Fall 2015

Cryptanalysis of the Purple Cipher using Random Restarts

Aparna Shikhare
San Jose State University

Recommended Citation
Shikhare, Aparna, "Cryptanalysis of the Purple Cipher using Random Restarts" (2015). *Master's Projects*. 428.
DOI: https://doi.org/10.31979/etd.tcqp-x6sz
https://scholarworks.sjsu.edu/etd_projects/428

Cryptanalysis of the Purple Cipher using Random Restarts

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Aparna Shikhare

December 2015

The Designated Project Committee Approves the Project Titled

Cryptanalysis of the Purple Cipher using Random Restarts

by

Aparna Shikhare

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

December 2015

Dr. Thomas Austin	Department of Computer Science
Dr. Sami Khuri	Department of Computer Science
Dr. Robert Chun	Department of Computer Science

Abstract

Cryptanalysis of the Purple Cipher using Random Restarts

by Aparna Shikhare

Cryptanalysis is the process of trying to analyze ciphers, cipher text, and crypto systems, which may exploit any loopholes or weaknesses in the systems, leading us to an understanding of the key used to encrypt the data.

This project uses Expectation Maximization (EM) approach using numerous restarts to attack decipherment problems such as the Purple Cipher. In this research, we perform cryptanalysis of the Purple cipher using genetic algorithms and hidden Markov models (HMM). If the Purple cipher has a fixed plugboard, we show that genetic algorithms are successful in retrieving the plaintext from cipher text with high accuracy. On the other hand, if the cipher has a plugboard that is not fixed, we can decrypt the cipher text with increasing accuracy given an increase in population size and restarts. We performed the cryptanalysis of PseudoPurple, which is less complex but more powerful than Purple using HMMs. Though we could not decrypt cipher text produced by PseudoPurple with good accuracy, there is an increase in accuracy of the decrypted plaintext with an increase in the number of restarts.

Acknowledgements

I am very thankful to my advisor Dr. Thomas Austin for his continuous guidance and support throughout this project and believing in me. Also, I would like to thank the committee members Dr. Sami Khuri and Dr. Robert Chun for monitoring the progress of the project and their valuable time. Special thanks to Dr. Mark Stamp for his valuable tips and advice on applying hidden Markov models to PseudoPurple.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Purple machine was a complex machine used to encrypt data not only in the 1930s, but even today. It falls under the category of homophonic substitution ciphers. A homophonic substitution cipher is a substitution cipher where a single plaintext letter can be replaced by any of the different cipher text letters. The Purple machine could essentially replace a single letter with a series of letters of hundreds of thousands in length before it would start repeating the same substitution of letters. Purple was a diplomatic cryptographic machine used by the Japanese Foreign office, just before and during World War II. The machine is an electromechanical stepping switch device, which has mainly three major components [7]. First, the electric typewriter is used to input information to the machine. The second part is a ‘‘cryptographic assembly" that consists of a plug board, four electric coding rings, and numerous wires and switches, which map the plaintext letter to various letters to produce the final cipher text character. The last part is an output unit that prints the encrypted message from the machine [8].

In this project, we propose to decipher the homophonic substitution cipher such as the Purple cipher using two approaches. The first approach is using genetic algorithms and the second approach is hidden Markov models (HMM).

Genetic algorithms are metaheuristics that use the principles of biology to search through the candidates and find a solution to the optimization and search problems [5]. It is inspired by natural evolution and uses techniques such as inheritance, mutation, selection, and crossover. A hidden Markov model is a Markov model that has hidden (unobserved) states. In HMMs, the state is not directly visible, but a series of tokens or

observations, which indicate certain information about the states, are visible. The states are related to the observations with discrete probability distributions.

We used genetic algorithms to decrypt the cipher text produced by Purple with a fixed plugboard. We could successfully decipher Purple with 100% accuracy. We also performed cryptanalysis on Purple without a fixed plugboard. With this, we saw that the accuracy of the decrypted plaintext improves with an increase in population size and number of restarts.

PseudoPurple is similar to Purple, except for the sixes-twenties split. That is, the plugboard is not partitioned into sixes and twenties. This makes PseudoPurple very powerful in terms of the effort required to brute-force the cipher. However, since there is no split of the plugboard into sixes and twenties, it is very straightforward and less complex in terms of its operation and configuration. Due to its simplicity, we used HMMs to obtain plaintext from cipher text. Though the accuracy is low, the result improves with an increase in the number of random restarts.

The report is organized as follows. In Chapter 2 we talk about the basics of cryptography and different types of ciphers. We briefly explain the significance of Purple in World War II. Then, we give details of its configuration and implementation. After this, we look at a slightly different model of Purple called PseudoPurple. We also discuss the background of genetic algorithms and hidden Markov models. In Chapter 3 we discuss cryptanalysis of Purple with genetic algorithms and also outline the results obtained from it. In Chapter 4 we describe HMM approach to decipher the cipher text produced by PseudoPurple. We summarize the results obtained from this approach as well. In Chapter 5 we conclude the report followed by future work and enhancements in Chapter 6.

Chapter 2

Background

2.1 Basics of Cryptography

2.1.1 Cryptography

Cryptography is the conversion of the text in readable form (called the plaintext) from the sender's side into an unintelligible form (called the cipher text) to the receiver's side. This is done when the message is sensitive and should be kept a secret in the presence of third parties. Cryptography allows us to store or send sensitive data over insecure networks so that they cannot be read by anyone except the intended recipient. The cryptographic algorithm is a mathematical function used to encrypt the plaintext and decrypt the cipher text in combination with the key [9]. The same plaintext can be encrypted to different cipher text with different keys. The key is the variable that is provided as an input to the cryptographic algorithm. The key remains private and ensures secure communication. The security of the entire cryptographic scheme depends on the security of the key.

2.1.2 Cryptanalysis

Cryptography basically deals with securing of data, whereas cryptanalysis tries to break the cipher used for securing data. This requires studying and analyzing the cipher to discover any hidden aspects with a view of finding some weakness that can be exploited in order to break the message.

The ciphers can be broadly classified into two types:

1. **Simple Substitution Ciphers**

2. **Homophonic Substitution Ciphers**

In the next two sections, we will be explain the two types of ciphers in detail.

2.1.3 Simple Substitution Ciphers

The simple substitution cipher is where every plaintext character is replaced by a different cipher text character. It is very different from the Caesar cipher where the plaintext characters are just shifted by a number (key) [10]. The simple substitution cipher offers very little security and is easy to break. There is basically one to one mapping between the plaintext and the cipher text characters.
An example of simple substitution cipher is:

Plain alphabet	`abcdefghijklmnopqrstuvwxyz`
Key	`bhqgismeaylonfdxjkrcvutzwp`

Suppose the plaintext is ‘‘alice in wonderland’’, on encrypting this plaintext with the given key we get the following cipher text.

Plaintext	`aliceinwonderland`
Ciphertext	`boaqiafudfgikobfg`

2.1.4 Homophonic Substitution Ciphers

The homophonic substitution cipher is different from simple substitution cipher, in that a single plaintext letter can be encrypted by any of the different cipher text letters [11]. There are generally more difficult to break than the simple substitution ciphers. The English alphabet letter frequency count method can be used to break simple substitution ciphers, but the same technique cannot be used to perform the cryptanalysis of homophonic substitution ciphers. The usual method used to break homophonic substitution ciphers are hill climb techniques.

2.2 Historical Significance of Purple

In the early 1930s, the Japanese government purchased the commercial version of the Enigma machine from the German government in order to build an enhanced version of it. The Japanese government modified the Enigma machine to add more security. This cryptographic machine was named ‘‘Red” by the US government. This machine was one of the most secure and evolved cryptographic machines in the world. It was used by the Japanese government from 1931 to 1936 [7], but later the US Signal Intelligence Service broke the cipher. However, US could not keep the decryption of Red very secret and this made the Japanese government suspicious.

Soon after the ‘‘Red” cipher was broken, the Japanese government created a more evolved and secure cipher known as ‘‘97-shiki O-bun In-ji-ki” or ‘‘97 Alphabetical Typewriter”, named for its creation on the Japanese year 2597 in 1937 [1]. The US later named it as Purple. Unlike the Enigma machine, which used the blinking lights to represent the message, Purple used an electric typewriter, which could write the message on paper [1]. This was easy to use when compared to the Enigma machine. However, it was very bulky and heavy, due to this the Purple machine was tedious to carry in combat areas.

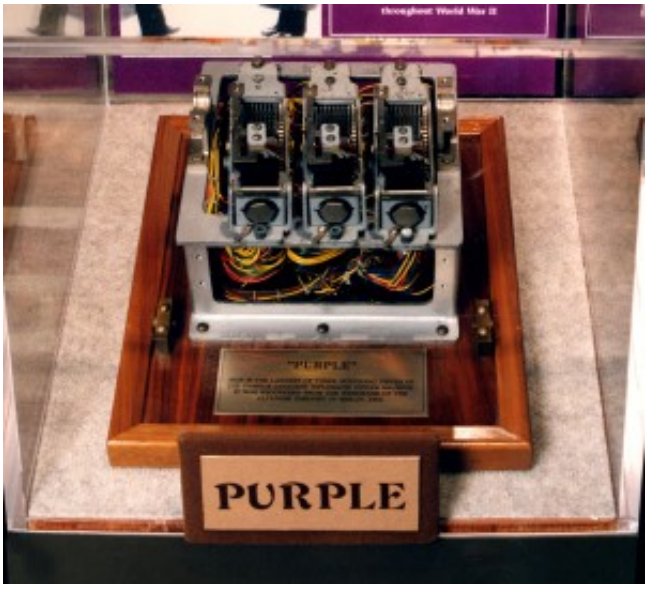

Figure 2.1: Purple machine used by Japanese government [1]

The Purple machine was used to send sensitive messages to diplomats and military officials in many places including Washington, Berlin, and London [1]. It was one of the

most complex and well developed ciphers used during that time. Though it was eventually broken by the US government, it was used to send secret messages for over two years during World War II. It required a huge amount of effort to break this cipher by US cryptanalysts. Once Purple was broken, the US government used this against the Japanese government to keep track of their activities. The US government never revealed to the Japanese government that Purple was broken and tried to keep this a secret, so that the Japanese would continue to send secret messages and this could reveal their plans of attack. On June 1942, the Japanese government sent a secret message revealing their plan of attack on Midway. A small number of Japanese soldiers were to attack nearby islands to distract the allied troops who were positioned at Midway. The US intercepted this message and learned about this secret attack by decrypting the message and warned the allied troops about this. In order to avoid suspicion that Purple was broken, the troops pretended to move away from Midway but as the Japanese soldiers came to attack, the allied troops turned around and initiated a surprise assault [12]. The US was able to stop Japan from taking over Midway island. The US used the advantage of being able to read encrypted messages to reveal their plans of attack on Pearl Harbor [1].

2.3 Purple Cipher

2.3.1 The Purple Machine

The Purple machine is made of three major components. The first part is an electric typewriter, which is used to input information to the machine.

The second part is a ‘‘cryptographic assembly”, which consists of plug boards, four electric coding rings, and numerous wires and switches, which map the plaintext letter to various letters to produce the final cipher text character. The Purple machine consists of four switches: a sixes switch and three twenties switches. The sixes switch is used to permute 6 letters from the English alphabet and the twenties switches are used to permute the remaining 20 letters of the English alphabet, as shown in Figure 2.2. Each switch has 25 hardwired unrelated permutations. The sixes switch steps for each character,

whereas the twenties switches step depending on the position of the other switch positions (fastSwitch, middleSwitch, slowSwitch).

The last part is an output unit that prints the encrypted message from the machine [8] . In order to decrypt the cipher text, the Purple machine needs to be set up with the exact same configuration as it is used to encrypt the plaintext. Then, the cipher text needs to be input through input typewriter and plaintext would be received from output typewriter [7] .

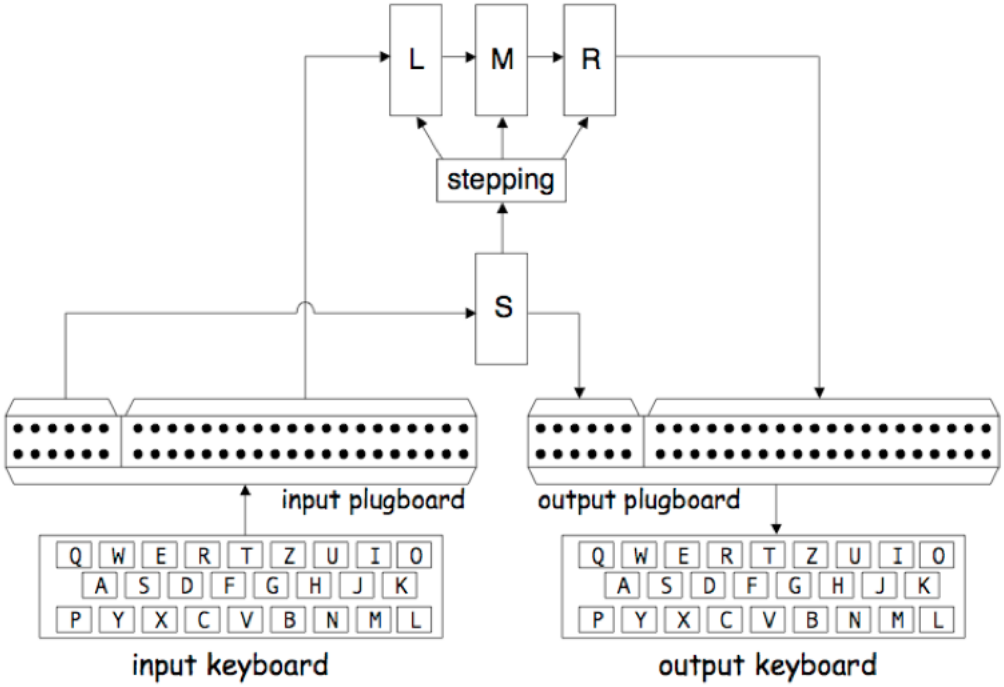

Figure 2.2: Purple machine (S-sixes switch, L-M-R - twenties switch) [2]

2.3.2 Components of the Purple Machine

The cryptographic elements of the Purple machine are an input plugboard, switches to permute the text, and the output plugboard [8]. Theoretically, the two plugboards can be connected independently of each other, but in reality the Japanese government kept the two plugboard settings identical [8].

The two plugboards are divided into two sets of characters: the sixes, which contains 6 letters from the English alphabet, and the twenties, which contain the remaining 20 letters

of the English alphabet. The two internal plugboards (sixes and twenties) are connected to external plugboards (sixes and twenties) that are used by typists. The external sixes portion is matched with internal plugboard's sixes and similarly external twenties portion is matched with internal plugboard's twenties. The external alphabets can be a permutation of the English alphabet agreed by both parties, in advance. Following is an example connection of external alphabets to the internal plugboard setting.

	Sixes	Twenties
Internal plugboard alphabet:	AEIOUY	BCDFGHJKLMNPQRSTVWXZ
External alphabet:	NOKTYU	XEQLHBRMPDICJASVWGZF

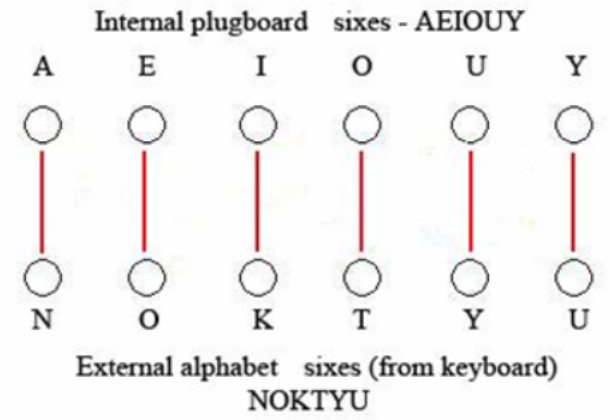

Figure 2.3: Mapping sixes from external keyboard to internal plugboard [3]

The sixes portion of the plugboard is connected to only one stepping switch. This switch decides the characters from the external plugboard that will be mapped to the characters from the internal plugboard. The six characters on the internal plugboard do not vary and are fixed. The switch can be used to have 25 mappings between the characters out of 6!, that is 720 different possibilities [3]. Each letter uses a different mapping because of the switch that steps for each character of the plaintext. Thus, 25 different mappings can be used for encryption of sixes portion.

The twenties portion of the plugboard is connected to three different switches. These three switches decide the twenty characters from the external plugboard that will be mapped to the twenty characters from the internal plugboard. Each of the three switches can be used for 25 different permutations of characters out of 20! total possibilities [3].

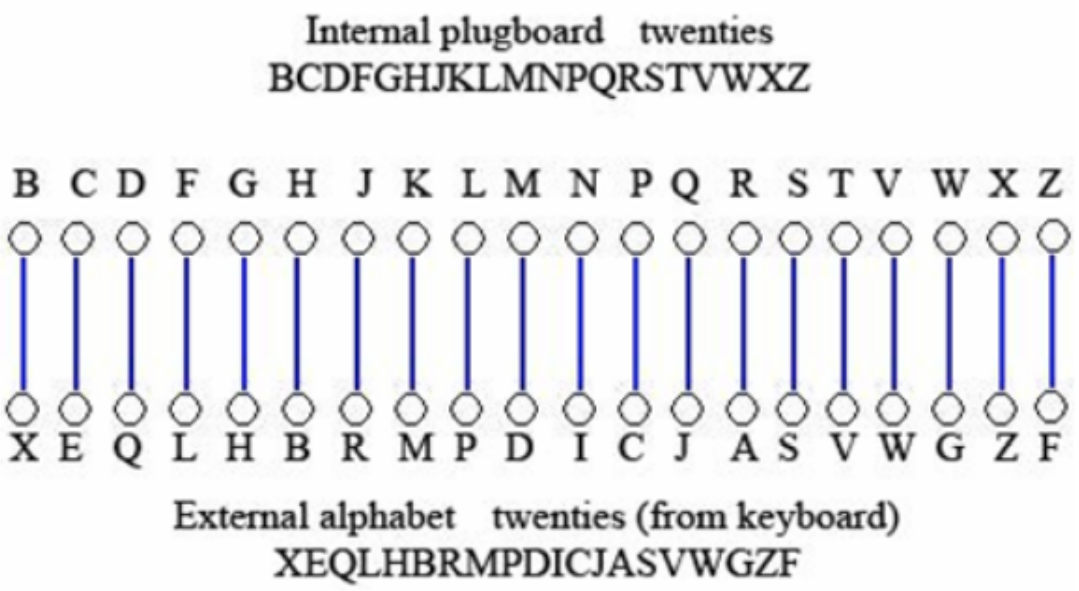

Figure 2.4: Mapping twenties from external keyboard to internal plugboard [3]

Therefore, the period length of this mapping sequence is 25x25x25 = 15,625. Along with these three switches, a concept of movement rules is also incorporated to make it more complex and secure. The motion rules such as ‘‘fast”, ‘‘middle”, and ‘‘slow” are assigned to each of the three switches. This adds an additional complexity of 6 possible switch motions.

The internal configuration and components are hardwired and cannot be changed.

In order to understand the internal workings of Purple’s encryption and decryption, one needs to understand the stepping of the switches. As mentioned earlier, there are two types of switches, one sixes switch and three twenties switches. For all the switches, when they reach their 25th position, they will rotate back to the 1st position on their next advancement and the routine goes on. The sixes switch steps for each character, therefore it repeats itself after encipherment of 25 characters.

However, there are certain rules or conditions that govern the stepping of the twenties switches. One of the twenties switches is labeled as ‘‘fast”, another one as ‘‘middle”, and the last one as ‘‘slow”. One of these switches steps for each character. Normally, the ‘‘fast” steps to its next position with two exceptions, which occur when the sixes switch reaches the 24th or 25th position. When the sixes switch moves to its 24th position (mapping), if

the twenties ‘‘middle” switch is positioned at its 25th mapping, the ‘‘slow” switch steps to next mapping [3]. Also, when the sixes switch is in its 25th position, the ‘‘middle” switch will advance. Here is the sample of how the internal stepping switches work:

Switches Position			
Sixes	TwentiesOne	TwentiesTwo	TwentiesThree
21	1	25	5
22	2	25	5
23	3	25	5
24	4	25	5
25	4	25	6
1	4	1	6
2	5	1	6

Table 2.1: Sample switch position One:fast, Two:middle, Three:slow [3]

The encryption process has 3 basic steps. First, the character entered or input, C1 is seen to identify if it belongs to sixes or twenties. Then C1 is permuted to another character C2 based on the input plugboard and external alphabet. If it belongs to the sixes, then the character C2 is encrypted to another character C3 by the sixes switch. The sixes switch advances to the next position for each character entered.

If the character entered belongs to the twenties portion, then it is encrypted thrice, once each by twentiesOne, twentiesTwo, and TwentiesThree switches sequentially (the decipher process is reversed; that is the letter will be permuted in the order: 3rd, 2nd, and 1st switch). Character C3 will be obtained after this step. Third, C3 is now taken to the external plugboard and permuted with same settings as the internal plugboard. The final character is displayed in the output device [3].

2.3.3 An example illustrating Purple’s Encipherment

Consider the plain text ‘‘IT TAKES COURAGE TO ADMIT FEAR’’ with the following configuration:

The permutations are cited in the Appendix A.

Initial sixes switch position: 3

Initial twenties switches position: 5, 8, 6 (1st, 2nd, and 3rd, respectively).

Twenties switches motion: 1, 2, 3 (fast, middle, and slow, respectively)

	Sixes	Twenties
Alphabet:	`NAGVPQ`	`OEFKUHCRYWSMIZJDXBLT`
Plugboard:	`AEIOUY`	`BCDFGHJKLMNPQRSTVWXZ`

Table 2.2 shows the encryption of plaintext using the mentioned switch positions for sixes and twenties. The cipher text is: "MYFNYOTZBZSAVCMIADUBXOYNX"

Plaintext letter	Input Plugboard	Sixes Switch	TwentiesOne Switch	TwentiesTwo Switch	TwentiesThree Switch	Output Plugboard
I	Q	4	6	8	6	M
T	Z	5	7	8	6	Y
T	Z	6	8	8	6	F
A	E	7	9	8	6	N
K	F	8	10	8	6	Y
E	C	9	11	8	6	O
S	N	10	12	8	6	T
C	J	11	13	8	6	Z
O	B	12	14	8	6	B
U	G	13	15	8	6	Z
R	K	14	16	8	6	S
A	E	15	17	8	6	A
G	I	16	18	8	6	V
E	C	17	19	8	6	C
T	Z	18	20	8	6	M
O	B	19	21	8	6	I
A	E	20	22	8	6	A
D	T	21	23	8	6	D
M	P	22	24	8	6	U
I	Q	23	25	8	6	B
T	Z	24	1	8	6	X
F	D	25	1	8	6	O
E	C	1	1	9	6	Y
A	E	2	2	9	6	N
R	K	3	3	9	6	X

Table 2.2: Example of Purple encipherment

2.3.4 Weakness of the Purple Cipher

The weakness of the Purple machine is the partition of the plugboard into sixes and twenties. This partition limits the number of keys or configurations possible. The sixes switch can have 25 keys or different permutations. As there are three twenties switches each with 25 different permutations, there can be a total of 25^3 possible keys. In addition, there are 6 possible switch motions assigned to the twenties switches. Therefore, Purple only has 6 x 25 (sixes) x 25^3 (twenties) = 2,343,750 possible configurations of switches [3]. Regardless of the 6 different switch motions assigned to twenties, the Purple machine will repeat after encrypting 25^4 = 390,625 characters. Due to the partitioning of plugboard, the whole 26 letters of English alphabet (which provide a total of 26! permutations) cannot be used. Instead, the alphabet is divided into 2 portions, the sixes and twenties, which produce a much smaller number of 6! x 20! keys [3]. This can be used as an advantage by the cryptanalysts by determining the sixes portion of the alphabet first. After the sixes is determined successfully, cryptanalysts can focus on the twenties portion. The time required to decipher the cipher text is reduced because of the separation of plugboard into sixes and twenties. The cryptanalysts need not consider all the possible keys simultaneously when attacking the Purple machine. That means there are only 20! possible keys and 6 x 25^3 = 93,750 possible configurations of the twenties, which is an improvement compared to 26! keys and 6 x 25^4 = 2,343,750 configurations [3].

2.3.5 Algorithm

2.3.5.1 Encryption

In Figure 2.5 and 2.6, the algorithm for Purple encryption is given. The input to the Purple machine are plaintext, switches, and plugboard. The input switches should be of the format (1-1,1,1-12) where the first 1 represents starting position of sixes switch and 1,1,1 represent starting position of twentiesOne, twentiesTwo and twentiesThree switches respectively. The last two digits represent the fast and middle switches. In line 1, the sixes switch is built and on line 2, the start position for the sixes switch is set that is

specified as part of input. In line 3, the twentiesOne switch (encrypt and decrypt) with the starting position and the matrix setting provided is built. In line 4, start position for the twentiesOne switch is set. Similarly, in lines 5, 6, 7, and 8 the other two twenties switches are built and start positions are specified. In lines 9 and 10, the switch motion rules (fast, middle, slow) are assigned to the twenties switches. In line 13, "purpleEncrypt" method is called to perform the encryption of the plaintext.

```
1. sixes=buildSwitch(Switches.Sixes, startSwitchPosSixes)
2. sixes.setPosition(startSwitchPosSixes)
3. twentiesOne=buildSwitch(Switches.TwentiesOne, startSwitchPosTwentiesOne)
4. twentiesOne.setPosition(startSwitchPosTwentiesOne)
5. twentiesTwo=buildSwitch(Switches.TwentiesTwo, startSwitchPosTwentiesTwo)
6. twentiesTwo.setPosition(startSwitchPosTwentiesTwo)
7. twentiesThree=buildSwitch(Switches.TwentiesThree, startSwitchPosTwentiesThree)
8. twentiesThree.setPosition(startSwitchPosTwentiesThree)
9. fastSwitch=twenties.get(fastValue-1)
10. middleSwitch=twenties.get(middleValue-1)
11. alphabet=plugboard
12. for i = 0 to alphabet.length() - 1
13.       plugboardMap.put(alphabet.charAt(i), i)
14. next i
15. CALL purpleEncrypt with plaintext
```

Figure 2.5: Encryption algorithm for Purple

The "purpleEncrypt" method algorithm is as shown in Figure 2.6. In line 3, the input plaintext character is mapped to the external plugboard character. In line 4, the character is checked to see if it falls under the sixes or twenties alphabet. If its a sixes character, the plaintext is encrypted using the sixes encryption permutation else it is encrypted using the twenties one, two, and three switches respectively. The encrypted character is appended to the cipher text.

2.3.5.2 Decryption

Figure 2.7 shows the method used to decrypt the cipher text. Decryption will be exactly the same as encryption except for a slight change. In line 8, if the plaintext is a twenties character then it is decrypted in the reverse order of switches, that is, it is decrypted using twenties three, two and one switches respectively.

```
1. begin purpleEncrypt(plaintext)
2.       for i = 0 to plaintext.length() - 1
3.            n = plugboardMap.get(plaintext.charAt(i))
4.            if n <6 then
5.                 x = sixesSwitch[switchPosition][n]
6.            else
7.                 n = n - 6
8.                 x = twentiesSwitch[switchPosition][n]
9.                 x = x + 6
10.            end if
11.            ciphertext.append(x)
12.       next i
13. end purpleEncrypt
```

Figure 2.6: Algorithm for ‘‘purpleEncrypt” method

```
1.  begin purpleDecrypt(plaintext)
2.       for i = 0 to ciphertext.length() - 1
3.            n = plugboardMap.get(ciphertext.charAt(i))
4.            if n <6 then
5.                 x = sixesSwitch[switchPosition][n]
6.            else
7.                 n = n - 6
8.                 x = twentiesSwitch[switchPosition][n]
9.                 x = x + 6
10.            end if
11.            decryptedplaintext.append(x)
12.       next i
13. end purpleDecrypt
```

Figure 2.7: Algorithm for ‘‘purpleDecrypt” method

2.4 PseudoPurple Cipher

2.4.1 Components of the PseudoPurple Machine

The PseudoPurple machine is a variation of the actual Purple machine explained in the previous section. It is a simple machine from the configuration and a complexity point of view, but a more powerful machine in terms of the effort required to break the cipher.

As with the Purple machine, the PseudoPurple machine consists of almost the same elements i.e input plugboard, switches to permute the text, and the output plugboard. Unlike the plugboards in Purple, which are divided into sixes and twenties parts, the plugboards in the PseudoPurple machine are not partitioned.

There is no concept of partitioning the alphabet into sixes and twenties, which makes PseudoPurple more powerful and less complex than Purple. As there is no partition of alphabets into sixes and twenties, any permutation of the English alphabet can be used. This permutation is agreed by both the parties in advance. Also, as there are no twenties switches, there is no necessity of movement rules for different switch positions such as slow, fast, and middle as illustrated in the Figure 2.8. The PseudoPurple machine essentially steps for each plaintext letter.

The encryption process is very simple. If the plaintext character is C1, which is entered from the typewriter, it is changed to another character C2 based on the chosen plugboard permutation, and then to C3 based on the encrypt switch configuration. The machine steps for each character from the initial switch position without any switch motion rules.

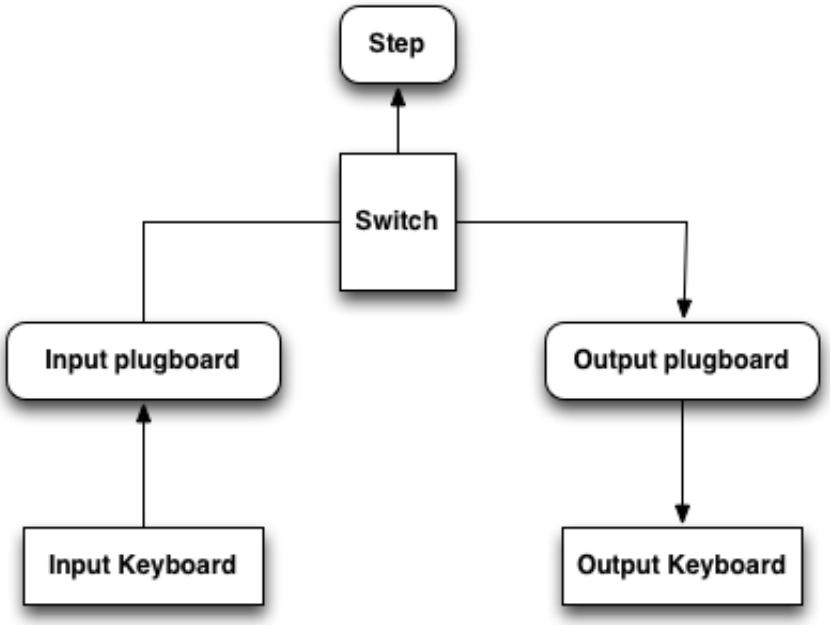

Figure 2.8: PseudoPurple Machine

2.4.2 An example illustrating PseudoPurple's Encipherment

Consider the plain text ‘‘IT TAKES COURAGE TO ADMIT FEAR’’ with the following configuration:

The initial switch position is 4 and the external alphabet is:

NAGVPQOEFKUHCRYWSMIZJDXBLT

Table 2.3 shows the encryption of plaintext using the mentioned switch positions for sixes and twenties.

The cipher text is: ‘‘HJJRCQVWSUXPZTXGAONQTFWXM”

Plaintext letter	Input Plugboard	Encrypt Switch	Output Plugboard
I	Q	5	H
T	Z	6	J
T	Z	7	J
A	E	8	R
K	F	9	C
E	C	10	Q
S	N	11	V
C	J	12	W
O	B	13	S
U	G	14	U
R	K	15	X
A	E	16	P
G	I	17	Z
E	C	18	T
T	Z	19	X
O	B	20	G
A	E	21	A
D	T	22	O
M	P	23	N
I	Q	24	Q
T	Z	25	T
F	D	1	F
E	C	2	W
A	E	3	X
R	K	4	M

Table 2.3: Example of PseudoPurple encipherment

2.4.3 Strength of PseudoPurple Machine

As discussed in Section 2.3.4, the Purple machine has a weakness due to the splitting of the internal input and output plugboards into sixes and twenties. Since the alphabet is divided into 2 portions, the total number of possible keys is: 6! x 20!. However, in the case of PseudoPurple the total number of possible keys is 26!, since the alphabets is not divided into two portions.

This means that cryptanalysts require more time and effort to decipher the cipher text in PseudoPurple when compared to Purple. In the case of Purple, the cryptanalysts can determine the sixes portion first, which will require them concentrating on 6! keys. After the sixes setting is determined, the cryptanalysts can concentrate on the twenties switches, which means 20! possibilities. But in PseudoPurple, the cryptanalysts should consider all the possible number of keys without splitting the alphabets into sixes and twenties i.e 26!.

2.4.4 Algorithm of PseudoPurple

2.4.4.1 Encryption Algorithm

In Figure 2.9 and 2.10, the algorithm for PseudoPurple encryption is given. The input to the PseudoPurple machine are plaintext, switch start position, and plugboard. In lines 1 and 2, the encrypt switch is built and start position is assigned. In line 7, the ‘‘purpleEncrypt” method is called to encrypt the plaintext.

```
1. switch = buildSwitch(startSwitchPosSixes)
2. switch.setPosition(startSwitchPosSixes)
3. alphabet = plugboard
4. for i = 0 to alphabet.length() - 1
5.      plugboardMap.put(alphabet.charAt(i), i)
6. next i
7. CALL purpleEncrypt with plaintext
```

Figure 2.9: PseudoPurple encryption algorithm

The ‘‘purpleEncrypt” method algorithm is as shown in Figure 2.10. In line 3, the input plaintext character is mapped to the external plugboard character. In lines 4 and 5, the plaintext is encrypted using the encrypt switch permutation and appended to the cipher

text.

```
1. begin purpleEncrypt(plaintext)
2.       for i = 0 to plaintext.length() - 1
3.            n = plugboardMap.get(plaintext.charAt(i))
4.            x = switch[switchPosition][n]
5.            ciphertext.append(x)
6.       next i
7. end purpleEncrypt
```

Figure 2.10: PseudoPurple encryption algorithm

2.4.4.2 Decryption Algorithm

Figure 2.11 shows the method used to decrypt the cipher text. Decryption will be exactly the same as encryption except for a slight change. In line 5, the cipher text is decrypted using the decrypt switch permutation and appended to the plaintext.

```
1. begin purpleDecrypt(cipher text)
2.       for i = 0 to ciphertext.length() - 1
3.            n = plugboardMap.get(ciphertext.charAt(i))
4.            x = switch[switchPosition][n]
5.            decryptedplaintext.append(x)
6.       next i
7. end purpleDecrypt
```

Figure 2.11: PseudoPurple decryption algorithm

2.5 Genetic Algorithms

2.5.1 History of Evolutionary Computation

In around 1950s, many computer researchers tried to understand evolutionary systems and believed that the evolution techniques can be used to solve many engineering problems. The main goal in these evolving systems was to generate a set of optimal or candidate solutions to a problem by making use of the genetic operators motivated by variation and selection [13].

In 1960s, Rechenberg introduced ‘‘evolution strategies”, a technique used to optimize the solution to given problems using adaption and evolution strategies. Fogel, Owens, and Walsh [13] then introduced a new technique called ‘‘evolutionary programming”. In this, candidate solutions were represented as finite state machine by modifying their state-transition diagrams after which the most optimal solutions were selected. Genetic algorithms were developed by John Holland [13] and his students in 1960. Holland's goal was to study the phenomenon of evolution as it occurs in the nature without thinking from the engineering perspective and then develop ways to import the same in computer systems.

The foundation of evolutionary computation is based on the three methods mentioned previously [13]. These evolutionary techniques can be applied to engineering problems, where we need to find an optimal solution among the huge number of possible solutions.

2.5.2 Introduction to Genetic Algorithms

Genetic algorithms (GA) are meta heuristics that use the principles of biology to search through the candidates and find a solution to the optimization and search problems [8]. Genetic algorithms are motivated by techniques of natural evolution such as inheriting traits from parents, recombination, and mutation to have individual unique traits. GAs encode the decision variables or the solutions to the problems as a string of binary values. These strings, which are the solutions to the problems, are referred to as chromosomes [5]. To evolve to good solutions, there should be a mechanism to separate the probable good solutions from the bad ones that is called fitness function. Another important terminology of GAs is the population size, i.e. the number of chromosomes (possible solutions). The population size, which is specified as part of input, is one of the deciding factors of scalability and performance of genetic algorithms [4]. A larger population of candidate solutions may give better solutions than a smaller population of candidate solutions.

Once we determine the two main factors, the representation of the search variable or chromosome and the measure of separating the good solutions from the bad ones, we can follow the steps mentioned below to evolve the search variables until we find a better and

optimal solution.

1. **Initialization**. The set of variable solutions are randomly selected to create a population of chromosomes or a set of possible solutions to the given problem.

2. **Evaluation**. After the initialization of a set of random possible solutions, the mathematical formula to evaluate the fitness of each of the solutions is applied. This helps us to determine the good solutions to the problem and identify the bad ones.

3. **Selection**. The main goal of this step is to select the optimal solution with higher probability given to solutions that have a better fitness course. This imposes the survival of the fittest ‘‘Darwin” mechanism on the solutions. More copies of solutions with better fitness scores are made than the solutions with low fitness scores. Many selection techniques such as fitness proportionate selection, ranking selection, and tournament selection are present to make sure that good solutions are separated from the bad ones [4].

4. **Recombination** In this step, an offspring is created from two or parental chromosomes or solutions. The offspring created does not match either of the parents completely, but has some characteristics of each of the parents. There are many ways of performing recombination such as one-point crossover, two-point crossover etc.

5. **Mutation** After the new child solution is created in the recombination step, the new solution is modified randomly by switching the bits to create a new mutated child. The purpose of this step is to provide the child solution with its individual traits.

6. **New Population** After the new solution is created using evaluation, selection, recombination, and mutation steps, it is placed in the new population.

7. **Termination** Repeat steps 2-6 until a termination point is reached, determined by the number of restarts provided or the best solution found.

In the next section, we will explain different techniques involved in selection, recombination, and mutation steps.

2.5.3 Operators in Genetic Algorithms

2.5.3.1 Selection Methods

Selection techniques can be broadly classified into two types.

1. **Fitness Proportionate Selection** This includes methods such as roulette-wheel selection (this selection method is employed for cryptanalysis of Purple) and stochastic universal selection [14]. In roulette-wheel selection, the fitness score of each solution is computed. Solution with better fitness score is assigned more slots when compared to solution with a low fitness score. That is, the number of slots allocated to each of the solutions are directly proportional to the scores. Then solutions are randomly selected by spinning the wheel.

   The following steps are employed in the roulette-wheel selection method:

   (a) Evaluate the fitness, f_i, of each individual in the population.

   (b) Calculate the total number of slots to be allocated. Say the population size is n, then s=10 x n, where s is the number of slots.

   (c) Compute the probability (slot size), $p_i = f_i \,/\, \sum_{j=1}^{n} f_j$, where n is the population size.

   (d) Allocate each solution, p_i x s number of slots.

   (e) Generate a random number $r \in (0,s]$.

   (f) Select the solution which is in r_{th} slot.

2. **Ordinal Selection** This involves techniques such as tournament and truncation selections. In tournament selection, s chromosomes are chosen randomly and a tournament is started between these chromosomes. The fittest or most optimal individual in the group of k chromosomes wins the tournament and is elected as the parent. Usually 2 chromosomes are chosen randomly to start the tournament.

This scheme requires at least n tournaments to select n parent chromosomes. In the truncation method, each mating pool requires that s copies of chromosomes be assigned to each of the top $(1/s)_{th}$ of the chromosomes. [4].

2.5.3.2 Recombination (Crossover) Methods

After the selection methods, two parent chromosomes are chosen [14]. Different recombination techniques listed below can be applied to generate new offspring.

1. **One point crossover** A uniform random number r is selected across the string length, which is called the crossover point. The bits from one side of the site are recombined as shown in Figure 2.12 to generate new offspring.

2. **Two point crossover** In two point crossover, two random crossover points are selected. The bits from two sides of the site are recombined as shown in Figure 2.12 to generate new offspring. There can be k-point crossover, where k random crossover points are selected.

3. **Uniform crossover** In uniform crossover as illustrated in Figure 2.12, all the bits are exchanged between the pair of chromosomes, which are selected at random with a certain swapping probability (usually this probability value is 0.5) [4].

2.5.3.3 Mutation methods

Mutation ensures that there is diversity in the offsprings and have their own individual traits. It prevents stagnation in the evolution [4]. There are two broad categories of mutation as mentioned below.

1. **Flip mutation** In this, a uniform random bit across the string length is selected and then flipped. That is, a '1' is changed to '0' and vice versa as shown in Figure 2.13

2. **Swap mutation** In this, two uniform random bits across the chromosome length are selected and then swapped as shown in Figure 2.14

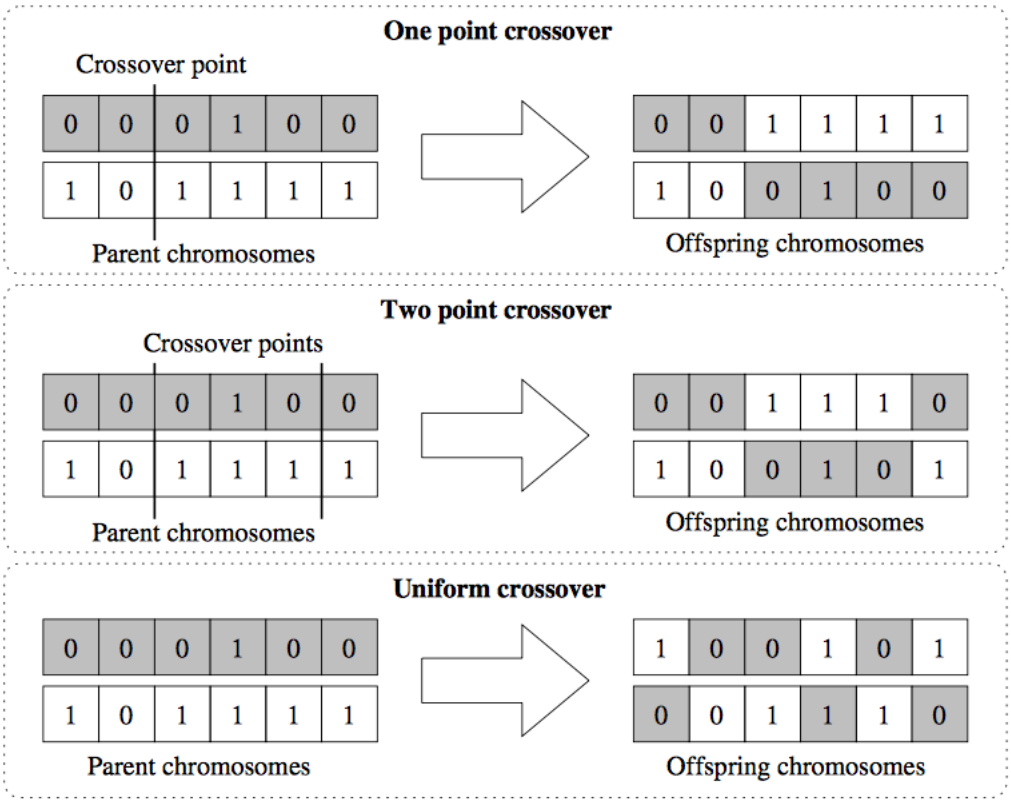

Figure 2.12: Different recombination techniques [4]

Before Mutation	1010110000110101
After Mutation	1011110000110101

Figure 2.13: Flip mutation [5]

Before Mutation	1010110000110101
After Mutation	1011110000110100

Figure 2.14: Swap mutation [5]

2.6 Hidden Markov Models

A stochastic process consists of a number of events where the outcome of each event depends on a certain probability [15].

A Markov process is a stochastic process that consists of the following three properties [15]

- The number of possible states is finite and fixed.

- The outcome of an event at each stage depends on the outcome of an event from the previous stage.
- The probabilities remain constant over time.

A hidden Markov model (HMM) is a Markov process in which the system being modeled is assumed to have hidden states [16]. In Markov models, the states are directly visible to the user and therefore the state transition probabilities are the only parameters. But, in HMMs the states are not directly visible, however the output, which is dependent on the states is visible. Each state has a possible distribution over the possible output tokens. Therefore the output token gives some information of the hidden states.

2.6.1 Notation

The various components and notations associated with hidden Markov models (HMM) are [6]

T = length of observation sequence

N = number of states in the model

M = number of observation symbols

$Q = \{q_0, q_1, ..., q_{N-1}\}$ = distinct states of the Markov process

$V = \{0, 1, ..., M\text{-}1\}$ = set of possible observations

A = state transition probabilities

B = observation probability matrix

π = initial state distribution

$\mathcal{O} = (\mathcal{O}_0, \mathcal{O}_1, ..., \mathcal{O}_{T-1})$ observation sequence

A generic hidden Markov model is as shown in Figure 2.15, where X_i is the hidden states and the rest of the notations are listed above. The Markov process is determined by the current state and the A matrix. We are only able to observe $\mathcal{O}$, which is related to the hidden states of the Markov process by the matrix B .

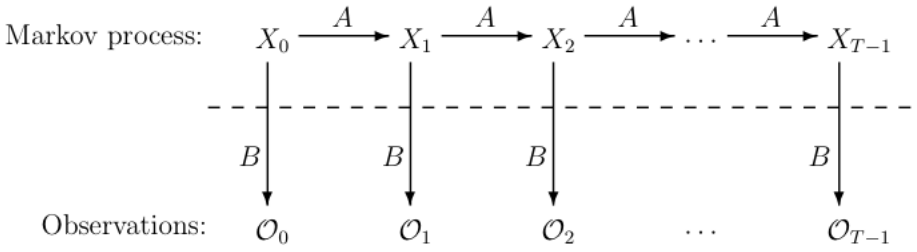

Figure 2.15: Hidden Markov model [6]

2.6.2 Three problems of HMM

HMM has three types of problems associated with it. We will discuss the three problems briefly.

2.6.2.1 Problem 1

This evaluates the model. Given the model $\lambda = (A, B, \pi)$ and a sequence of observations $\mathcal{O}$, find P $(\mathcal{O} \mid \lambda)$. Here, we want to determine the probability of the output sequence, given the model.

2.6.2.2 Problem 2

This deals with decoding of the model. Given the model $\lambda = (A, B, \pi)$ and a sequence of observations $\mathcal{O}$, we need to find the state sequence that produced the observation sequence.

2.6.2.3 Problem 3

Given the parameters N and M, the observation sequence $\mathcal{O}$, we need to find the model $\lambda = (A, B, \pi)$, which has the highest probability of generating the given observation sequence.

2.6.3 Baum-Welch Algorithm

The Baum Welch algorithm is used to find the hidden parameters of the hidden Markov model. It makes use of the forward-backward algorithm. We will discuss the

forward-backward algorithms in detail in the next sections.

2.6.3.1 Forward Algorithm

The forward algorithm in context of HMM is used to calculate the probability of state at given time, given the history of evidence.

The Forward Algorithm analyzes the probabilities of transitioning from one state to the next based on the given observations [17]. The solution to the Forward Algorithm is to iteratively compute the state transition probabilities for each observation [6].

To find the solution to problem 1, we find $P\ (\mathcal{O} \mid \lambda)$ with the help of forward algorithm or α - pass. For $t = 0, 1,..., T$ - 1 and $i = 0, 1,..., N$ -1, define

$$\alpha_t(i) = P\ (\mathcal{O}_0, \mathcal{O}_1, ..., \mathcal{O}_t, x_t = q_i \mid \lambda)$$

$\alpha_t(i)$ is the probability of the partial observation sequence upto time t, where the Markov process is in current state q_i at time t.

$\alpha_t(i)$ can be computed recursively as follows [6]

1. Let $\alpha_0(i) = \pi_i b_i(\ \mathcal{O}_0\ )$, for $i = 0, 1,...,N$ - 1.

2. For $t = 1,2,...,T$ - 1 and $i = 0,1,...,N$ - 1, compute

$$\alpha_t(i) = \left[\ \sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji}\ \right] b_i(\mathcal{O}_t)$$

3. Then finally,

$$P\ (\mathcal{O} \mid \lambda) = \sum_{i=0}^{N-1}\ \alpha_{T-1}(i)$$

2.6.3.2 Backward Algorithm

To find the solution to problem 2, we use the Backward algorithm, or β - pass. This is almost same as forward algorithm, but we start calculating the probability from the end and work our way up to the start. That is, the β - pass starts from the last observation symbol and ends with the first observation symbol.

For $t = 0, 1,..., T$ - 1 and $i = 0, 1,..., N$ - 1, we define [6]

$$\beta_t(i) = P\ (\mathcal{O}_{t+1},\ \mathcal{O}_{t+2},\ldots,\ \mathcal{O}_{T-1},\ x_t\ = q_i\ |\ \lambda)$$

$\beta_t(i)$ can be computed recursively as follows [6]

1. Let $\beta_{T-1}(i) = 1$, for i = 0,1,...,N - 1.

2. For $t = T$ - 2,T - 3,...,0 and i = 0,1,...,N - 1 compute

$$\beta_t(i) = \sum_{j=0}^{N-1} a_{ij} b_j\ (\mathcal{O}_{t+1})\ \beta_{t+1}(j)$$

2.6.3.3 Computing Gamma and di-Gammas

For t = 0, 1,..., T - 1 and i = 0, 1,..., N - 1, we define [6]

$$\gamma_t(i) =\ \ P\ (\ x_t = q_i\ |\ \mathcal{O},\ \lambda)$$

From the above formula, $\gamma_t(i)$ can be defined as most likely state at time t is the state q_i for which $\gamma_t(i)$ is maximum.

$\gamma_t(i)$ can be calculated from α and β as follows

$$\gamma_t(i)= \alpha_t(i)\ \beta_t(i)\ /\ P\ (\mathcal{O}\ |\ \lambda)$$

Then $\gamma_t(i, j)$ shows the probability of transition from state q_i at time t to another state q_j at time $t + 1$.

$\gamma_t(i, j)$ can be defined in terms of α, β, A and B as follows

$$\gamma_t(i,j)= \alpha_t(i)a_{ij}b_j(\mathcal{O}_{t+1})\beta_{t+1}(j)\ /\ P\ (\mathcal{O}\ |\ \lambda)$$

2.6.3.4 Scaling and Re-estimation

The above calculations require multiplying the probabilities. The values tend to 0 as T increases, which may lead to underflow. The solution to this problem is to scale the values, but care should be taken to verify the validity of the re-estimation formula. Re-estimation is an iterative process. After all the values are computed, the final step is to re-estimate values for π, A, B. The next iteration starts again with new re-estimated values to calculate α, β, γ and Di-gamma values. The process repeats until the probability is better than the previous iteration. The subsequent section describes the entire process of the Baum-Welch Algorithm in detail with the help of pseudo code.

2.6.4 HMM algorithm

The values N and M are fixed and observation sequence is assumed to be known [6].

1. **Initialization**

   The initial values for matrix A,B, and π are computed. π is 1 x N, $A = \{a_{ij}\}$ is N x N, $B = \{b_j(k)\}$ is N x M. The three matrices are row-stochastic [6].

   Let
   maxIters = maximum number of re-estimation iterations
   iters = 0
   oldLogProb = - ∞.

2. **The α pass**
   //compute $\alpha_0(i)$
   $c_0 = 0$
   for $i = 0$ to N - 1
   $\alpha_0(i) = \pi_i b_i(\ \mathcal{O}_0\ )$
   $c_0 = c_0 + \alpha_0(i)$
   next i

   // scale the $\alpha_0(i)$ $c_0 = 1/\ c_0$
   for $i = 0$ to N - 1
   $\alpha_0(i) = c_0 \alpha_0(i)$
   next i

   // compute $\alpha_t(i)$
   for $t = 0$ to T - 1
   $c_t = 0$
   for $i = 0$ to N - 1
   $\alpha_t(i) = 0$
   for $j = 0$ to N - 1
   $\alpha_t(i) = \alpha_t(i) + \alpha_{t-1}(j) a_{ji}$
   next j
   $\alpha_t(i) = \alpha_t(i) b_i(\mathcal{O}_t)$
   $c_t = c_t + \alpha_t(i)$
   next i
   // scale $\alpha_t(i)$
   $c_t = 1\ /\ c_t$
   for $i = 0$ to N - 1
   $\alpha_t(i) = c_t \alpha_t(i)$
   next i
   next t

3. **The β pass**
   //Let $\beta_{T-1}(i) = 1$ scaled by c_{T-1}

for $i = 0$ to N - 1
$\beta_{T-1}(i) = c_{T-1}$
next i

// β pass
for $t = T$ - 2 to 0 by -1
for $i = 0$ to N - 1
$\beta_t(i) = 0$
for $j = 0$ to N - 1
$\beta_t(i) = \beta_t(i) + a_{ij} bj(\mathcal{O}_{t+1})\beta_{t+1}(j)$
next j
// scale $\beta_t(i)$ with same scale factor as $\alpha_t(i)$
$\beta_t(i) = c_t\beta_t(i)$
next i
next t

4. **Compute $\gamma_t(i, j)$ and $\gamma_t(i)$**
for $t = 0$ to T - 2
denom = 0
for $i = 0$ to N - 1
for $j = 0$ to N - 1
denom = denom + $\alpha_t(i)a_{ij}b_j(\mathcal{O}_{t+1})\beta_{t+1}(j)$
next j
next i
for $i = 0$ to N - 1
$\gamma_t(i) = 0$
for $j = 0$ to N - 1
$\gamma_t(i, j) =(\alpha_t(i)a_{ij}b_j(\mathcal{O}_{t+1})\beta_{t+1}(j))$ /denom
$\gamma_t(i)= \gamma_t(i) + \gamma_t(i, j)$
next j
next i
next t

5. **Re-estimate B and π**

// re-estimate π
for $i = 0$ to N - 1
$\pi_i = \gamma_0(i)$ next i

// re-estimate B
for $i = 0$ to N - 1
for $j = 0$ to M - 1
numer = 0
denom = 0
for $t = 0$ to T - 2
if ($\mathcal{O}_t == j$) then
numer = numer + $\gamma_t(i)$

end if
denom = denom + $\gamma_t(i)$
next t
$b_i(j)$ = numer / denom
next j
next i

6. **Compute log[P ($\mathcal{O}$ | λ)]**
logProb = 0
for i = 0 to T - 1
logProb = logProb + log(c_i)
next i
logProb = - logProb

7. **Should we iterate again ...?**
iters = iters + 1
if (iters < maxIters and logProb > oldLogProb) then
oldLogProb = logProb
goto 2
else
output λ= (A,B, π)
end if

Chapter 3

Genetic Algorithms Approach

3.1 Purple with Fixed Plugboard

The Purple cipher consists of two parts that form a key. One is the permutation of the alphabets to form the plugboard, which remains constant throughout the execution of Purple algorithm. That is, it does not step for each character in the plaintext. Second is the switch settings that are variable and step for each of the characters in the plaintext with some governing rules. The plugboard is connected to the internal plugboard. The sixes portion of the plugboard (external plugboard) is mapped to the sixes portion of the internal plugboard and the twenties portion of the plugboard is mapped to the twenties portion of the internal plugboard. However, in reality the Japanese government kept the two plugboard settings identical [8]. Therefore, in this section we use genetic algorithms to decipher the text keeping the plugboard fixed, whereas the switch setting is unknown.

3.1.1 Overview of the Algorithm

The main steps used to decipher the text using genetic algorithms are listed below.

- The switch settings are generated randomly in the format x-x,x,x-xx. The number of such variable solutions generated depends on the population size parameter specified as an input to this algorithm.

- After the initial set of variable solutions are generated randomly, the fitness of each solution is computed. This is done by decrypting the cipher text with each of the switch settings generated to obtain the putative plaintext. Then the putative

plaintext and the actual plaintext are compared using the hamming distance. This is a known plaintext attack, which is where the attacker has access to both the plaintext and the cipher text. Thus, we obtain a fitness score that helps us to differentiate between the potential good solutions and identify the bad ones. In reality, we could use the English alphabet letter frequency count technique as a means to determine the ideal score of the cipher text.

- Then we use the roulette-wheel selection technique explained in Section 2.5.3.1 to select two parents (possible solutions) giving higher probability to solutions that have better fitness score.

- After selecting two switch settings from the population, we create a child switch setting. To avoid the child resembling either one of the parents completely, one-point crossover, explained in Section 2.5.3.2, is performed so that the child has traits of both the parent switch settings.

- The new child switch solution is mutated by flipping any one of the random bits of the switch solution. This is done so that the child switch has some traits of its own.

- The new child switch is placed in a new population, where the size of the new population is the same as the size of the old population.

- The above steps are repeated for the number of restarts specified by the input parameters.

- The switch with the best fitness score is selected as the key.

Figure 3.1 explains the above algorithm in terms of a flowchart.

In this experiment, we encrypt some sample text using the Purple encryption method with the switch setting "12-6,3,10-31" and a fixed plugboard. The plaintext is as follows.

```
ALICEWASBEGINNINGTOGETVERYTIREDOFSITTINGBYHERSISTERONTHEBANKANDOF
HAVINGNOTHINGTODOONCEORTWICESHEHADPEEPEDINTOTHEBOOKHERSISTERWASRE
ADINGBUTITHADNOPICTURESORCONVERSATIONSINITANDWHATISTHEUSEOFABOOKT
HOUGHTALICEWITHOUTPICTURESORCONVERSATIONSOSHEWASCONSIDERINGINHERO
```

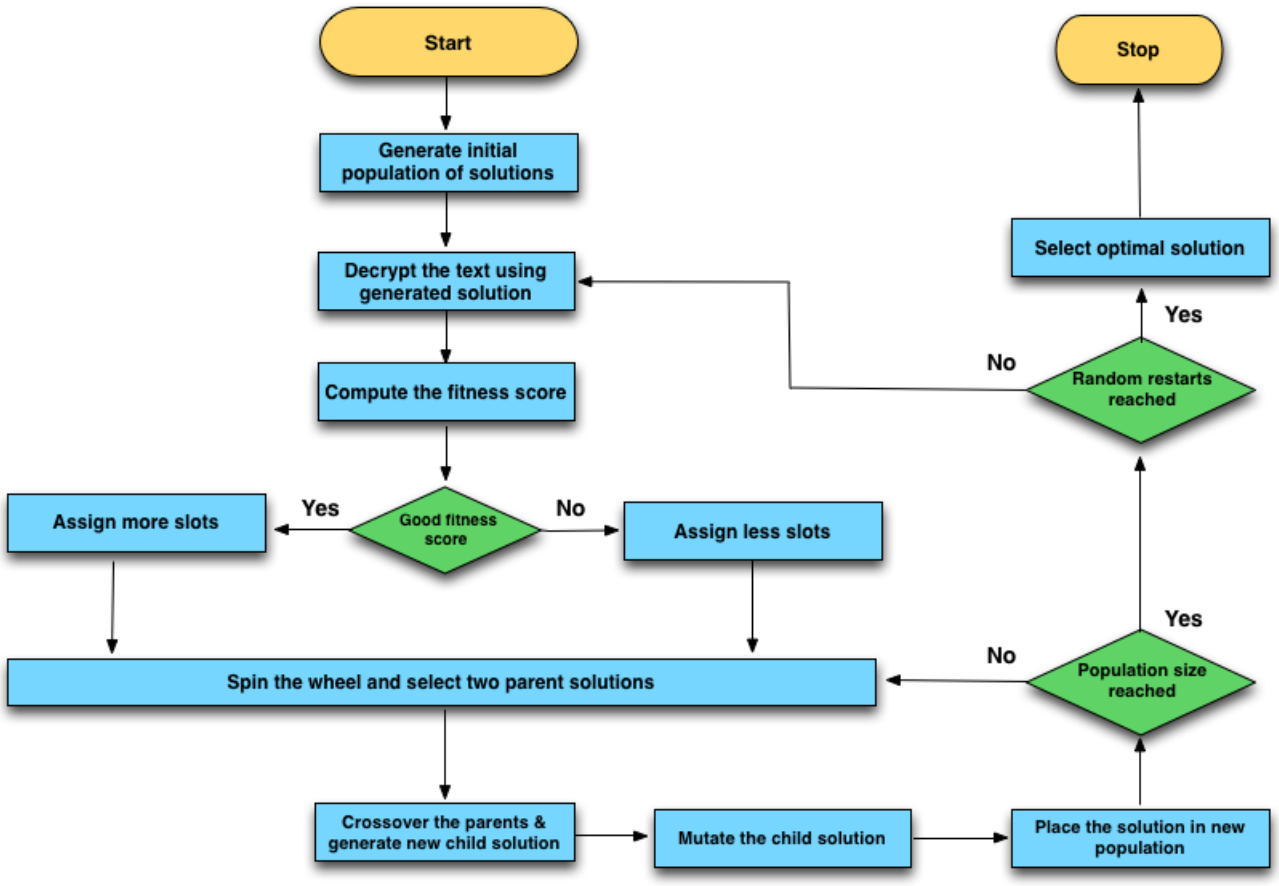

Figure 3.1: Flowchart for genetic algorithm

WNMINDASWELLASSHECOULDFORTHEHOTDAYMADEHERFEELVERYSLEEPYANDSTUPIDW
HETHERTHEPLEASUREOFMAKINGADAISYCHAINWOULDBEWORTHTHETROUBLEOFGETTI
NGUPANDPICKINGTHEDAISIESWHENSUDDENLYAWHITERABBITWITHPINKEYESRANCL
OSEBYHERTHEREWASNOTHINGSOVERYREMARKABLEINTHATNORDIDALICETHINKITSO
VERYMUCHOUTOFTHEWAYTOHEARTHERABBITSAYTOITSELFOHDEAROHDEARISHALLBE
LATEWHENSHETHOUGHTITOVERAFTERWARDSITOCCURREDTOHERTHATSHEOUGHTTOHA
VEWONDEREDATTHISBUTATTHETIMEITALLSEEMEDQUITENATURALBUTWHENTHERABB
ITACTUALLYTOOKAWATCHOUTOFITSWAISTCOATPOCKETANDLOOKEDATITANDTHENHU
RRIEDONALICESTARTEDTOHERFEETFORITFLASHEDACROSSHERMINDTHATSHEHADNE
VERBEFORESEENARABBITWITHEITHERAWAISTCOATPOCKETORAWATCHTOTAKEOUTOF
ITANDBURNINGWITHCURIOSITYSHERANACROSSTHEFIELDAFTERITANDFORTUNATEL
YWASJUSTINTIMETOSEEITPOP

We get the following cipher text from the plaintext after encrypting using the previously mentioned key.

YQURYFOVTOBUXRILBKOQOLZOLITUNYTIWSYHFYQJBOFYCTADXANUHJHEDAKRAXTIZSAPAT
RVETGOGLSEGIYZZUEZWMEQIRMOZYCLUICIGOGDOLRYRIUNZIWQYCXOPREJJYYKIGFDIKOK
CATPYKAHPOKIMYTQIRPOKZINIYCFAJYSYXHCHOWICNZUERYASERIARNGOAFSBESOBELIKW
OIMDIBBYWANOGNIPQENJYTIOXFESXAZABRYQKUWAXACBIXCEDUFSXONSARCANVYCGXIDYA
XVNOHRLOZUTJOEGUTIBAJNEIJJAWIPJYOMYUZFPHEDUKZFIMZYBZKUXZYIDOMUAVGUNUNR
YHOOHADZEYBDAAPWFIKYJMMFQUBPYATKEOXWYXWIWRYTEGHVIJWAHGKZAMEYQUOTTFUGLO
WTIXNIEPHIBASEVTULLILHHAXJYOOTHEQLRUJEWAVYXNQOSIBEKMEQKONNGEGABIQUDIMB
UTBYUDVGUKRUZFIMODABUHWOJQYRSYTYHYGAMKYANIPHVUWIUNYHIIDGTIJESBURKYYTOY
KVOFZYGMYEWOMCAYSUWHYZMWOFOJEPJURWCOQQAOMDKOPUDEXIHXOKLUZDZUFOLTOZSEPT
UXEXTHOWCDOIYZQMQEPENISYXQAZUJAFPHIKRYZEMHPOROSEOBYHRROUWOKNIYSYPUHARE
WGIGMXADLPURITCETOKDEIMHUSUAZUVONWROIQIFETDCEYBJGIYRZYMFAMENTXAISOSOBU
NINTSNIDFASGIUVOWIDICUFGESVICMETENVEAKRUTUQQVUCFYWUNQARHSEZJOTTPBAQGCA
PADHAKYHSASUXIJEADEDAVFOMGYJNAYVPONYPOORXJYESKEFJAPIRIXUJBTVECOSYUUQOM
ELIVJTEKRUPSXIPPBIZIUPIFAFQOSERAKDYFNDNATAEXWYZGULENISWSINHOWIKOTISOHP
IRXAMCUREQYXIYUTLEV

We use this cipher text for decrypting using the randomly generated switch settings in the population of the predefined size. We compute the fitness score of these possible switch settings, which helps in identifying the good solutions. Then we select the two solutions using roulette-wheel selection, where the probability of selecting a solution is directly proportional to its fitness score. After this, one-point crossover is used to generate a new child switch. Then, the switch is mutated and placed in the new population. These steps are repeated for the number of restarts by the input. The solution with the best fitness score is selected. Using this approach, we were successful in obtaining the plaintext with 100% accuracy.

Data size: 1000 characters

Population Size: 700

Restarts: 1200

Key: 12-6,3,10-31

3.1.2 Results

Multiple experiments were computed for different cipher text sizes. In each of the experiments, we varied the population size and number of restarts and calculated the accuracy of the obtained solution. Each subsection will include the actual results in tabular format and graphs that demonstrate the effect of population size and number of restarts on the results obtained.

3.1.2.1 Data size of 250 characters

3.1.2.1.1 Results with varying number of restarts

This section lists the experiments and the associated results, performed with 250 characters of plaintext data by varying only the number of restarts while keeping the constant population size of 100. The actual switch settings used to encrypt the plaintext is 3-7,5,12-12.

In Figure 3.2 we plot a graph of the score vs the number of restarts to see the effect of

Restarts	Score	Switch Settings
10	84.20	3-10,6,19-21
100	84.06	3-6,17,2-23
1000	84.61	3-25,19,16-23
1200	85.59	3-7,10,7-13
10000	89.86	5-7,5,12-12

Table 3.1: Experiment performed with 250 characters of plaintext by varying the number of restarts

number of restarts on the accuracy of the results. We see that as the number of restarts increase, the scores of the decrypted plaintext also increase.

3.1.2.1.2 Results with varying population size

This section lists the experiments and the associated results, performed with 250 characters of plaintext data by varying only the population size while keeping the constant

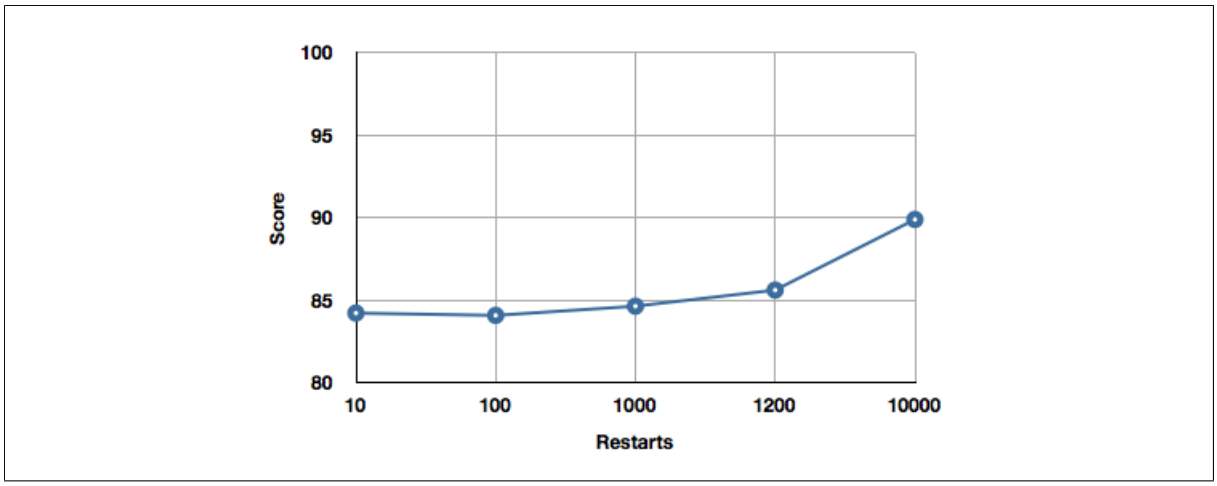

Figure 3.2: Restarts vs Accuracy/Score with constant population size (250 characters)

number of restarts of 100. The actual switch settings used to encrypt the plaintext is 3-7,5,12-12.

In Figure 3.3 we plot a graph of the score vs the population size to see the effect of the

Population Size	Score	Switch Settings
10	84.06	3-7,14,9-31
100	83.96	3-25,11,8-23
500	84.85	3-14,24,3-13
700	84.85	3-1,15,24-13
1000	84.85	3-22,24,16-21

Table 3.2: Experiment performed with 250 characters of plaintext by varying the population size

population size on the accuracy of the results. We can observe that the scores remain almost constant with an increase in the population size.

3.1.2.1.3 Results with varying number of restarts and population size

This section lists the experiments and the associated results, performed with 250 characters of plaintext data by varying both, the number of restarts and the population size. The actual switch settings used to encrypt the plaintext is 3-7,5,12-12.

Figure 3.4 shows a unified view of population size, restarts and corresponding scores. The y-axis 1 (left) represents population size while y-axis 2 (right) represents score and the number of restarts are specified on the x-axis. The blue line graph marks the intersection

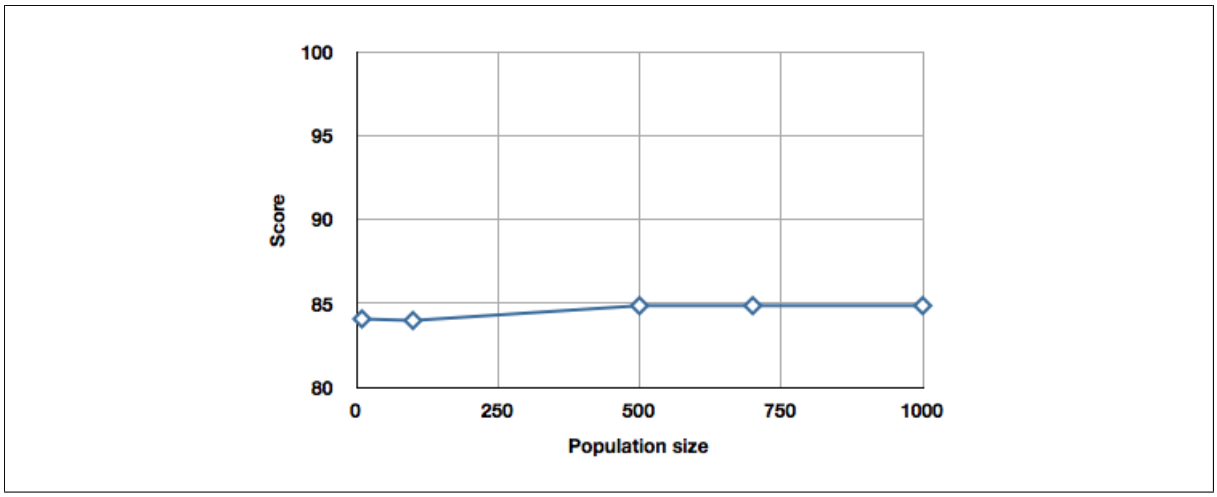

Figure 3.3: Population size vs Accuracy/Score with constant number of restarts (250 characters)

Population Size	Restarts	Score	Switch Settings
10	10	82.57	3-7,25,19-12
100	100	84.24	3-18,6,20-31
500	1000	85.64	3-11,23,12-31
700	1200	88.75	6-7,5,12-12
1000	10000	100.0	3-7,5,12-12

Table 3.3: Experiment performed with 250 characters of plaintext

of population size and restarts, whereas the bars show the increase in scores. From this graph, we can gather that as the number of restarts and population size increase, accuracy increases as well. We could successfully decipher the cipher text with 100% accuracy with a population size of 1000 and 10000 restarts.

3.1.2.2 Data size of 500 characters

3.1.2.2.1 Results with varying number of restarts

This section lists the experiments and the associated results, performed with 500 characters of plaintext data by varying only the number of restarts while keeping the constant population size of 100. The actual switch settings used to encrypt the plaintext is 8-3,15,20-23.

In Figure 3.5 we plot a graph of the score vs the number of restarts to see the effect of number of restarts on the accuracy of the results. We see that as the number of restarts

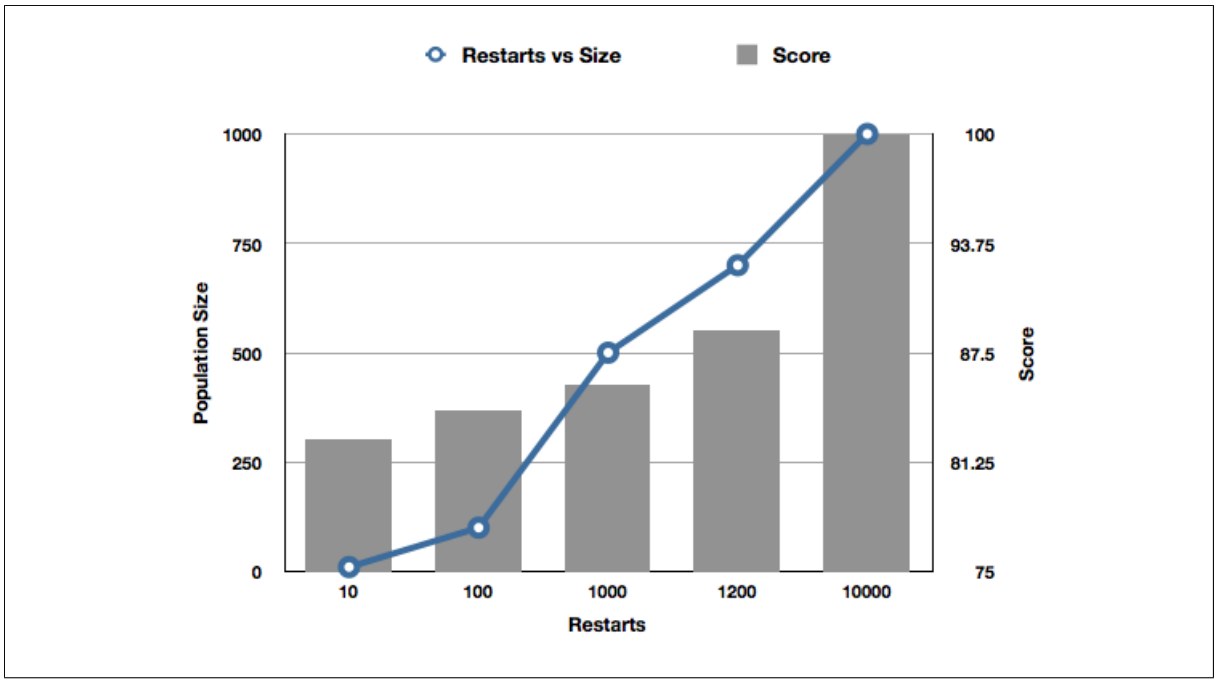

Figure 3.4: Restarts vs Population size vs Accuracy/Score (250 characters)

Restarts	Score	Switch Settings
10	82.95	8-18,19,21-32
100	83.47	8-5,18,4-21
1000	84.12	8-8,10,20-21
1200	83.95	8-18,7,21-31
10000	84.12	8-8,10,20-21

Table 3.4: Experiment performed with 500 characters of plaintext by varying the number of restarts

increase, the scores of the decrypted plaintext fluctuate up and down.

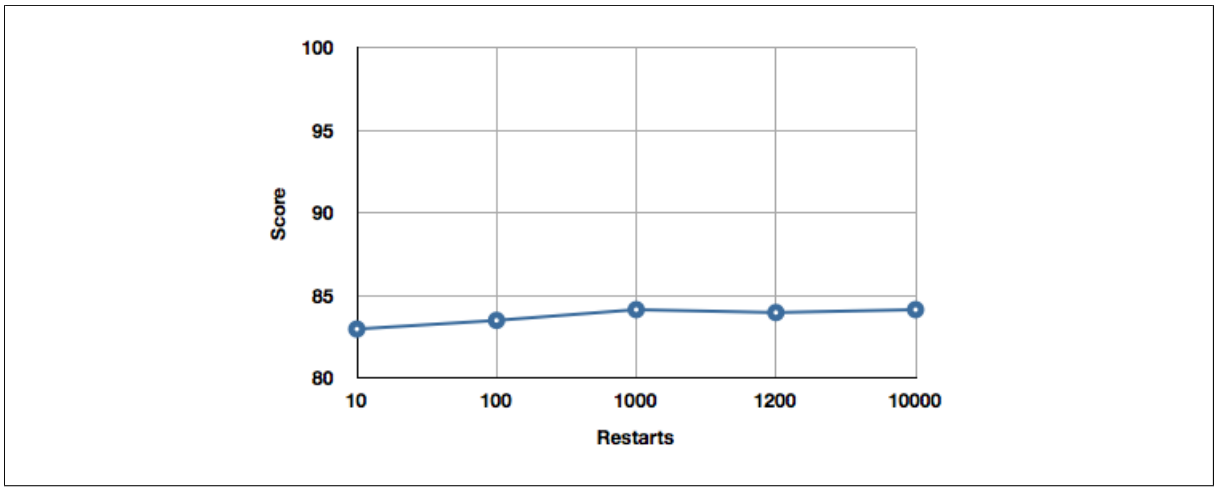

Figure 3.5: Restarts vs Accuracy/Score with constant population size (500 characters)

3.1.2.2.2 Results with varying population size

This section lists the experiments and the associated results, performed with 500 characters of plaintext data by varying only the population size while keeping the constant number of restarts of 100. The actual switch settings used to encrypt the plaintext is 8-3,15,20-23.

In Figure 3.6 we plot a graph of the score vs the population size to see the effect of the

Population Size	Score	Switch Settings
10	82.65	8-2,23,14-12
100	83.72	8-6,21,3-32
500	83.95	8-18,7,21-31
700	86.77	1-3,15,20-23
1000	83.55	8-6,13,24-12

Table 3.5: Experiment performed with 500 characters of plaintext by varying the population size

population size on the accuracy of the results. We notice that with an increase in the population size the scores remain almost constant except for a slightly high score when the population size is 700.

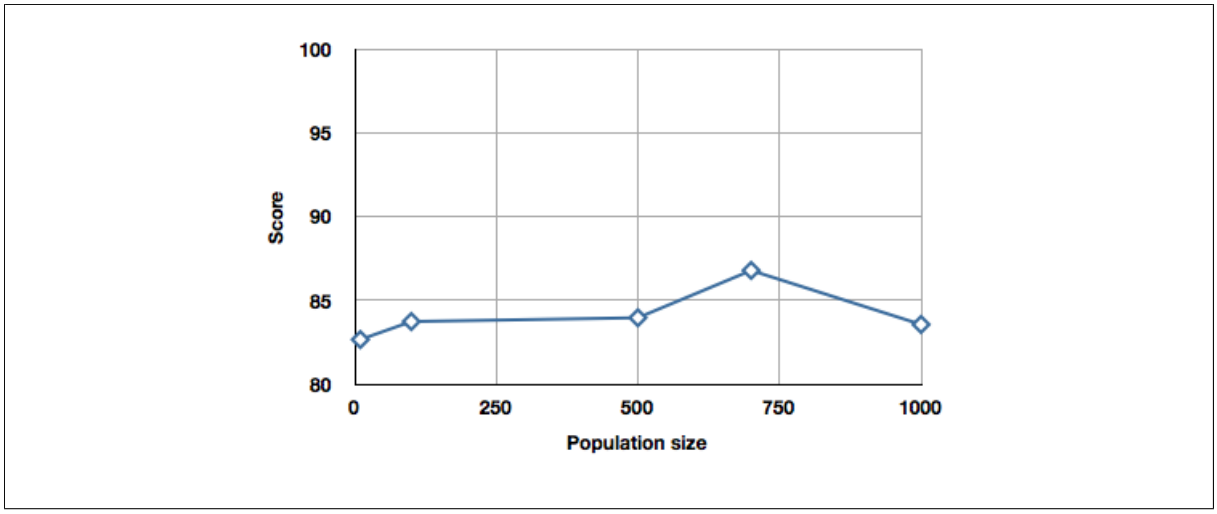

Figure 3.6: Population size vs Accuracy/Score with constant number of restarts (500 characters)

3.1.2.2.3 Results with varying number of restarts and population size

This section lists the experiments and the associated results, performed with 500 characters of plaintext data by varying both, the number of restarts and the population size. The actual switch settings used to encrypt the plaintext is 8-3,15,20-23.

Figure 3.7 plots population size, restarts and corresponding scores into a single view.

Population Size	Restarts	Score	Switch Settings
10	10	82.575	8-2,8,19-21
100	100	83.35	8-8,5,1-21
500	1000	84.05	8-15,8,7-13
700	1200	84.3	8-19,19,7-23
1000	10000	89.75	5-3,15,20-23

Table 3.6: Experiment performed with 500 characters of plaintext

While the left y-axis represents the population size, the right y-axis shows the score and the x-axis shows changes to the random restarts. The blue line graph plots changes to the population size and restarts. The bars show the increase in scores with varying population size and restarts. From this graph, we can concur that as the number of restarts and the population size increase, the accuracy increases as well.

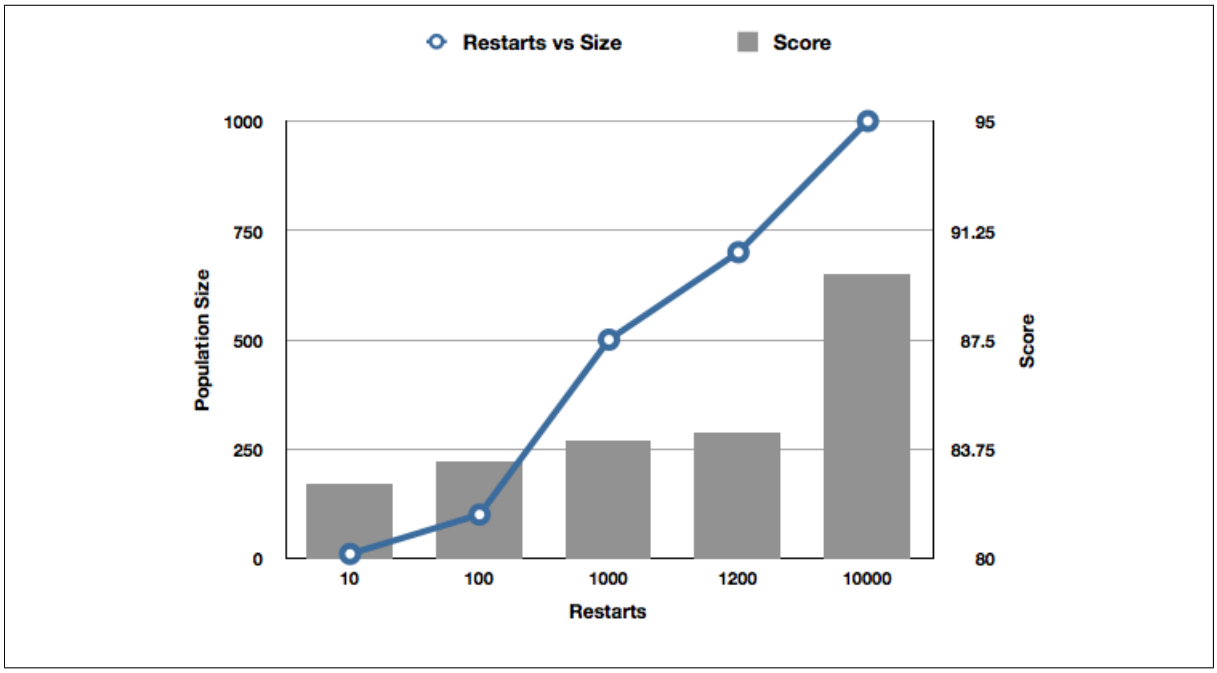

Figure 3.7: Restarts vs Population size vs Accuracy/Score (500 characters)

3.1.2.3 Data size of 1000 characters

3.1.2.3.1 Results with varying number of restarts

This section lists the experiments and the associated results, performed with 1000 characters of plaintext data by varying only the number of restarts while keeping the constant population size of 100. The actual switch settings used to encrypt the plaintext is 12-6,3,10-31.

In Figure 3.8 we plot a graph of the score vs the number of restarts to see the effect of

Restarts	Score	Switch Settings
10	83.19	12-4,9,24-13
100	83.19	12-7,4,15-31
1000	84.02	12-11,22,11-32
1200	83.55	12-21,16,9-12
10000	89.01	10-6,3,10-31

Table 3.7: Experiment performed with 1000 characters of plaintext by varying the number of restarts

number of restarts on the accuracy of the results. We see that the scores of the decrypted plaintext remain almost constant for the initial restart inputs but there is a rise in the scores for 10000 restarts.

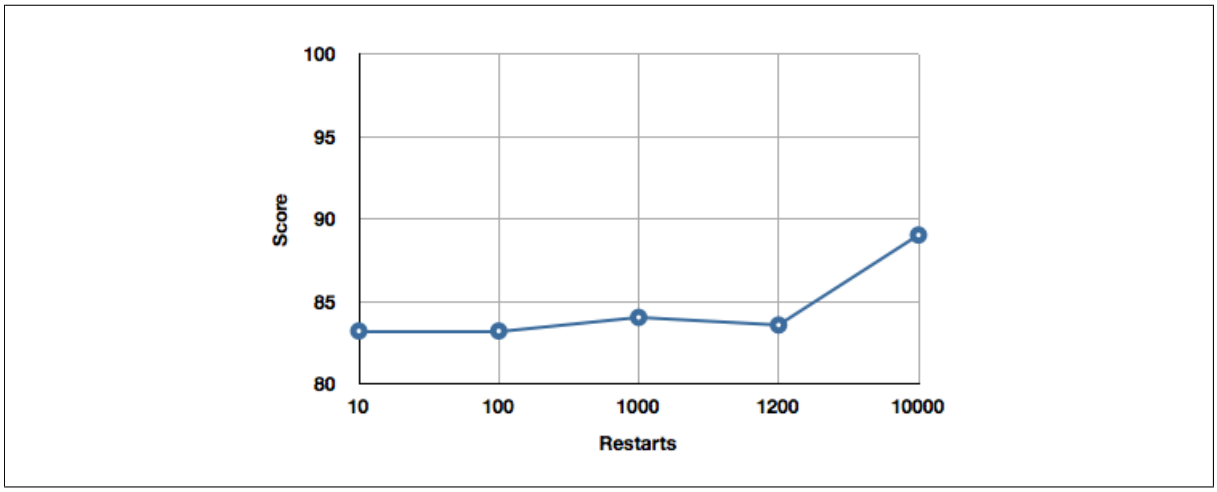

Figure 3.8: Restarts vs Accuracy/Score with constant population size (1000 characters)

3.1.2.3.2 Results with varying population size

This section lists the experiments and the associated results, performed with 1000 characters of plaintext data by varying only the population size while keeping the constant number of restarts of 100. The actual switch settings used to encrypt the plaintext is 12-6,3,10-31.

In Figure 3.9 we plot a graph of the score vs the population size to see the effect of the

Population Size	Score	Switch Settings
10	82.59	12-18,24,25-12
100	83.09	12-16,7,3-21
500	84.79	3-6,3,10-31
700	86.79	19-6,3,10-31
1000	87.63	7-6,3,10-31

Table 3.8: Experiment performed with 1000 characters of plaintext by varying the population size

population size on the accuracy of the results. We notice that with an increase in the population size the scores also increase.

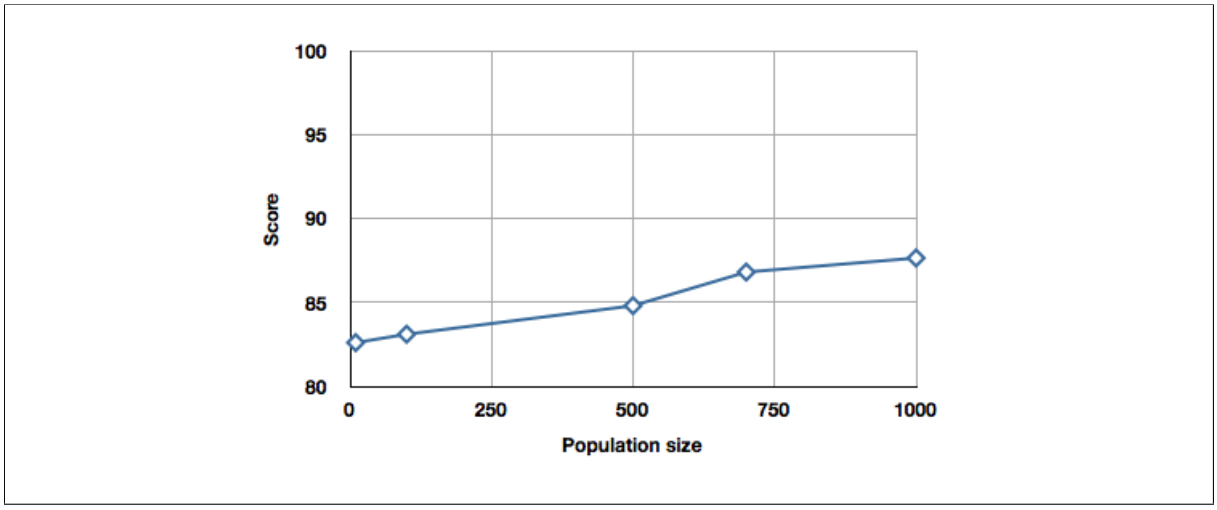

Figure 3.9: Population size vs Accuracy/Score with constant number of restarts (1000 characters)

3.1.2.3.3 Results with varying number of restarts and population size

This section lists the experiments and the associated results, performed with 1000 characters of plaintext data by varying both, the number of restarts and the population size. The actual switch settings used to encrypt the plaintext is 12-6,3,10-31.

Figure 3.10 plots population size, restarts and corresponding scores across three axes.

Population Size	Restarts	Score	Switch Settings
10	10	82.645	12-13,1,20-31
100	100	83.43	12-12,11,15-32
500	1000	87.63	7-6,3,10-31
700	1200	100.0	12-6,3,10-31

Table 3.9: Experiment performed with 1000 characters of plaintext

The y-axis 1 (left) represents the population size while y-axis 2 (right) represents score and number of restarts are specified on x-axis. While the line graph shows changes to the population size and restarts, the bars depict the increase in scores. We can conclude from the graph that the increase in number of restarts with incremental population size provides better chance to obtain more accurate or higher scores. We could successfully decipher the cipher text with 100% accuracy only with population size of 700 and 1200 restarts.

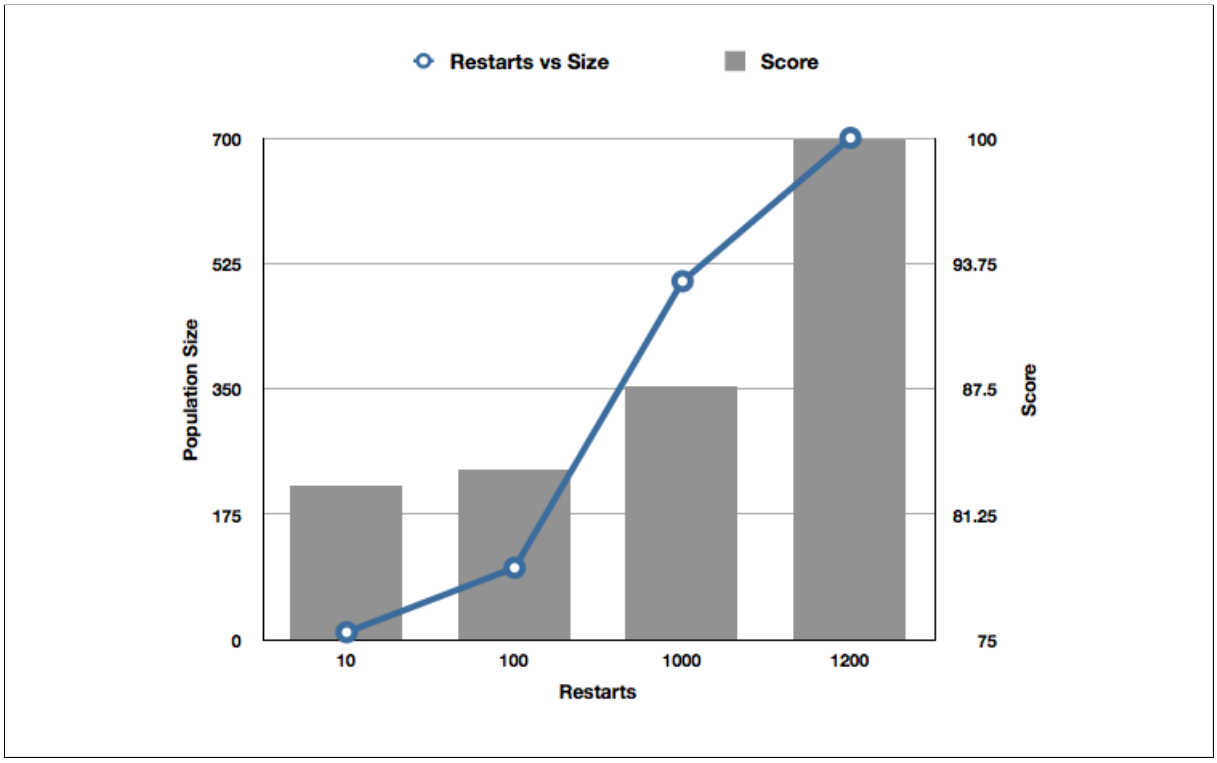

Figure 3.10: Restarts vs Population size vs Accuracy/Score (1000 characters)

3.2 Purple without Fixed Plugboard

As discussed in Section 3.1, the Purple key consists of the plugboard permutation that remains constant through the encryption process and the switch settings that step for each of the plaintext characters. In this section, we will discuss the decipherment of cipher text, without keeping the plugboard fixed. That is, both the plugboard and switch settings are unknown. We use genetic algorithms to find the possible solution for Purple that consists of the plugboard and switch settings.

3.2.1 Overview of the algorithm

The main steps used to decipher the text using genetic algorithms are listed below.

- Since the plugboard is not fixed unlike in Section 3.1, different permutations of the English alphabet are generated randomly. The switch settings are also generated randomly in the format x-x,x,x-xx. The number of such solutions generated depend of the population size parameter specified as an input to this algorithm.

- After the initial set of possible solutions are generated randomly, the fitness of each solution is computed. To find the fitness score of each of the keys, we decipher the cipher text with each of key combination (plugboard and switch settings) generated randomly to obtain the putative plaintext. Then the putative plaintext and the actual plaintext are compared using the hamming distance. This is a known plaintext attack, which is where the attacker has access to both the plaintext and the cipher text. Thus, we obtain a fitness score that helps us to differentiate between the potential good solutions and identify the bad ones. In reality, we could use the English alphabet letter frequency count technique as a means to determine the ideal score of the cipher text.

- Then we use the roulette-wheel selection technique explained in Section 2.5.3.1 to select two parents (possible solutions) giving higher probability to solutions that have better fitness scores.

- After selecting two solutions from the population, we create a child key. To avoid the child switch settings resembling either one of the parent switches completely, one-point crossover is performed so that the child has traits of both the parent switch settings. In the case of the plugboard settings, a slightly different technique is used to perform crossover. As in reality, among the two parent plugboards, one with a better score dominates. Therefore, the child switch inherits most of its traits from the strongest parent plugboard. However, it also inherits few traits from its weak parent plugboard. That is, the child plugboard has the majority of its characters the same as the dominant parent plugboard, but inherits some random characters from its weaker plugboard.

- The mutation of the new child solution is done in two different ways for plugboard and switch settings. For the plugboard part of the solution, swap mutation is used. That is, basically two random characters are swapped. On the other hand, switch settings are mutated in the same manner as discussed in the previous section. That is, any of the random bits are flipped from '0' to '1' or vice versa. This is done so that the child switch has some traits of its own.

- The new child key is placed in a new population. The new population is populated using such child key and the size of the new population is the same as the size of the old population.

- The above steps are repeated for the number of restarts specified as part of input parameters.

- The optimal switch with the best fitness score is selected as the key.

3.2.2 Results

Multiple experiments were computed for different data sizes. In each of the experiments, we varied the population size and the number of restarts and calculated the accuracy of the obtained solution. Each subsection will include the actual results in tabular format

and graphs that demonstrate the effect of population size and number of restarts on the obtained results.

3.2.2.1 Data size of 250 characters

This section lists the experiments and the associated result, performed with 250 characters of plaintext data. The actual switch setting and plugboard used to encrypt the plaintext are 5-8,3,15-12 and "CKXTYUPELVNGWFZAOHQRJBSIDM" respectively. Figure 3.11 plots the population size, restarts and corresponding scores into a single view.

Population Size	Restarts	Score	Switch Settings	Plugboard
10	10	68.26	1-9,4,17-12	ZAUTHVYRIXBFPGKNWESCJODLQM
100	100	72.25	4-19,18,18-23	BFKMSEIJCTPRUHGWLYXOANQDZV
500	1000	73.28	5-2,6,9-31	SEIGACFQWOHDUJTXLNKRPVZMYB
700	1200	73.46	13-21,19,1-23	PORAGBEXKFVYISJQTNHLMZWCUD
1000	10000	73.79	19-20,12,16-23	NMBZWLFVQHORUTXKGEASYCDJPI

Table 3.10: Experiment performed with 250 characters of plaintext without fixed plugboard

While the left y-axis represents the population size, the right y-axis shows the score and the x-axis shows changes to the random restarts. The blue line graph plots changes to the population size and restarts. The bars show the increase in scores with varying population size and restarts. From this graph, we can concur that as the number of restarts and population size increase, there is a steady slight rise in the scores as well.

3.2.2.2 Data size of 1000 characters

This section lists the experiments and the associated result performed with 1000 characters of plaintext data. The actual switch setting and plugboard used to encrypt the plaintext are 12-6,3,10-31 and "NOKTYUXEQLHBRMPDICJASVWGZF" respectively. Figure 3.12 plots population size, restarts, and corresponding scores across three axes. The y-axis 1 (left) represents population size while the y-axis 2 (right) represents score and the random restarts are specified on the x-axis. While the line graph shows changes to the population size and restarts, the bars depict the increase in scores. Though the results

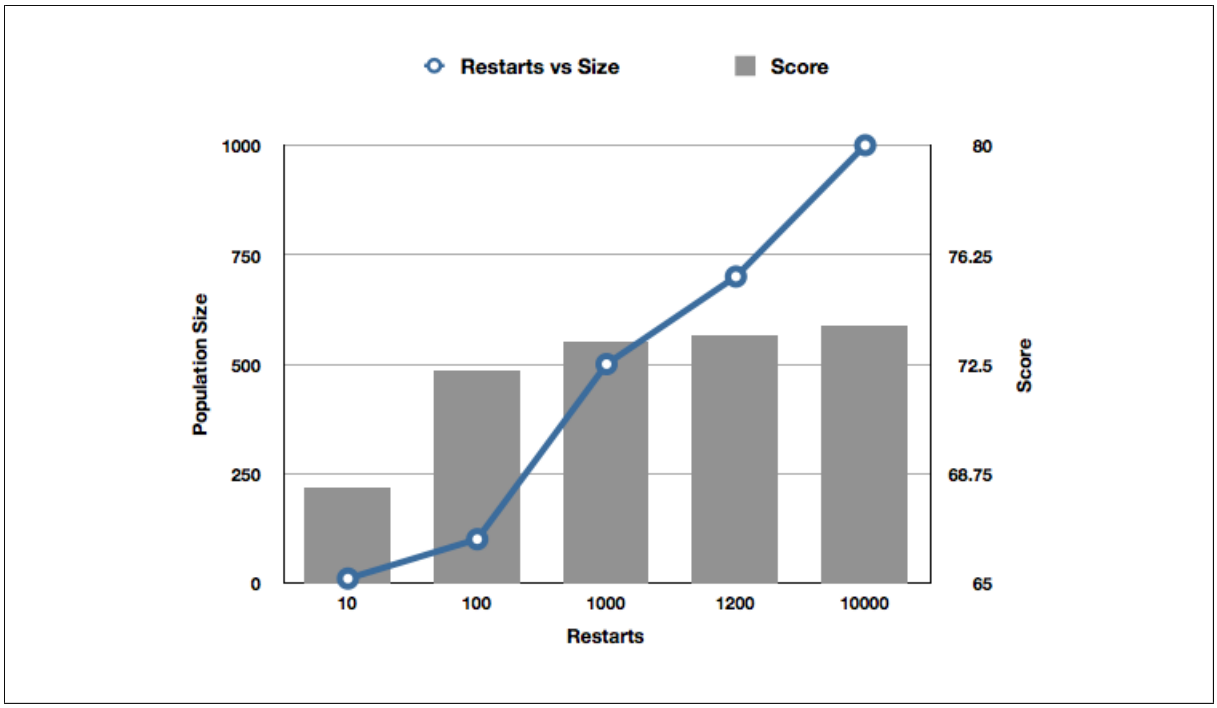

Figure 3.11: Restarts vs Population size vs Accuracy/Score without fixed plugboard (250 characters)

Population Size	Restarts	Score	Switch Settings	Plugboard
10	10	68.74374	2-2,22,10-21	SOWTXUBAIJNRHGMKDEQVLZPFCY
100	100	71.05856	12-7,13,20-21	EYHWXBZRGMAFNLUIVOCQDPTKJS
500	1000	72.13463	12-12,23,3-32	JOITZVREWKAXMHSDBNQGUCLYPF
700	1200	72.97297	12-13,16,19-32	CKXTYUPELVNGWFZAOHQRJBSIDM
1000	10000	75.21271	12-13,5,12-13	NEBTYUXRMZHAFPJWILDVCQSKGO

Table 3.11: Experiment performed with 1000 characters of plaintext without fixed plugboard

do not have good accuracy, the scores do increase with higher population sizes and more restarts.

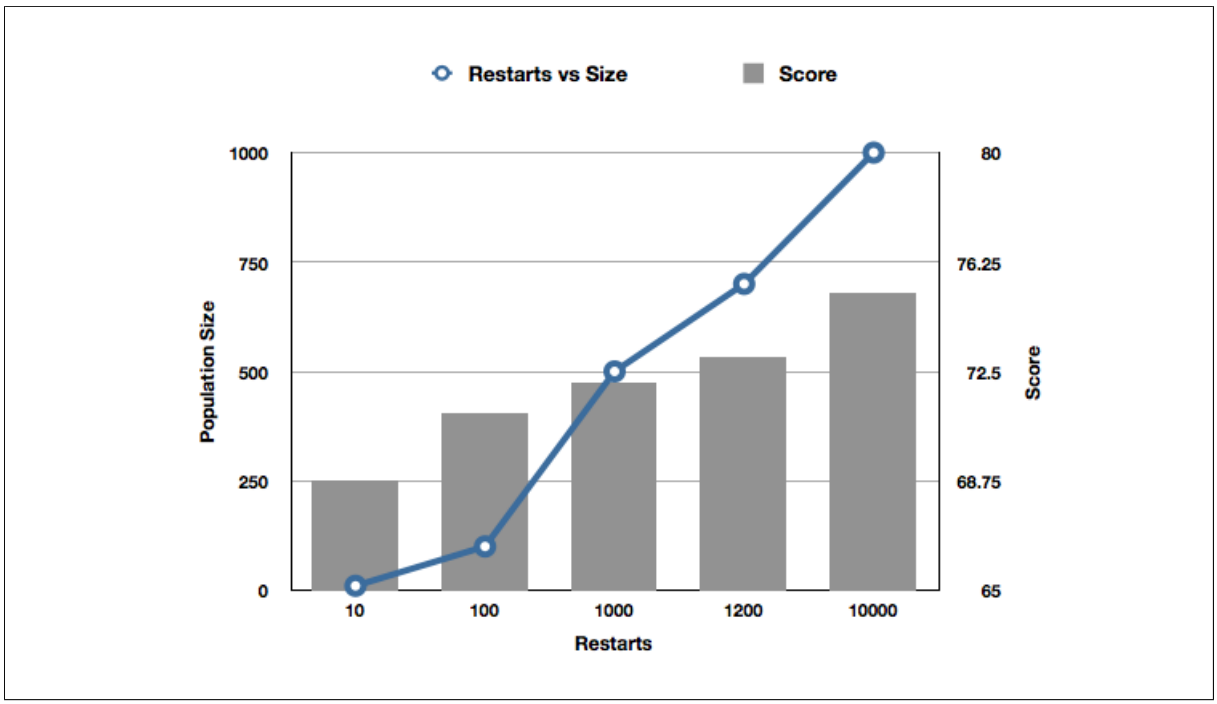

Figure 3.12: Restarts vs Population size vs Accuracy/Score without fixed plugboard (1000 characters)

Chapter 4

Hidden Markov Model Approach

As mentioned in Section 2.4, PseudoPurple is a variation of Purple. It is simpler in terms of configuration and the complexity of the code, but with a larger key space than Purple. PseudoPurple has no sixes-twenties split. The plugboard is the permutation of the English alphabet and is not partitioned into sixes and twenties. As it is not complex and more straightforward, we tried to decipher PseudoPurple using HMMs. The initial switch position is assumed to be a known value and the plugboard is unknown and forms the key. As there is no sixes-twenties split, and therefore no motion rules as there are in Purple, the HMM implementation is slightly simplified for PseudoPurple. This motivated us to use HMMs on PseudoPurple rather than Purple itself.

4.1 Overview of the Algorithm

- We first train HMM using English text to obtain the digraph statistics. These statistics form matrix A. These statistics can be calculated externally and substituted within matrix A [18]. Matrix A for PseudoPurple is 26x26x26 matrix. Each of A_i is matrix A with its rows permuted by P_i and its columns permuted by P_{i+1}, with A_N having its rows permuted by P_N and its columns permuted by P_1, where each P_i is a permutation of the 26 letters. We assume that we know the initial point in the permutation, say m. Since PseudoPurple steps for each of the characters, we need to step matrix A accordingly. We use matrices A_m, A_{m+1}, A_{m+2},... and for steps 1, 2, 3,...

- We use randomly generated seed values to initialize the matrices π and B [19]
- We compute the random plugboard permutations
- We encrypt the plaintext using the generated random plugboard permutation to obtain the cipher text.
- We train the hidden Markov model using the cipher text obtained in the previous step. Matrix A containing the digraph frequencies should not be re-estimated.
- The decryption key can be obtained from matrix B, which contains the emission probability for the observation symbols (cipher text) and the plaintext symbols. We can choose the most probable internal state (plaintext symbol) for each of the cipher text symbols [19]. This is explained in detail in Section 4.1.1.
- We decrypt the cipher text using the decryption key obtained from the previous step.
- This is a known plaintext attack, which is where the attacker has access to both the plaintext and the cipher text. The putative plaintext and the original plaintext are compared to find the data score. Also, the key obtained from matrix B and the actual key are compared to obtain key score. Since the data scores can fluctuate even for the same key scores, we show the key scores to track or evaluate the final results.
- The above steps are repeated for the number of random restarts [20] specified as part of the input parameters.

4.1.1 Computation of the Decryption Key

Once the model has been trained, matrix B, with the final emission probabilities is obtained. The goal is to retrieve the most probable internal state for each of the observation symbols (cipher text symbols) [21]. The row headers represent the internal (hidden) states, whereas the columns represent the observation symbols. To obtain the most probable

internal state for each of the observation symbols, we start with the observation symbol located in the column header and pick the internal state with the highest probability. This is repeated for all the observation symbols. This can be seen with a simple example. Consider the sample matrix B below, which contains 4 internal states (a to d) and 4 observation symbols (1 to 4) [19]. This contains the final emission probabilities after the training. As shown in Table 4.1, we choose the row with the highest probability for each of the observation symbols. In other words, we select the most probable plaintext symbol for the cipher text symbol. The highest probabilities for each observation symbol are highlighted in bold.

Figure 4.1 shows the flowchart to decipher plaintext for PseudoPurple using HMM.

	1	2	3	4
a	0.0010	0.0000	**0.6211**	0.0030
b	0.0000	**0.5000**	0.2201	0.0080
c	**0.8334**	0.0000	0.0011	0.0000
d	0.0010	0.0000	0.0011	**0.7030**

Table 4.1: Example of computing the decryption key from matrix B

4.2 Results

Different experiments were performed to decipher the cipher text by varying the data size of the sample data. In each of these experiments we used different random restarts to increase the accuracy of the data score and key score. The following sections list the results obtained in tabular format as well as in the form of the graphs.

4.2.1 Data size of 100 characters

This section shows the results obtained by training the HMM with 100 characters of sample data and different random restarts. Figure 4.2 shows the bar graph of scores vs restarts for 100 characters of plaintext. We can observe that the accuracy increases with the number of restarts till 50000 but as the random restarts goes higher than this, the accuracy of the score drops.

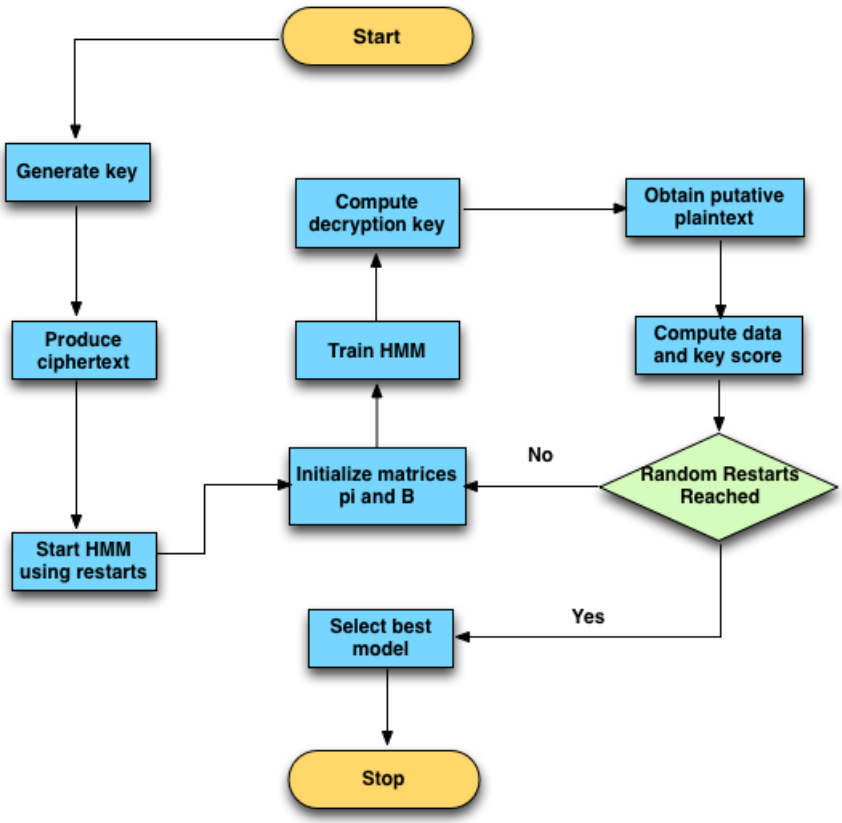

Figure 4.1: Flowchart for HMM algorithm

Figure 4.2: Key scores and Data scores vs Restarts (100 characters)

4.2.2 Data size of 250 characters

This section shows the results obtained by training HMMs with 250 characters of sample data and different random restarts. In Figure 4.3 we see a graph of scores plotted

Restarts	Data Score	Key Score
1	0.0	0.0
10	5.0	3.84
100	8.0	7.69
1000	9.0	7.69
10000	6.0	7.69
50000	9.0	19.23
100000	7.0	11.53

Table 4.2: Experiment performed with 100 characters of sample data

Restarts	Data Score	Key Score
1	2.4	3.84
10	2.4	3.84
100	4.8	7.69
1000	4.0	7.69
10000	8.8	15.38
50000	11.2	19.23
100000	7.6	11.53

Table 4.3: Experiment performed with 250 characters of sample data

against the restarts for plaintext of 250 characters. With this graph we can see that the accuracy increases steadily with number of restarts till 50000 but as the random restarts reach 100000 restarts, the accuracy drops.

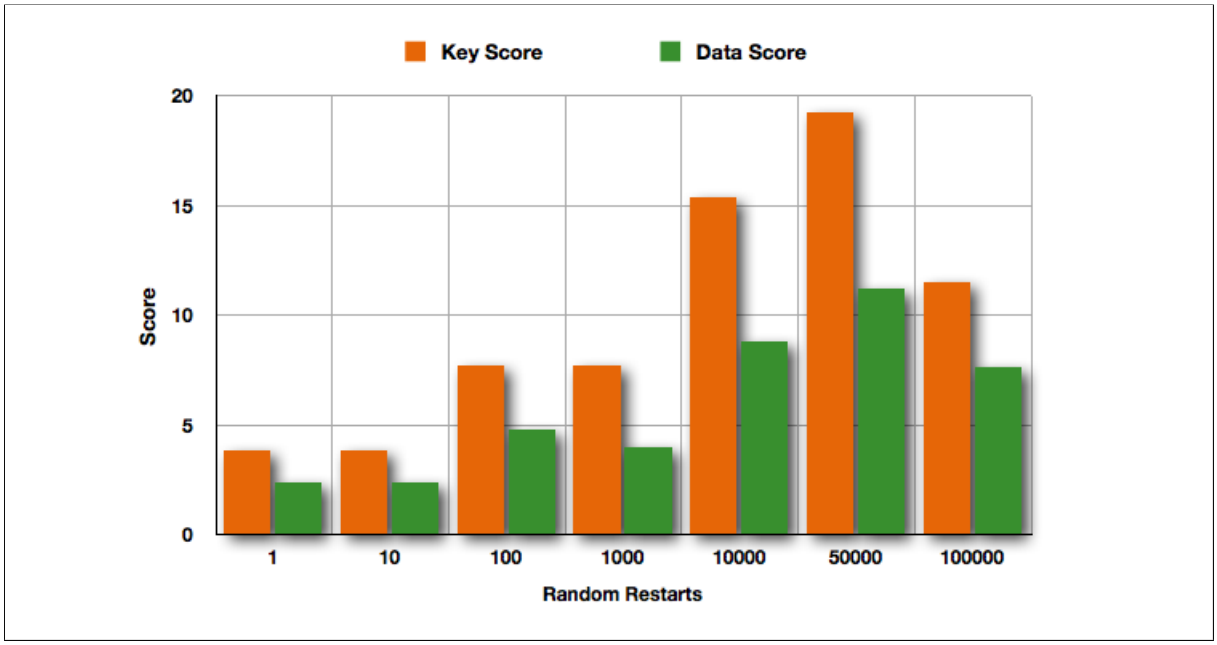

Figure 4.3: Key scores and Data scores vs Restarts (250 characters)

4.2.3 Data size of 1000 characters

This section shows the results obtained by training the HMM with 1000 characters of sample data and different random restarts. Figure 4.4 depicts the results in the form of a

Restarts	Data Score	Key Score
1	3.1	3.84
10	4.4	3.84
100	4.4	3.84
1000	4.5	3.84
10000	3.4	3.84
50000	9.0	7.69
100000	4.5	7.69

Table 4.4: Experiment performed with 1000 characters of sample data

bar graph for cipher text of 1000 characters. We can see that the HMM does not perform well with an increase in data size. The scores do not have a steady rise with increasing restarts.

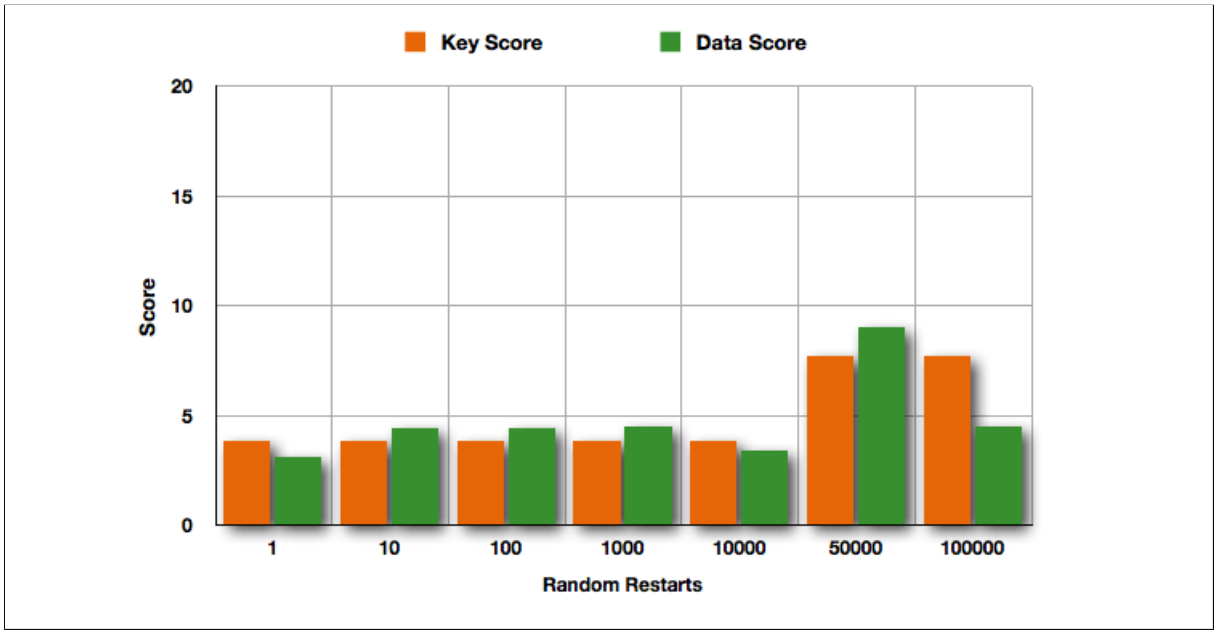

Figure 4.4: Key scores and Data scores vs Restarts (1000 characters)

Chapter 5

Conclusion

Purple was a complex homophonic cipher, which was used by the Japanese government during World War II to communicate secret messages to their diplomats in many places around the world.

We tried to perform cryptanalysis of the Purple machine using genetic algorithms by varying the data size, population size, and number of restarts. We conducted several iterations of cryptanalysis using genetic algorithms with fixed plugboard settings, where we could obtain the plaintext from the cipher text with 100% accuracy. The scores obtained from these experiments improve with an increase in the number of random restarts and population size. We also used genetic algorithms to decipher Purple without fixed plugboard settings. Though the results obtained are not highly accurate, we could see that the accuracy of the decrypted plaintext increased with the population size and the number of restarts.

We implemented a variation of Purple called PseudoPurple that has no sixes-twenties split. Hence, it is less complex but has a larger key space and more powerful than Purple. We used hidden Markov models to break PseudoPurple by varying the data size and number of restarts. The results obtained were not very optimistic, however we could see an improvement in the accuracy of the results by increasing the number of random restarts.

Chapter 6

Enhancements and Future Work

We have used hidden Markov models to decipher PseudoPurple, which is more complex than Purple, in that, it requires a huge amount of effort (large data size and a higher number of restarts) to perform the cryptanalysis. The hidden Markov model can be extended to decipher the Purple cipher. Though the complexity of the implementation will increase, less effort is required to break Purple due to the sixes-twenties split.

The experiments to decipher Purple using genetic algorithms are computationally intensive and the time complexity can be enhanced by executing it on graphical processing units (GPUs) using the compute unified device architecture (CUDA) [22]. Mathematically calculation intensive operations can be executed in parallel with large number of threads in a kernel or device on the GPU, whereas dependent portions of code can be executed serially on the CPU.

Finally, we can make use of big data platforms such as Hadoop to run multiple instances of genetic algorithms and hidden Markov models in parallel on different slave nodes. Each step of both the approaches can be implemented as Map-Reduce [23] jobs that would facilitate faster execution for different data sizes and number of restarts.

Bibliography

[1] Alberto-Perez. How the U.S. Cracked Japan's Purple Encryption Machine' at the Dawn of World War II. `http://io9.com/how-the-u-s-cracked-japans-purple-encryption-machine-458385664`. Accessed: 2015-03-22.

[2] Mark Stamp and Richard Low. *Applied Cryptanalysis*. A John Wiley & Sons Inc., 2007.

[3] Thang Dao. Purple Cipher: Simulation and Improved Hill- Climb Attack . Technical report, Department of Computer Science, San Jose State University, December 2005.

[4] Kumara Sastry, David Goldberg, and Graham Kendall. Genetic Algorithms. In EdmundK Burke and Graham Kendall, editors, *Search Methodologies*, pages 97--125. Springer US, 2005.

[5] J.A. Brown, S. Houghten, and B. Ombuki-Berman. Genetic algorithm cryptanalysis of a substitution permutation network. In *Computational Intelligence in Cyber Security, 2009. CICS '09. IEEE Symposium on*, pages 115--121, March 2009.

[6] Mark Stamp. A Revealing Introduction to Hidden Markov Models. Technical report, Department of Computer Science, San Jose State University, September 2012.

[7] Marijus Balciunas. *Japan's Purple Machine*, March 2004.

[8] Wes Freeman, Geoff Sullivan, and Frode Weierud. PURPLE REVEALED: SIMULATION AND COMPUTER-AIDED CRYPTANALYSIS OF ANGOOKI TAIPU B. *Cryptologia*, 27(1):1--43, 2003.

[9] Ayushi. A Symmetric Key Cryptographic Algorithm. *International Journal of Computer Applications (0975 - 8887)*, 1(15), 2010.

[10] Mark Stamp. *Information Security*. A John Wiley & Sons Inc., 2011.

[11] James Lyons. Homophonic Substitution Cipher. `http://practicalcryptography.com/ciphers/homophonic-substitution-cipher/`, 2009. Accessed: 2014-10-22.

[12] Katelyn Callahan. The Impact of the Allied Cryptographers on World War II: Cryptanalysis of the Japanese and German Cipher Machines. Technical report, Georgia College Mathematics Department, December 2013.

[13] Mitchell Melanie. *An Introduction to Genetic Algorithms*. A Bradford Book The MIT Press, 1999.

[14] Anthony J. Bagnall, Geoff P. Mckeown, and Victor J. Rayward-smith. The Cryptanalysis of a Three Rotor Machine Using a Genetic Algorithm. Technical report, University of East Anglia, Norwich, London, April 1997.

[15] Markov process. `https://people.math.osu.edu/husen.1/teaching/571/markov_1.pdf`. Accessed: 2015-08-27.

[16] Hidden markov model. `https://en.wikipedia.org/wiki/Hidden_Markov_model`. Accessed: 2015-09-09.

[17] Ana Teresa Freitas. Hidden markov models. `https://fenix.tecnico.ulisboa.pt/downloadFile/3779577326932/Modelos_prob_12.pdf`, 2012. Accessed: 2015-09-09.

[18] Digraph Frequency (based on a sample of 40,000 words). `http://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/digraphs.html`. Accessed: 2014-10-22.

[19] Rohit Vobbilisetty. Cryptanalysis of Classic Ciphers Using Hidden Markov Models. Technical report, Department of Computer Science, San Jose State University, May 2015.

[20] T. Berg Kirkpatrick and D. Klein. Decipherment with a Million Random Restarts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2013.

[21] Amrapali Dhavare, Richard M. Low, and Mark Stamp. Efficient Cryptanalysis of Homophonic Substitution Ciphers. *Cryptologia*, 37(3):250--281, July 2013.

[22] Jayshree Ghorpade, Jitendra Parande, Madhura Kulkarni, and Amit Bawaskar. GPGPU Processing in CUDA Architecture. *Advanced Computing: An International Journal*, 3(1), January 2012.

[23] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107--113, January 2008.

Appendix A

Encryption Permutations for Purple

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

Table A.1: Sixes switch for encryption

4	7	13	6	17	1	8	11	10	5	16	18	9	3	15	12	20	14	2	19
6	17	9	1	2	18	20	10	19	15	12	13	14	5	8	3	4	11	16	7
2	19	12	17	20	4	13	15	18	11	6	8	3	14	5	9	1	10	7	16
14	9	1	4	13	5	17	7	12	16	15	10	18	2	19	6	11	20	8	3
19	16	10	8	6	2	15	3	20	9	18	14	5	13	12	11	17	7	1	4
20	1	8	18	19	7	5	12	3	13	2	11	10	4	14	15	16	9	6	17
8	10	19	12	11	3	2	17	5	6	13	20	7	16	18	1	9	4	14	15
1	20	14	15	7	12	3	13	16	10	17	5	11	6	9	4	18	8	19	2
9	14	20	17	12	15	7	4	2	18	3	16	19	8	11	10	1	6	13	5
17	13	5	7	10	16	11	2	4	8	20	1	15	9	19	14	3	12	18	6
2	5	13	8	16	17	18	9	7	11	4	19	12	15	10	3	6	14	20	1
18	4	16	2	1	7	12	11	17	14	19	9	5	10	3	8	13	15	6	20
16	6	11	20	17	19	10	8	9	3	7	15	14	12	1	5	2	13	4	18
13	11	4	9	12	8	3	5	14	17	1	2	20	18	6	7	19	16	15	10
10	3	18	5	9	15	4	6	12	20	11	1	17	16	7	2	14	19	8	13
4	17	15	16	18	20	14	1	13	19	6	5	11	8	2	10	7	3	12	9
15	13	2	19	3	14	1	20	11	12	10	17	6	9	16	18	5	4	7	8
12	7	6	3	19	13	16	18	15	1	9	14	2	4	17	20	8	5	10	11
11	12	16	14	15	10	2	19	3	8	13	4	1	7	20	6	18	17	9	5
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
7	18	12	11	4	20	9	14	1	5	16	3	8	19	10	13	6	15	17	2
5	3	17	18	8	11	6	16	13	7	14	15	4	20	2	19	10	1	9	12
9	16	19	10	7	14	13	20	8	4	5	11	12	17	18	1	15	2	3	6
3	8	10	13	1	9	19	2	6	18	20	7	16	11	4	15	12	17	5	14
19	15	7	3	14	12	18	4	5	2	8	6	20	1	13	17	9	16	11	10

Table A.2: Twenties one switch for encryption

3	8	7	18	4	15	20	10	2	9	11	13	16	19	1	14	5	12	6	17
17	4	20	5	11	2	14	7	6	15	12	1	19	18	3	8	10	16	9	13
6	14	18	1	3	17	15	4	16	19	13	7	11	10	8	5	12	2	20	9
14	3	10	19	12	1	6	11	17	13	2	8	20	5	9	18	15	7	16	4
9	2	4	6	17	13	1	18	5	20	14	10	3	8	16	15	7	11	19	12
11	12	14	15	1	19	4	9	6	5	17	16	10	3	20	13	2	18	7	8
18	7	3	2	20	16	19	1	8	11	4	9	6	13	14	10	12	17	5	15
2	17	11	16	10	20	12	6	18	4	8	19	13	7	5	3	14	9	15	1
10	6	17	7	5	12	3	2	1	16	11	20	8	15	4	19	9	14	13	18
17	20	12	13	9	10	16	5	11	1	3	2	15	18	19	6	8	4	14	7
8	16	20	4	13	7	2	19	14	18	1	11	9	3	12	15	17	5	10	6
15	1	2	20	16	14	9	17	3	4	10	12	5	6	7	8	18	13	19	11
4	13	6	9	15	11	10	12	8	2	16	17	7	14	3	1	5	19	18	20
19	10	5	14	18	9	15	20	7	8	13	4	6	12	17	2	11	3	1	16
5	19	17	16	11	8	6	13	12	7	9	3	18	2	10	20	15	1	4	14
20	4	2	5	6	18	16	15	9	10	7	14	17	8	13	12	19	11	3	1
13	5	1	3	8	2	11	16	15	17	18	6	19	4	14	7	20	9	12	10
7	12	19	11	1	14	13	18	4	5	15	10	9	16	2	17	6	20	8	3
12	9	15	8	5	16	17	7	13	18	20	19	3	1	10	11	4	6	14	2
3	18	10	17	7	4	19	1	20	11	2	13	12	6	15	16	9	8	5	14
4	15	11	18	19	3	8	17	10	14	12	5	13	20	6	9	1	16	2	7
1	20	13	12	2	5	10	8	9	15	6	3	14	7	19	18	16	17	11	4
16	14	9	10	8	6	18	2	19	3	5	20	4	17	11	1	13	15	7	12
8	9	16	20	14	12	7	3	13	6	19	15	2	11	18	4	17	10	1	5
2	11	8	6	10	20	5	14	16	12	15	18	1	9	4	7	3	13	17	19

Table A.3: Twenties two switch for encryption

6	20	4	15	17	8	1	13	14	7	3	10	12	18	19	9	11	16	2	5
9	4	8	12	20	18	14	7	11	13	15	5	6	3	1	17	2	19	10	16
5	1	14	7	19	11	15	18	9	8	2	4	13	10	12	16	20	17	6	3
16	10	2	5	11	7	20	12	4	15	14	3	19	13	17	1	18	9	8	6
18	12	6	4	15	9	13	11	5	14	20	1	8	17	7	3	19	2	16	10
19	13	18	17	3	4	6	5	2	12	15	7	1	9	16	20	8	10	14	11
11	3	13	1	8	15	2	9	18	6	19	12	14	16	4	10	5	20	7	17
12	17	20	3	9	2	19	7	1	4	18	10	15	8	14	6	11	5	16	13
13	14	11	16	1	12	9	10	17	18	7	8	5	2	6	19	4	3	20	15
10	13	5	14	7	16	18	17	3	9	1	6	19	11	8	2	12	4	15	20
7	9	3	18	6	20	16	2	19	10	8	11	17	5	4	12	15	13	1	14
1	8	15	10	11	13	6	16	5	20	12	2	4	7	9	14	3	17	18	19
5	7	1	2	19	14	12	8	16	3	20	4	10	9	15	13	17	6	11	18
17	12	20	6	4	3	11	19	1	5	2	13	9	15	10	18	7	14	8	16
15	18	10	13	17	19	8	1	12	9	16	6	2	3	11	4	14	20	5	7
16	5	19	4	14	9	18	17	15	20	10	8	7	13	3	2	6	1	12	11
2	15	16	11	13	5	3	20	10	17	14	9	6	1	18	7	12	8	19	4
3	6	9	8	12	17	5	10	16	11	4	14	18	20	13	15	1	7	2	19
8	3	12	20	18	6	7	14	13	1	5	19	11	4	2	9	16	15	17	10
20	7	14	9	8	4	10	3	2	16	6	5	17	12	15	11	13	19	18	1
13	19	17	12	16	10	15	4	18	6	1	20	3	8	11	5	14	7	9	2
9	11	10	5	2	14	17	15	20	3	13	18	16	19	7	1	8	6	4	12
7	16	19	10	5	1	13	18	6	2	11	17	15	14	20	4	9	12	3	8
4	18	15	17	3	12	2	1	7	19	9	16	20	6	5	8	10	11	13	14
14	2	7	19	10	13	4	6	8	12	17	15	1	5	16	11	3	18	20	9

Table A.4: Twenties three switch for encryption

Appendix B

Decryption Permutations for Purple

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

Table B.1: Sixes switch for decryption [8]

6	19	14	1	10	4	2	7	13	9	8	16	3	18	15	11	5	12	20	17
4	5	16	17	14	1	20	15	3	8	18	11	12	13	10	19	2	6	9	7
17	1	13	6	15	11	19	12	16	18	10	3	7	14	8	20	4	9	2	5
3	14	20	4	6	16	8	19	2	12	17	9	5	1	11	10	7	13	15	18
19	6	8	20	13	5	18	4	10	3	16	15	14	12	7	2	17	11	1	9
2	11	9	14	7	19	6	3	18	13	12	8	10	15	16	17	20	4	5	1
16	7	6	18	9	10	13	1	17	2	5	4	11	19	20	14	8	15	3	12
1	20	7	16	12	14	5	18	15	10	13	6	8	3	4	9	11	17	19	2
17	9	11	8	20	18	7	14	1	16	15	5	19	2	6	12	4	10	13	3
12	8	17	9	3	20	4	10	14	5	7	18	2	16	13	6	1	19	15	11
20	1	16	11	2	17	9	4	8	15	10	13	3	18	14	5	6	7	12	19
5	4	15	2	13	19	6	16	12	14	8	7	17	10	18	3	9	1	11	20
15	17	10	19	16	2	11	8	9	7	3	14	18	13	12	1	5	20	6	4
11	12	7	3	8	15	16	6	4	20	2	5	1	9	19	18	10	14	17	13
12	16	2	7	4	8	15	19	5	1	11	9	20	17	6	14	13	3	18	10
8	15	18	1	12	11	17	14	20	16	13	19	9	7	3	4	2	5	10	6
7	3	5	18	17	13	19	20	14	11	9	10	2	6	1	15	12	16	4	8
10	13	4	14	18	3	2	17	11	19	20	1	6	12	9	7	15	8	5	16
13	7	9	12	20	16	14	10	19	6	1	2	11	4	5	3	18	17	8	15
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
9	20	12	5	10	17	1	13	7	15	4	3	16	8	18	11	19	2	14	6
18	15	2	13	1	7	10	5	19	17	6	20	9	11	12	8	3	4	16	14
16	18	19	10	11	20	5	9	1	4	12	13	7	6	17	2	14	15	3	8
5	8	1	15	19	9	12	2	6	3	14	17	4	20	16	13	18	10	7	11
14	10	4	8	9	12	3	11	17	20	19	6	15	5	2	18	16	7	1	13

Table B.2: Twenties one switch for decryption [8]

15	9	1	5	17	19	3	2	10	8	11	18	12	16	6	13	20	4	14	7
12	6	15	2	4	9	8	16	19	17	5	11	20	7	10	18	1	14	13	3
4	18	5	8	16	1	12	15	20	14	13	17	11	2	7	9	6	3	10	19
6	11	2	20	14	7	18	12	15	3	8	5	10	1	17	19	9	16	4	13
7	2	13	3	9	4	17	14	1	12	18	20	6	11	16	15	5	8	19	10
5	17	14	7	10	9	19	20	8	13	1	2	16	3	4	12	11	18	6	15
8	4	3	11	19	13	2	9	12	16	10	17	14	15	20	6	18	1	7	5
20	1	16	10	15	8	14	11	18	5	3	7	13	17	19	4	2	9	12	6
9	8	7	15	5	2	4	13	17	1	11	6	19	18	14	10	3	20	16	12
10	12	11	18	8	16	20	17	5	6	9	3	4	19	13	7	1	14	15	2
11	7	14	4	18	20	6	1	13	19	12	15	5	9	16	2	17	10	8	3
2	3	9	10	13	14	15	16	7	11	20	12	18	6	1	5	8	17	19	4
16	10	15	1	17	3	13	9	4	7	6	8	2	14	5	11	12	19	18	20
19	16	18	12	3	13	9	10	6	2	17	14	11	4	7	20	15	5	1	8
18	14	12	19	1	7	10	6	11	15	5	9	8	20	17	4	3	13	2	16
20	3	19	2	4	5	11	14	9	10	18	16	15	12	8	7	13	6	17	1
3	6	4	14	2	12	16	5	18	20	7	19	1	15	9	8	10	11	13	17
5	15	20	9	10	17	1	19	13	12	4	2	7	6	11	14	16	8	3	18
14	20	13	17	5	18	8	4	2	15	16	1	9	19	3	6	7	10	12	11
8	11	1	6	19	14	5	18	17	3	10	13	12	20	15	16	4	2	7	9
17	19	6	1	12	15	20	7	16	9	3	11	13	10	2	18	8	4	5	14
1	5	12	20	6	11	14	8	9	7	19	4	3	13	10	17	18	16	15	2
16	8	10	13	11	6	19	5	3	4	15	20	17	2	18	1	14	7	9	12
19	13	8	16	20	10	7	1	2	18	14	6	9	5	12	3	17	15	11	4
13	1	17	15	7	4	16	3	14	5	2	10	18	8	11	9	19	12	20	6

Table B.3: Twenties two switch for decryption [8]

7	19	11	3	20	1	10	6	16	12	17	13	8	9	4	18	5	14	15	2
15	17	14	2	12	13	8	3	1	19	9	4	10	7	11	20	16	6	18	5
2	11	20	12	1	19	4	10	9	14	6	15	13	3	7	16	18	8	5	17
16	3	12	9	4	20	6	19	18	2	5	8	14	11	10	1	15	17	13	7
12	18	16	4	9	3	15	13	6	20	8	2	7	10	5	19	14	1	17	11
13	9	5	6	8	7	12	17	14	18	20	10	2	19	11	15	4	3	1	16
4	7	2	15	17	10	19	5	8	16	1	12	3	13	6	14	20	9	11	18
9	6	4	10	18	16	8	14	5	12	17	1	20	15	13	19	2	11	7	3
5	14	18	17	13	15	11	12	7	8	3	6	1	2	20	4	9	10	16	19
11	16	9	18	3	12	5	15	10	1	14	17	2	4	19	6	8	7	13	20
19	8	3	15	14	5	1	11	2	10	12	16	18	20	17	7	13	4	9	6
1	12	17	13	9	7	14	2	15	4	5	11	6	16	3	8	18	19	20	10
3	4	10	12	1	18	2	8	14	13	19	7	16	6	15	9	17	20	5	11
9	11	6	5	10	4	17	19	13	15	7	2	12	18	14	20	1	16	8	3
8	13	14	16	19	12	20	7	10	3	15	9	4	17	1	11	5	2	6	18
18	16	15	4	2	17	13	12	6	11	20	19	14	5	9	1	8	7	3	10
14	1	7	20	6	13	16	18	12	9	4	17	5	11	2	3	10	15	19	8
17	19	1	11	7	2	18	4	3	8	10	5	15	12	16	9	6	13	20	14
10	15	2	14	11	6	7	1	16	20	13	3	9	8	18	17	19	5	12	4
20	9	8	6	12	11	2	5	4	7	16	14	17	3	15	10	13	19	18	1
11	20	13	8	16	10	18	14	19	6	15	4	1	17	7	5	3	9	2	12
16	5	10	19	4	18	15	17	1	3	2	20	11	6	8	13	7	12	14	9
6	10	19	16	5	9	1	20	17	4	11	18	7	14	13	2	12	8	3	15
8	7	5	1	15	14	9	16	11	17	18	6	19	20	3	12	4	2	10	13
13	2	17	7	14	8	3	9	20	5	16	10	6	1	12	15	11	18	4	19

Table B.4: Twenties three switch for decryption [8]

Appendix C

Permutations for PseudoPurple

C.1 Encryption Permutations

6	9	18	8	22	1	11	16	14	7	21	23	13	4	20	17	26	19	2	25	12	15	3	24	5	10
9	23	14	2	3	24	26	15	25	20	17	18	19	7	13	5	6	16	21	11	1	12	8	4	22	10
3	25	16	22	26	6	17	19	23	15	9	12	4	18	8	13	1	14	10	20	5	11	2	21	24	7
19	14	2	6	18	8	22	11	17	21	20	15	24	3	25	9	16	26	12	5	4	13	7	10	1	23
25	22	14	12	9	3	20	5	26	13	24	19	8	18	17	15	23	11	2	7	1	4	6	10	16	21
26	2	12	24	25	10	8	16	4	17	3	15	14	6	19	20	22	13	9	23	5	7	11	1	18	21
11	14	25	16	15	5	3	22	7	9	18	26	10	21	24	1	13	6	19	20	12	2	23	4	8	17
1	25	17	19	10	15	3	16	20	13	21	6	14	7	12	5	22	11	24	2	26	23	9	18	8	4
14	19	26	22	17	20	10	6	3	24	5	21	25	11	16	15	2	9	18	7	8	12	23	1	13	4
22	17	6	9	14	21	15	2	5	10	26	1	20	11	24	18	4	16	23	8	13	3	19	25	7	12
3	8	17	11	20	22	23	12	10	15	7	24	16	19	13	5	9	18	26	1	25	6	2	14	21	4
24	5	21	2	1	10	17	15	22	19	25	13	6	14	4	12	18	20	8	26	9	23	11	16	7	3
21	10	15	26	22	25	14	12	13	5	11	20	19	16	1	9	3	17	6	23	4	8	2	18	24	7
16	14	5	11	15	10	3	6	17	21	1	2	26	23	7	9	24	20	19	12	13	4	22	8	18	25
14	5	24	8	13	19	6	10	16	26	15	2	22	21	11	3	18	25	12	17	7	23	20	4	9	1
6	22	20	21	23	26	18	1	17	25	8	7	15	12	3	14	10	4	16	13	2	9	5	19	24	11
20	18	3	25	5	19	1	26	16	17	15	23	10	14	21	24	7	6	11	12	13	22	4	9	2	8
16	9	8	4	25	17	21	24	20	1	12	18	2	5	22	26	11	7	13	14	3	15	19	6	10	23
15	16	21	19	20	13	3	25	4	11	17	6	1	10	26	8	24	22	12	7	2	9	5	14	18	23
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
11	24	17	15	4	26	13	19	1	5	21	3	12	25	14	18	10	20	23	2	6	7	9	8	16	22
7	5	22	23	10	13	8	20	15	9	17	18	6	26	3	25	12	1	11	14	4	2	16	19	21	24
15	22	25	16	12	20	19	26	13	8	10	17	18	23	24	1	21	4	6	11	5	9	7	14	2	3
4	11	13	17	1	12	24	3	8	23	25	9	20	14	5	19	15	21	7	18	2	16	10	22	6	26
25	21	11	4	20	18	24	8	9	3	14	10	26	1	19	23	15	22	17	16	2	7	13	6	12	5

Table C.1: PseudoPurple encrypt switch

C.2 Decryption Permutations

6	19	23	14	25	1	10	4	2	26	7	21	13	9	22	8	16	3	18	15	11	5	12	24	20	17
21	4	5	24	16	17	14	23	1	26	20	22	15	3	8	18	11	12	13	10	19	25	2	6	9	7
17	23	1	13	21	6	26	15	11	19	22	12	16	18	10	3	7	14	8	20	24	4	9	25	2	5
25	3	14	21	20	4	23	6	16	24	8	19	22	2	12	17	9	5	1	11	10	7	26	13	15	18
21	19	6	22	8	23	20	13	5	24	18	4	10	3	16	25	15	14	12	7	26	2	17	11	1	9
24	2	11	9	21	14	22	7	19	6	23	3	18	13	12	8	10	25	15	16	26	17	20	4	5	1
16	22	7	24	6	18	9	25	10	13	1	21	17	2	5	4	26	11	19	20	14	8	23	15	3	12
1	20	7	26	16	12	14	25	23	5	18	15	10	13	6	8	3	24	4	9	11	17	22	19	2	21
24	17	9	26	11	8	20	21	18	7	14	22	25	1	16	15	5	19	2	6	12	4	23	10	13	3
12	8	22	17	9	3	25	20	4	10	14	26	21	5	7	18	2	16	23	13	6	1	19	15	24	11
20	23	1	26	16	22	11	2	17	9	4	8	15	24	10	13	3	18	14	5	25	6	7	12	21	19
5	4	26	15	2	13	25	19	21	6	23	16	12	14	8	24	7	17	10	18	3	9	22	1	11	20
15	23	17	21	10	19	26	22	16	2	11	8	9	7	3	14	18	24	13	12	1	5	20	25	6	4
11	12	7	22	3	8	15	24	16	6	4	20	21	2	5	1	9	25	19	18	10	23	14	17	26	13
26	12	16	24	2	7	21	4	25	8	15	19	5	1	11	9	20	17	6	23	14	13	22	3	18	10
8	21	15	18	23	1	12	11	22	17	26	14	20	16	13	19	9	7	24	3	4	2	5	25	10	6
7	25	3	23	5	18	17	26	24	13	19	20	21	14	11	9	10	2	6	1	15	22	12	16	4	8
10	13	21	4	14	24	18	3	2	25	17	11	19	20	22	1	6	12	23	9	7	15	26	8	5	16
13	21	7	9	23	12	20	16	22	14	10	19	6	24	1	2	11	25	4	5	3	18	26	17	8	15
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
9	20	12	5	10	21	22	24	23	17	1	13	7	15	4	25	3	16	8	18	11	26	19	2	14	6
18	22	15	21	2	13	1	7	10	5	19	17	6	20	9	23	11	12	24	8	25	3	4	26	16	14
16	25	26	18	21	19	23	10	22	11	20	5	9	24	1	4	12	13	7	6	17	2	14	15	3	8
5	21	8	1	15	25	19	9	12	23	2	6	3	14	17	22	4	20	16	13	18	24	10	7	11	26
14	21	10	4	26	24	22	8	9	12	3	25	23	11	17	20	19	6	15	5	2	18	16	7	1	13

Table C.2: PseudoPurple decrypt switch