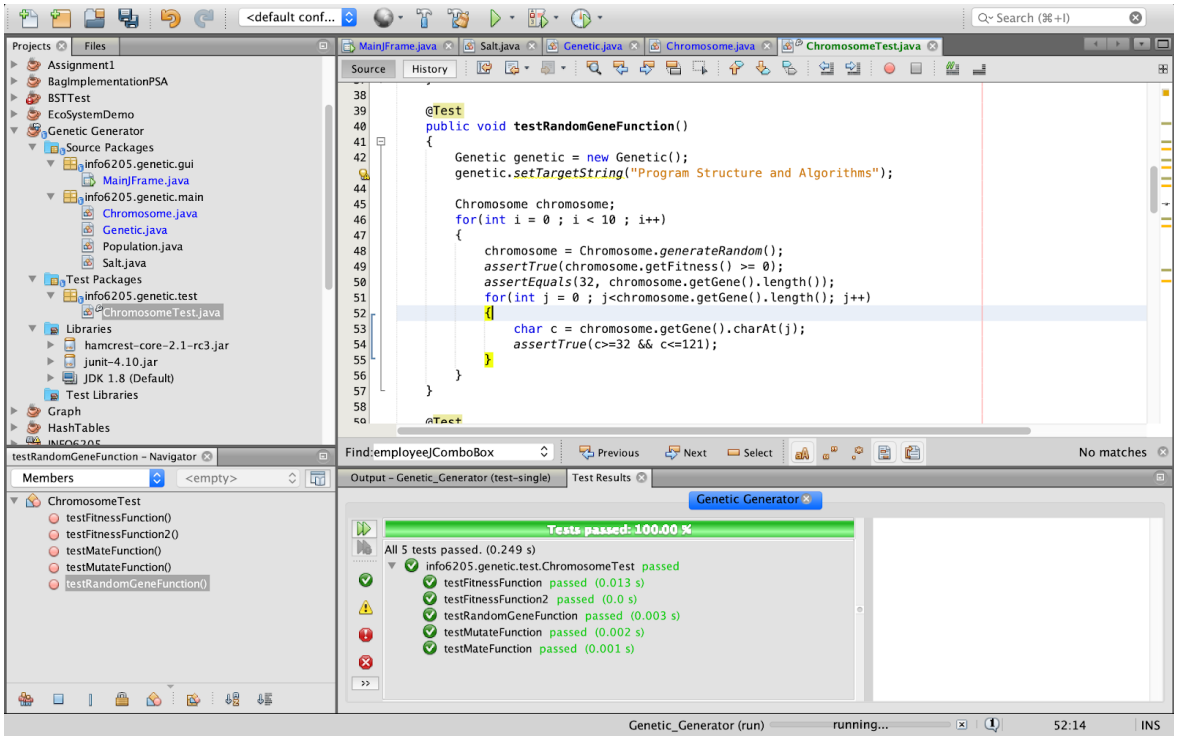


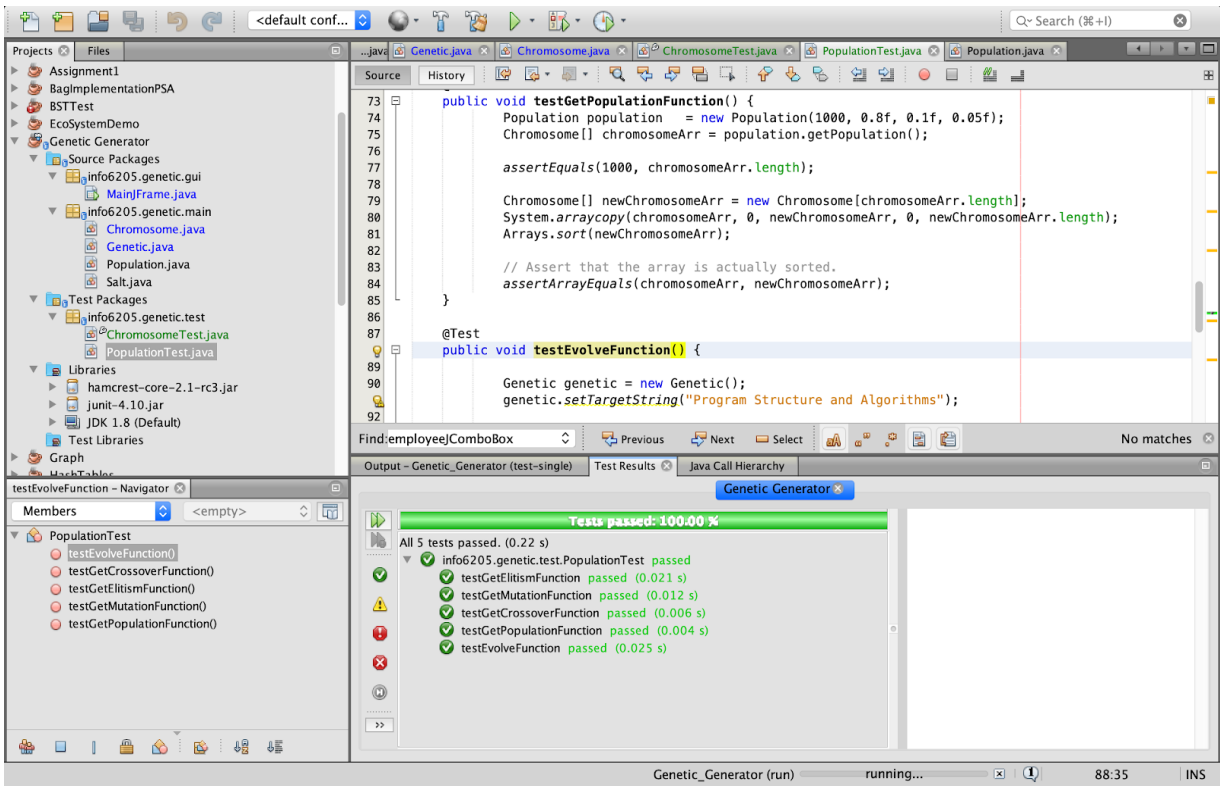
Fig. 3.1 Random Password Generation

We select the fifth generation (no logic behind selecting this number, completely random) as our root string. The user then selects the number of characters *p* for the password. The system then selects *p* random indexes from the root string to generate a string which becomes our random password. This method of generating passwords randomly can be used in browsers for sign up forms.

Hemin Joshi joshi.he@husky.neu.edu
Juhi Pareek pareek.j@husky.neu.edu
Aditya Soni soni.ad@husky.neu.edu

Tests





Hemin Joshi
joshi.he@husky.neu.edu

Juhi Pareek
pareek.j@husky.neu.edu

Aditya Soni
soni.ad@husky.neu.edu

Conclusion

Genetic Algorithm is a probabilistic stochastic search algorithm. Even if all the parameters are the same, the results are not the same for each operation. These algorithms are designed for those problems which cannot be solved otherwise or it takes more than acceptable time to solve. These solutions are near optimal solutions. Optimum solutions maybe unique if no other identical solution exists. But near optimum solutions are plenty. By varying the parameters of the algorithm, improved solutions can be achieved.

The Password strength of a random password against a particular attack (brute-force search), can be calculated by computing the information entropy of the random process that produced it. If each symbol in the password is produced independently and with uniform probability, the entropy in bits is given by the formula

$$H = L \frac{log\, N}{log 2}$$

where N is the number of possible symbols and L is the number of symbols in the password. The function $\log_2$ is the base-2 logarithm. H is typically measured in bits.

Hemin Joshi
joshi.he@husky.neu.edu

Juhi Pareek
pareek.j@husky.neu.edu

Aditya Soni
soni.ad@husky.neu.edu

References

1. An Introduction to Genetic Algorithms, Jenna Carr
2. A Random-Key Genetic Algorithm for the Generalized Traveling Salesman Problem, Lawrence V. Snyder, Mark S. Daskin
3. GAHelloWorld, John Svazic
4. Applications of genetic algorithms, Peter Ross, Dave Corne
5. A Genetic Algorithm With Self-Generated Random Parameters, Sonja Novkovic and Davor Sverko

Hemin Joshi joshi.he@husky.neu.edu
Juhi Pareek pareek.j@husky.neu.edu
Aditya Soni soni.ad@husky.neu.edu