

Genetic Algorithm:
Substitution Cipher Text Decryption

by

Team 231

Apurva Arvind Bakshi (001430557)

Shayesta Parveen Reehan (001401028)

Table of contents

Introduction	3
Problem Statement	4
Implementation and Output.....	6
Conclusion	9
References.....	10

PROBLEM STATEMENT

In this project we aim to apply genetic algorithm to the problem decrypting text that has been encrypted using a substitution cipher. We use the property of genetic algorithm that relies on multiple search space to converge to the solution for our NP complete problem.

INTRODUCTION

SUBSTITUTION CIPHER

Substitution cipher replaces every instance of a particular letter in the plain text with a different letter from the cipher text. A substitution cipher key is a set of one-to-one mappings relating every letter in the plain text alphabet with the corresponding letter in cipher text alphabet.

Example of substitution cipher key we used in our algorithm is,

alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
key	p	h	q	g	i	u	m	e	a	y	l	n	o	f	b	x	j	k	r	c	v	s	t	z	w	b

Using above key, each letter in the plaintext is replaced by the key letter to create encrypted text.

Plain Text: the quick brown fox jumps over the lazy dog

Cipher Text: cei jvaql hkdtef udz yvoxr dsik cei npbw gdm

To decrypt this cipher, we have 26 possibilities to choose for the first letter, 25 for the second, 24 for the third etc. So, there are total 26! Possible keys to decrypt this cipher text.

$$26! = 4.0329146e+26$$

We will need approximately 13,000,000,000 number of years to resolve this problem using different combinations of keys.

However, this cipher is particularly vulnerable to a technique known as frequency analysis since although it does change the letters in the plain text to different ones in the cipher text it does not change the underlying frequency of those letters.

Hence, you can group letters into n-grams where n represents the number of letters in the n-gram and look up the corresponding frequencies for those.

GOAL

We apply the technique of genetic algorithms to the problem of finding the key for a Substitution Cipher.

IMPLEMENTATION DETAILS

GENETIC ALGORITHM

We are using genetic algorithm to resolve our problem.

1. Population

Population is a set of chromosomes which is a part of one generation. In our case, each chromosome is represented as a key to decrypt cipher text. So, example of chromosome can be given as a random key.

Chromosome = "yhexasarblptudvqwjiknfczogm"

2. Fitness function

Fitness function calculates score for each chromosome. Score of each chromosome is calculated using Fitness weights which are provided as an input for our algorithm.

For example,

In the plain text string,

"the quick brown fox jumps over the lazy dog"

We have 2-Grams and 3-Grams with weights as below:

n-Gram	Th	He	Er	Ov	Zy	Dog	Laz	The	ox
Weight	2	2	1	1	1	1	1	1	1

Fitness weights are multiplied by number of times the occurrence of the n-Gram in the decrypted text.

$$\text{Score} = \text{n-Gram count} * \text{Fitness weight of n-Gram}$$

3. Selection

After calculating fitness score of each chromosome, we find two new chromosome using sum of all fitness values and random number calculations. These two new chromosomes can be used to create two chromosomes for next generation.

4. Crossover

After selecting two chromosomes as parents, we crossover them to generate new child chromosomes. Crossover is decided using crossover probability value. Technique for crossover used is as below:

- Choose p points of crossover.
- Copy values of those p points from first parent into p points in the first child.
- Then iterate through remaining values from second parent filling in missing values in first child.
- Follow same process to generate second.

5. Mutation

After crossover, we mutate newly produced chromosomes using below technique:

- Select two random points or indexes from each chromosome.
- Swap letters at these index

Mutation step is decided from mutation probability value.

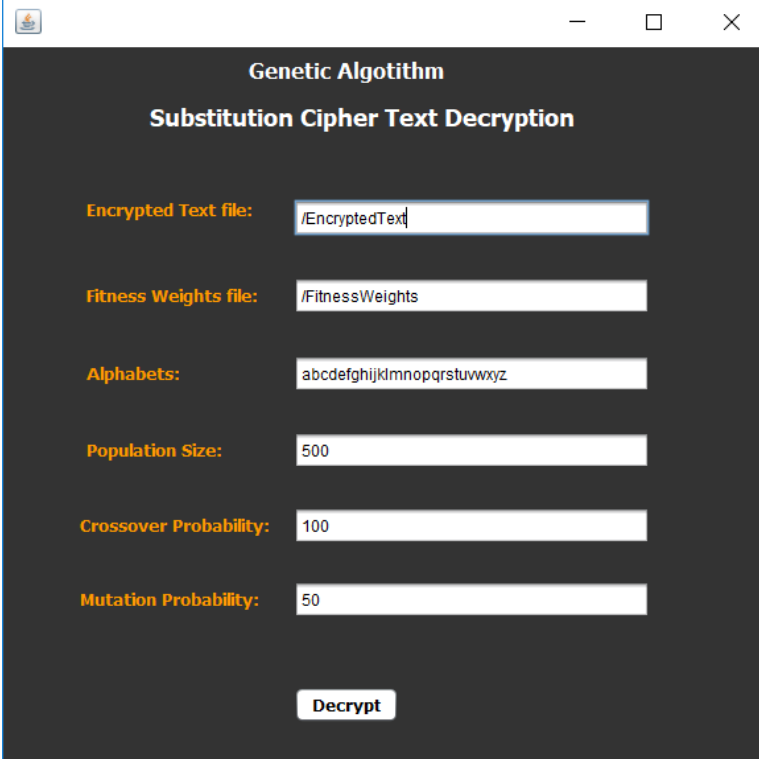
6. Elitism

To ensure the survival of individuals with high fitness values and to increase the average fitness of the each generation we use Elitism to copy a fixed percentage of the fittest individuals from the current population into the next generation.

IMPLEMENTATION AND OUTPUT

We have implemented this genetic algorithm using Java and added user Interface to input population size and probabilities.

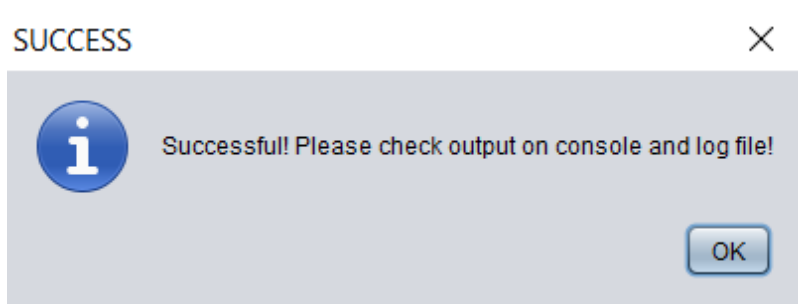
User Interface



The screenshot shows a Java Swing window titled "Genetic Algorithm" with a subtitle "Substitution Cipher Text Decryption". The window has a dark gray background and contains several input fields with orange labels:

- Encrypted Text file:** A text box containing the text "/EncryptedText".
- Fitness Weights file:** A text box containing the text "/FitnessWeights".
- Alphabets:** A text box containing the text "abcdefghijklmnopqrstuvwxyz".
- Population Size:** A text box containing the text "500".
- Crossover Probability:** A text box containing the text "100".
- Mutation Probability:** A text box containing the text "50".

At the bottom center of the window is a white button with the text "Decrypt".



Console Output

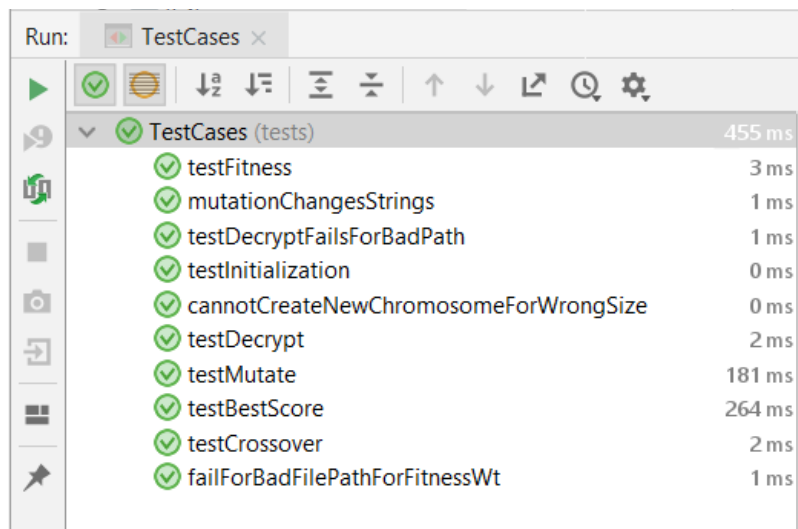
```
Best chromosome index:446
the iabrv kfoml yox pajuz owef the cnsq dog
-----
Generation:994
Best score:26
Best chromosome index:392
the iabrw kfoml yox pajuz ovef the cngq dos
-----
Generation:995
Best score:26
Best chromosome index:36
the iabrw kfoml yox pajuz ovef the cngq dos
-----
Generation:996
Best score:28
Best chromosome index:352
the baipw nqosl yox jafuz oveq the ckrm dog
-----
Generation:997
Best score:24
Best chromosome index:133
the baipw nqosl yox jafug oveq the ckrm doz
-----
Generation:998
Best score:24
Best chromosome index:245
the icarv kfoml yox bojuz owef the pnsq dog
-----
Generation:999
Best score:26
Best chromosome index:72
the qjanx plorf yoz ljmwu ovei the bcsk dog
-----
```

Log File output

We have used logger to keep track of the events of each generation.

```
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Initialzld population with generation size:500
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Chromosome with best score is at index:384 and best score:7
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Output using best chromosome: oqt flawn zheym cek vljzp exth oqt dgui ber
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - New generation created with Generation number:1
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Chromosome with best score is at index:472 and best score:5
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Output using best chromosome: pwh ovqms djzrl aze nvxiy zthj pwh bcku fzg
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - New generation created with Generation number:2
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Chromosome with best score is at index:478 and best score:5
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Output using best chromosome: hzy jdaet figrp cgn kdovx gsyi hzy umlw bgq
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - New generation created with Generation number:3
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Chromosome with best score is at index:242 and best score:7
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Output using best chromosome: wax iynmk plocj dou zyheq ogxl wax svbr fot
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - New generation created with Generation number:4
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Chromosome with best score is at index:147 and best score:8
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Output using best chromosome: uaq iwnmk llocj doy rwhet ogql uaq psbx fov
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - New generation created with Generation number:5
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Chromosome with best score is at index:258 and best score:9
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Output using best chromosome: uaq itnmk wlocj dov rthex ogql uaq psby foz
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - New generation created with Generation number:6
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Chromosome with best score is at index:181 and best score:10
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Output using best chromosome: zaq itnmk wlocj dov rthey ogql zaq psbu fox
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - New generation created with Generation number:7
[AWT-EventQueue-0] DEBUG textdecryptga.GeneticAlgorithm - Chromosome with best score is at index:349 and best score:9
```

Test cases output



The screenshot shows the 'Run' window of an IDE, specifically the 'TestCases' tab. The window displays a list of test cases, each with a green checkmark icon indicating a successful pass. The test cases are listed in a table with two columns: the test name and the execution time in milliseconds (ms). The total execution time for all tests is 455 ms.

Test Case	Execution Time (ms)
TestCases (tests)	455 ms
testFitness	3 ms
mutationChangesStrings	1 ms
testDecryptFailsForBadPath	1 ms
testInitialization	0 ms
cannotCreateNewChromosomeForWrongSize	0 ms
testDecrypt	2 ms
testMutate	181 ms
testBestScore	264 ms
testCrossover	2 ms
failForBadFilePathForFitnessWt	1 ms

CONCLUSION

We decrypted the text to some extent when we used size of population as 500 and number of generations 1000. That means closest key to the actual is found. The example encrypted text was converged to this solution:

Plain Text: "the quick brown fox jumps over the lazy dog"

Cipher Text: "cei jvaql hkdtf udz yvoxr dsik cei npbw gdm"

Genetic Algorithm solution: "the qjanx piorf yoz ljmwu ovei the bcsk dog"

The results were quite expected as to resolve problem with $26!$ Possibilities, you will need approximately 13,000,000,000 number of years using brute force method.

REFERENCES

- <https://people.cs.uct.ac.za/~jkenwood/JasonBrownbridge.pdf>
- <http://practicalcryptography.com/ciphers/simple-substitution-cipher/>