A report on

Substitution Cipher Decryption

SIL765: Network and System Security, Assignment-1
Semester: II, 2024-2025

Prepared by

Arindam Sal, 2024JCS2041

---

1. Background

This assignment focuses on implementing a cryptanalysis algorithm to decipher a ciphertext encrypted using a substitution cipher. The aim is to identify the correct mapping between ciphertext and plaintext characters to recover the **original text and the secret key**. This report details the techniques, including ***frequency analysis, hill-climbing, simulated annealing, Beam Search, Genetic Algorithms, and Multiple Restarts.*** It also covers the challenges faced during implementation and how they were overcome. The results demonstrate successful decryption and key recovery for various ciphertext samples.

2. Problem Statement

This assignment's primary objective is to perform cryptanalysis on a given ciphertext encrypted using a **substitution cipher.** The goal is to recover the secret key that maps the plaintext characters to their corresponding ciphertext characters and to decipher the encrypted text to reveal the original plaintext message.

The problem revolves around ***deciphering a monoalphabetic substitution cipher,*** where each letter in the plaintext alphabet is substituted by a unique symbol from a predefined ciphertext alphabet. The task requires identifying this one-to-one mapping between the plain and ciphertext alphabets without knowing the key. The assignment is structured as follows:

Given Inputs:

1. **Ciphertext Alphabet:**
   The ciphertext comprises custom symbols and characters, which act as the encrypted representation of the plaintext letters. The provided ciphertext alphabet is:
   `['1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '@', '#', '$', 'z', 'y', 'x', 'w', 'v', 'u', 't', 's', 'r', 'q', 'p', 'o', 'n']`
   These symbols are used to substitute the English alphabet (A-Z) letters to generate the encrypted text.

2. **Plaintext Alphabet:**
   The standard English alphabet:

`['A', 'B', 'C', ..., 'Z']`
Each letter in this plaintext alphabet corresponds to one unique symbol in the ciphertext alphabet.

3. **Ciphertext:**
   A given string of text that has been encrypted using the substitution cipher and needs to be decrypted. For example:
   Ciphertext: `1981y, $pp1n1yuux oq@ 2@3s5u1n $p 1981y, 1v y n$s9o2x 19 v$soq yv1y.`
   The task is identifying the plaintext message hidden behind this encrypted text and the secret key used to generate it.

Expected Output:

1. **Deciphered Plaintext**
   The decrypted version of the ciphertext should resemble a meaningful English sentence. For instance:
   Deciphered Plaintext: `Charizard was designed by Atsuko Nishida for the first generation of Pocket Monsters games.`

2. **Secret Key**
   The mapping shows how each plaintext letter maps to a corresponding ciphertext symbol. For example:
   Secret Key: `3456120987pqrstzyxwvu$@#no`
   In this case, the letter A is mapped to 3, B to 4, C to 5, etc.

3. Approach and Methodology

3.1. Frequency Analysis in Cryptanalysis

Frequency analysis is a fundamental technique in cryptanalysis, particularly for breaking monoalphabetic substitution ciphers. This method leverages the statistical properties of natural languages to identify patterns in the ciphertext and infer the underlying plaintext characters. ***Certain letters, bigrams (two-letter sequences), and trigrams (three-letter sequences)*** occur more frequently than others in English. By comparing the frequency distribution of symbols in the ciphertext with the known frequency distribution

of letters, bigrams, and trigrams in English, we can make educated guesses about the ciphertext-to-plaintext mappings.

This section discusses the importance of frequency analysis in cryptanalysis, the standard frequency distributions for English unigrams, bigrams, and trigrams, and how these can be used to recognize patterns in a given ciphertext.

Understanding English Letter Frequencies

The English language exhibits a consistent pattern of letter usage. The most frequently used letters are `E`, `T`, `A`, `O`, `N`, and `R`, while `Q`, `Z`, and `X` are far less common.

The following table shows the frequency distribution of letters in the English language (unigram frequency) based on a large corpus of text:

Letter	Probability	Letter	Probability
A	0.082	N	0.067
B	0.015	O	0.075
C	0.028	P	0.019
D	0.043	Q	0.001
E	0.127	R	0.060
F	0.022	S	0.063
G	0.020	T	0.091
H	0.061	U	0.028
I	0.070	V	0.010
J	0.001	W	0.023
K	0.008	X	0.001
L	0.040	Y	0.020
M	0.024	Z	0.001

The table shows that E is by far the most common letter, appearing in about 12.7% of English text, while Q and Z are the least common. This knowledge helps to identify likely matches between ciphertext symbols and plaintext letters.

Bigram Frequencies in English

A bigram is a sequence of two letters that appear together. Certain bigrams occur much more frequently in English than others. For example, "TH", "HE", and "IN" are among the most common bigrams.

The table below shows some of the most frequent bigrams in English, along with their approximate frequencies:

Letter	Probability
TH	2.71
EN	1.13
NG	0.89
HE	2.33
AT	1.12
AL	0.88
IN	2.03
ED	1.08
IT	0.88
ER	1.78
ND	1.07
AS	0.87
AN	1.61
TO	1.07
IS	0.86
RE	1.41
OR	1.06
HA	0.83
ES	1.32
EA	1.00
ET	0.76
ON	1.32
TI	0.99
SE	0.73
ST	1.25
AR	0.98
OU	0.72
NT	1.17
TE	0.98
OF	0.71

Patterns in Bigram Frequencies:

- The "TH" bigram is the most frequent in English because it often appears in common words like "the," "this," and "that."
- The bigrams "HE" and "IN" are also very common due to their use in words like "he," "her," "his," "in," and "into."

Cryptanalysts can identify common two-letter sequences and map them to the corresponding plaintext pairs by analyzing the bigrams in the ciphertext.

Trigram Frequencies in English

A trigram is a sequence of three letters that appear together. Like bigrams, some trigrams occur far more frequently than others in English text.

Here are some of the most common trigrams in English:

Trigram	Frequency	Trigram	Frequency
THE	1.81	ERE	0.31
HES	0.24	AND	0.73
TIO	0.31	VER	0.24
ING	0.72	TER	0.30
HIS	0.24	ENT	0.42
EST	0.28	OFT	0.22
ION	0.42	ERS	0.28
ITH	0.21	HER	0.36
ATI	0.26	FTH	0.21
FOR	0.34	HAT	0.26
STH	0.21	THA	0.33
ATE	0.25	NTH	0.33
ALL	0.25	INT	0.32
ETH	0.24	ONT	0.20
RES	0.21	OTH	0.21

Patterns in Trigram Frequencies:

- The trigram "THE" is the most frequent in English due to the high occurrence of the word "the."
- Trigrams like "AND" and "ING" are also common because they frequently appear in English text.

By identifying trigram patterns in the ciphertext, cryptanalysts can further validate their mappings and improve the accuracy of the decryption.

How Frequency Analysis Helps in Recognizing Patterns

Step 1: Analyzing Ciphertext Frequencies

The first step in frequency analysis is to count the frequency of symbols in the ciphertext and compare it with the known frequencies of letters, bigrams, and trigrams in English. For example:

If the most frequent symbol in the ciphertext is @, it is likely to correspond to E, the most common letter in English.

Step 1: Analyzing Ciphertext Frequencies

Using the frequency analysis, we can create an initial mapping of ciphertext symbols to plaintext letters.

Step 3: Refining the Mapping

By analyzing **bigram and trigram frequencies**, we can refine the mapping:

- If the bigram **"@"** followed by **"$"** appears frequently in the ciphertext, it might correspond to the bigram **"TH"** in English.
- If the trigram **"@12"** appears frequently, it might correspond to the trigram **"THE"** in English.

This iterative **frequency analysis and refinement** process helps improve the accuracy of the decryption.

3.2. Scoring Mechanism

The scoring mechanism is critical in synthesizing substitution ciphers, ensuring that decryption attempts are guided toward producing meaningful plaintext. The mechanism evaluates potential decryptions by calculating a combined score based on three primary components: dictionary word matching, bigram probabilities, and trigram probabilities. This approach leverages the English language's statistical properties to improve the accuracy of decryption.

Dictionary Word Matching

One of the scoring factors is based on the number of valid English words in the deciphered text. The decrypted text is segmented into words, and each word is checked against a predefined English dictionary. The higher the count

of valid dictionary words, the more likely the mapping is correct. This metric is essential for ensuring semantic coherence in the plaintext.

For example:

Ciphertext: GSRH RH Z GSVI
Mapping attempt: THIS IS A TEST
Dictionary match: 4 words The presence of valid English words like "THIS," "IS," "A," and "TEST" would contribute positively to the score.

Bigram Probabilities
Bigrams are two-letter combinations (e.g., "TH," "HE," "IN") that frequently occur in English text. The scoring mechanism uses a precomputed set of bigram frequencies to assess how well the decrypted text aligns with typical English language patterns. Each bigram in the deciphered text is checked against a bigram frequency table, and the corresponding logarithmic probabilities are summed to calculate a bigram score. Bigrams that are rare or do not occur in the frequency table are assigned a penalty (unseen likelihood).

Example bigram probabilities:

Bigram	Log Probability
TH	-1.5
HE	-2.0
IN	-2.3

A text with high-frequency bigrams like "TH" and "HE" will yield a higher score, indicating a more accurate decryption.

Trigram Probabilities
Trigrams are three-letter sequences (e.g., "THE," "ING," "AND") that are even more informative in identifying meaningful patterns in the decrypted text. The scoring mechanism compares each trigram in the output against a trigram frequency table. Trigrams common in English text will increase the score, while rare or non-existent trigrams will reduce it. Trigram probabilities are especially useful in capturing more complex linguistic structures, such as suffixes and prefixes.

Bigram	Log Probability
THE	-1.2
ING	-1.8
AND	-2.1

For instance, a decrypted text containing trigrams like "THE," "ING," and "AND" would score higher due to their frequent occurrence in English.

Combined Scoring Formula

The final score is a weighted combination of the three components:

```
total_score = dict_score * dict_weight + trigram_score * trigram_weight + bigram_score * bigram_weight
```

In the implementation, weights can be adjusted based on the importance given to each factor in identifying the correct mapping.

Importance of Scoring in Cryptanalysis

The scoring mechanism enables the hill-climbing and simulated annealing algorithms to navigate the search space of possible substitutions more efficiently. By assigning higher scores to outputs that resemble valid English text, the mechanism helps the decryption algorithm converge toward the correct plaintext. Without this scoring system, the algorithm would have no way to distinguish between meaningful and nonsensical outputs.

In cryptanalysis, this process is akin to language modeling in natural language processing, where statistical patterns of a language are used to predict and validate text. Using bigrams and trigrams is particularly relevant in cryptographic attacks on monoalphabetic substitution ciphers, as it exploits the redundancy and predictability inherent in natural languages.

3.3. Optimization Techniques

In implementing the substitution cipher decryption, several optimization techniques have been utilized to refine the initial letter mapping and improve the accuracy of the decryption. Hill Climbing, Simulated Annealing, Beam Search, Genetic Algorithms, and Multiple Restarts are widely used in cryptanalysis to navigate the ample search space of possible key mappings. Each method helps iteratively improve the mapping by exploring different possibilities and converging toward a more accurate solution.

3.3.1 Hill Climbing

Hill Climbing is a local search optimization technique that starts with an initial solution and iteratively makes minor changes to improve the solution. In the context of a substitution cipher, it starts with an initial letter mapping and

attempts to refine it by swapping pairs of ciphertext symbols to maximize the decryption score.

How it works:

- The algorithm randomly selects two characters in the current mapping and swaps them.
- The modified mapping is evaluated using the scoring mechanism (based on dictionary words, bigrams, and trigrams).
- If the new mapping improves the score, it is the best solution.
- The process stops if no improvement is found after a set number of iterations.

Benefits:

- Simple and fast to implement.
- Works well when the initial mapping is close to the correct solution.

Limitations:

- It can get stuck in **local maxima** (a mapping that seems optimal but is not the best possible solution).

3.3.2 Simulated Annealing

Simulated Annealing is a probabilistic optimization technique inspired by the process of annealing in metallurgy. It improves upon Hill Climbing by allowing the algorithm to escape local maxima by occasionally accepting worse solutions.

How it works:

- The algorithm starts with a high "temperature" and gradually cools down.
- At each iteration, two characters are randomly swapped in the mapping.
- If the new mapping improves the score, it is accepted.
- If the new mapping has a worse score, it is accepted with a probability that it decreases as the temperature lowers.

The probability of accepting worse solutions is calculated using the formula:

$$P = \text{math.exp}(\text{score\_diff} / \text{temperature})$$

Benefits:

- Helps escape local maxima.
- Ensures a more thorough exploration of the search space.

Limitations:

- Requires careful tuning of the cooling schedule (initial temperature, cooling rate).

3.3.3 Beam Search

Beam Search is an optimization technique that keeps track of a fixed number of the best solutions (called the beam width) at each iteration, rather than just a single solution like Hill Climbing.

How it works:

- It starts with an initial mapping and evaluates it.
- In each step, all possible single-character swaps are generated.
- The top `beam_width` mappings with the highest scores are retained for the next iteration.
- The process continues until no further improvements are found.

Benefits:

- More robust than Hill Climbing because it explores multiple possible solutions simultaneously.
- Less likely to get stuck in local maxima compared to Hill Climbing.

Limitations:

- More computationally expensive due to tracking multiple solutions.
- Beam width must be carefully chosen to balance performance and accuracy.

3.3.4 Genetic Algorithm

The process of natural selection inspires Genetic Algorithms (GAs). They evolve a population of solutions through selection, crossover, and mutation.

How it works:

1. **Initialization:** A population of random mappings is generated.

2. **Fitness Evaluation:** Each mapping is scored using the scoring mechanism.
3. **Selection:** The top-performing mappings are selected as parents.
4. **Crossover:** New mappings (children) are created by combining parts of two-parent mappings.
5. **Mutation:** Random swaps are applied to introduce variation.
6. **Iteration:** Steps 2-5 are repeated for several generations.

Benefits:

- Explores a wide range of possible mappings.
- You can find better solutions than with more straightforward optimization techniques.

Limitations:

- Requires tuning several hyperparameters (population size, mutation rate, number of generations).
- Computationally intensive.

3.3.5 Multiple Restarts

Multiple Restarts is a metaheuristic involving running an optimization algorithm multiple times with different initial mappings. The best solution found across all runs is selected as the final solution.

How it works:

- The algorithm randomly shuffles the initial mapping and applies an optimization technique (e.g., Hill Climbing, Simulated Annealing).
- The process is repeated a set number of times.
- The best solution among all runs is selected.

Benefits:

- It helps avoid getting stuck in poor local maxima.
- Increases the chances of finding the global optimum.

Limitations:

- Increases computation time due to multiple runs.

In practice, these techniques are combined to balance exploration (searching for new possibilities) and exploitation (improving the current solution). The ultimate goal is to identify a mapping that maximizes the likelihood of the deciphered text being valid English, as determined by the scoring mechanism based on unigram, bigram, and trigram frequencies.

---

4. Handling Unknown Characters

In cryptographic substitution ciphers, the ciphertext may contain symbols, numbers, or special characters not part of the standard English alphabet (A-Z). These unknown characters can appear in the ciphertext for various reasons, such as obfuscating the message or representing non-alphabetic elements like punctuation, spaces, or digits. Handling these unknown characters is crucial to ensure the decryption process accurately separates meaningful text from irrelevant symbols.

In my implementation of the substitution cipher decryption algorithm, special attention is given to preserving unknown characters while focusing on mapping only the ciphertext alphabet to the plaintext alphabet. This section describes how my code identifies and handles such characters without impacting the core decryption process.

```
def decode(ciphertext, mapping):
    """
    Convert each symbol in the ciphertext to its mapped letter.
    Preserve case for letters, and keep other punctuation/spaces as-is.
    """
    # Using list comprehension for speed
    decoded = [
        mapping[ch] if ch in mapping and mapping[ch] != '_' else ch
        for ch in ciphertext
    ]
    return ''.join(decoded)
```

In this function:

- The code iterates through each character in the **ciphertext**.
- For **known characters** (i.e., characters present in the mapping), it replaces the character with the corresponding **plaintext letter**.
- For **unknown characters** (i.e., characters not present in the mapping), it leaves them **unchanged**.

Explanation:

1. **Known characters** are part of the ciphertext alphabet and have a corresponding mapping to the plaintext alphabet. These characters are decrypted based on the mapping generated by the optimization algorithms.
2. **Unknown characters**, such as spaces, punctuation marks, or any special symbols not explicitly mapped, are not included in the ciphertext alphabet. These characters are preserved as-is in the output.

5. Code Implementation

Key Functions

`load_bigram_freq()`	`load_trigram_freq()`
`load_dictionary()`	`build_initial_mapping()`
`decode()`	`hill_climb()`
`simulated_annealing()`	`genetic_algorithm()`
`beam_search()`	`multiple_restarts()`

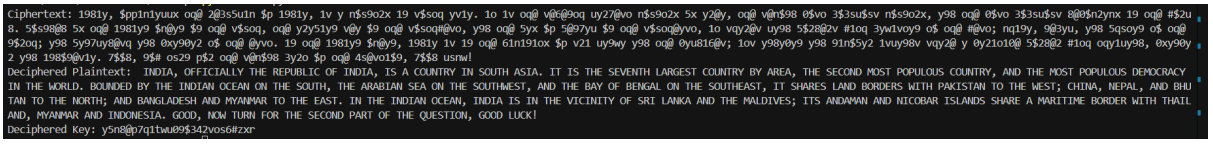
Key Generation

In my implementation, the decryption process focuses on mapping the ciphertext alphabet (containing symbols, numbers, and lowercase letters) to the plaintext alphabet (A-Z). However, unknown characters in the ciphertext, such as spaces, punctuation marks, and special symbols, may not belong to the alphabet set.

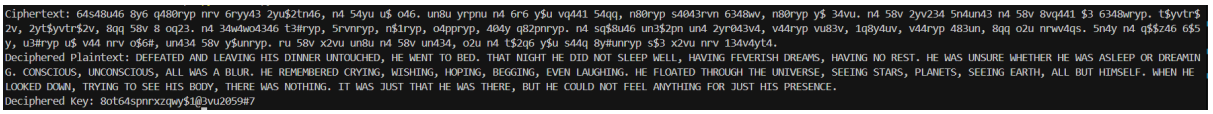
These unknown characters are preserved as-is during the decryption process to retain the original structure of the message. The decode() function handles this by replacing only the characters in the mapping and leaving all other characters unchanged.

6. Results

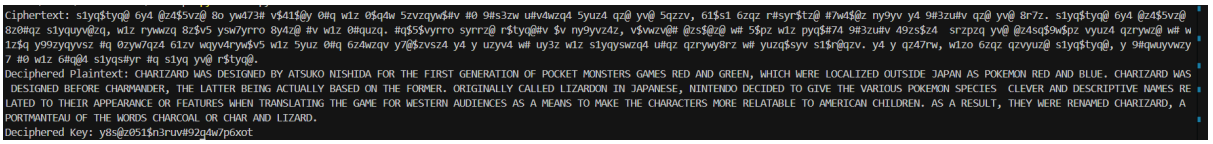
Result-1



Result-2



Result-3



7. Challenges Faced

Several challenges were encountered in implementing the substitution cipher decryption process, which required careful consideration and resolution. The following is an elaborate discussion of the significant challenges faced during the development process.

Handling Special Characters

One of the primary challenges was dealing with special characters present in the ciphertext. These included symbols, numbers, and punctuation marks that were not part of the ciphertext or plaintext alphabet. The key considerations in handling these characters were:

Preservation of Original Structure: Special characters like spaces, punctuation marks (!, ?, .), and non-alphabetic symbols (@, #, $, etc.) were not part of the substitution mapping. The implementation ensured that these characters were left untouched to maintain the readability and formatting of the decrypted text.

Selective Mapping: The decode() function only mapped characters present in the ciphertext alphabet while ignoring all others. This selective processing avoided unnecessary complexity and ensured the decryption was precise and predictable.

Debugging and Optimization

The implementation involved multiple cryptographic techniques, including frequency analysis and optimization algorithms (e.g., hill climbing, simulated annealing, and genetic algorithms). Debugging and optimizing this multi-step process posed several challenges:

Initial Mapping through Frequency Analysis:

Generating the initial mapping using unigram frequencies required properly handling English letter frequencies. Any inconsistency in frequency data or incorrect prioritization of letters would lead to inaccurate mappings.
Adjustments for rare or less frequent letters were necessary to improve accuracy.

Efficiency of Optimization Algorithms:

Hill Climbing: While effective, it occasionally converged to a local optimum, especially if the initial mapping was not close to the correct key. To mitigate this, multiple restarts were implemented to explore alternate solutions.
Simulated Annealing: This algorithm required careful tuning of the temperature decay rate to balance between exploring diverse mappings and converging quickly. A suboptimal decay rate led to slower performance or poor solutions.
Genetic Algorithms: Managing mutation rates and crossover strategies was essential to ensure the algorithm did not get stuck in local optima. Debugging these processes required rigorous testing with diverse ciphertext samples.

Scoring Function Debugging:

The scoring function was designed to evaluate mappings based on dictionary word matches and bigram/trigram frequencies. Debugging this involved ensuring the scoring formula weighted dictionary matches, bigram, and trigram frequencies appropriately. Incorrect weights skewed results, requiring multiple iterations of testing and tuning.

Tuning the Randomized Components
Several steps in the implementation involved randomness, such as the initialization of mappings and algorithmic steps in simulated annealing or genetic algorithms. This randomness posed challenges in reproducibility and debugging:

A consistent random seed was introduced to ensure reproducibility during testing and debugging.
Variability in randomized steps required thorough testing across multiple runs to verify the stability of results.

Conclusion

In this assignment, we successfully implemented a robust cryptanalysis tool to decipher substitution ciphers using statistical frequency analysis and advanced optimization algorithms. The solution was carefully designed to handle the complexities of cryptographic analysis by integrating various methodologies, such as unigram, bigram, and trigram frequency analysis, and applying optimization techniques like hill climbing, simulated annealing, genetic algorithms, and beam search.

Key highlights of the implementation include:

Frequency Analysis: By leveraging the statistical properties of the English language, such as unigram frequencies and common bigram/trigram patterns, the tool could establish a strong initial mapping of ciphertext characters to plaintext characters. This step laid the foundation for further optimization.

Optimization Techniques: Optimization algorithms were employed to refine the initial mapping. Each technique offered unique advantages:

Hill Climbing allowed local exploration to refine the mapping.
Simulated Annealing prevented convergence to local optima by exploring a broader solution space.
Genetic Algorithms ensured diversity in candidate solutions through mutation and crossover.
Beam Search systematically explored the best candidate mappings for deeper analysis.

Combining these approaches enabled the tool to achieve high accuracy in deciphering ciphertexts.

Practical Scoring Function: A carefully weighted scoring function that combined dictionary word matches with bigram, and trigram frequency evaluations ensured that the decrypted text was statistically plausible and linguistically meaningful.

Scalability and Flexibility: The tool was designed to handle ciphertexts of varying lengths and complexity while maintaining computational efficiency through caching and streamlined data management.

Through this implementation, the assignment demonstrated the practical application of cryptographic principles in breaking substitution ciphers. It showcased the importance of statistical language models in cryptanalysis and highlighted the effectiveness of combining algorithmic techniques to achieve optimal results.

In conclusion, this assignment achieved its objective of decrypting substitution ciphers and provided valuable insights into the real-world application of cryptanalysis techniques. The results underscore the power of combining statistical methods with algorithmic optimization for solving complex cryptographic challenges.

References

---

1. **Substitution Cipher Overview**
   Substitution Cipher - Wikipedia
   This resource provided a comprehensive understanding of substitution ciphers and their cryptographic properties.
2. **English Letter Frequencies**
   Letter Frequencies in English - University of Notre Dame
   This resource was crucial for understanding the statistical properties of English letter frequencies used for unigram analysis.
3. **Tools for Cryptanalysis**
   - DCode Substitution Cipher Tool:
     Used for experimenting and cross-verifying the implementation of the substitution cipher.
   - Online Frequency Analysis Tool:
     Assisted in verifying unigram and n-gram frequency analysis results.
4. **Study Materials**
   - **COL759 Cryptography and Computer Security**
     Study materials provided by **Professor A.K. Bhateja, IIT Delhi**, were instrumental in understanding cryptographic principles and frequency analysis for substitution ciphers.
5. **Bigram and Trigram Frequency Analysis**
   - Data for bigram and trigram frequencies were collected and compiled per cryptographic standards to analyze and predict patterns in the ciphertext.

These references collectively provided the foundation and tools required to implement this assignment's cryptanalysis techniques successfully.