

アルファベット文字の登場頻度について

- ・よく使われる文字・組み合わせの頻度を測り，スコア化
- ・よく出る組み合わせが多い文＝スコアが高い文になるように暗号文を解読する
- ・暗号化する人は，文字の頻度が一定となり解読されないようにする必要がある
- ・1文字あたりの頻度
- ・5文字あたりの頻度

Monogram Frequencies

English single letter frequencies are as follows (in percent %):

A : 8.55	K : 0.81	U : 2.68
B : 1.60	L : 4.21	V : 1.06
C : 3.16	M : 2.53	W : 1.83
D : 3.87	N : 7.17	X : 0.19
E : 12.10	O : 7.47	Y : 1.72
F : 2.18	P : 2.07	Z : 0.11
G : 2.09	Q : 0.10	
H : 4.96	R : 6.33	
I : 7.33	S : 6.73	
J : 0.22	T : 8.94	

Quintgram Frequencies

These are 5-gram frequencies for English. We can't list all of them here, the top 30 are the following (in percent %):

OF THE : 0.18	AND TH : 0.07	CTION : 0.05
ATION : 0.17	ND THE : 0.07	WHICH : 0.05
IN THE : 0.16	ON THE : 0.07	THESE : 0.05
THERE : 0.09	ED THE : 0.06	AFTER : 0.05
ING TH : 0.09	THEIR : 0.06	EOFT H : 0.05
TOT HE : 0.08	TION A : 0.06	ABOUT : 0.04
NG THE : 0.08	ORT HE : 0.06	ERT HE : 0.04
OTHER : 0.07	FORTH : 0.06	IONAL : 0.04
AT THE : 0.07	ING TO : 0.06	FIRST : 0.04
TIONS : 0.07	THE CO : 0.05	WOULD : 0.04

<http://practicalcryptography.com/cryptanalysis/letter-frequencies-various-languages/english-letter-frequencies/>

暗号化の仕組み（英語の例）①

置換表の作成

- 英語の母音等，頻度が高い文字を多めに暗号文字を割り当てる．
（但し最低 1 英字1数字とする）
- 英字以外の文字種（数字文字，改行文字，スペースを含む）は削除する．ただし大段落は維持

```
def generate_random_frequencies(max_value):  
    # 一般的な、英字の頻度に基づく重み付けを下記に定義する  
    weights = {  
        'A': 8.55, 'B': 1.60, 'C': 3.16, 'D': 3.87, 'E': 12.10, 'F': 2.18, 'G': 2.09, 'H': 4.96,  
        'I': 7.33, 'J': 0.22, 'K': 0.81, 'L': 4.21, 'M': 2.53, 'N': 7.17, 'O': 7.47, 'P': 2.07,  
        'Q': 0.10, 'R': 6.33, 'S': 6.73, 'T': 8.94, 'U': 2.68, 'V': 1.06, 'W': 1.83, 'X': 0.19,  
        'Y': 1.72, 'Z': 0.11  
    }  
  
    # 処理1. まず全文字に最低「1個」ずつ割り当てる  
    # max_valueとは、置換表で使用する数値の最大値を指す  
    counts = {char: 1 for char in weights.keys()}  
  
    # 処理2. 全体の置換文字総数 (0からmax_valueまでなので +1)  
    total_slots = max_value + 1  
  
    # 処理3. 残りの残数を計算  
    remaining_slots = total_slots - len(counts)  
  
    # 処理4. 重みに基づいて、残りの枠をランダムに分配  
    # chars: 文字リスト, probs: 重みリスト, weightsは重みづけ配列  
    chars = list(weights.keys())  
    probs = list(weights.values())  
  
    if remaining_slots > 0: # 残り枠がある場合処理  
        # (重みづけ付き) ランダムに選んで割り当て  
        extra_allocations = random.choices(chars, weights=probs, k=remaining_slots)  
        for char in extra_allocations:  
            counts[char] += 1  
  
    return counts #countsの形は配列, {'A': 5, 'B': 2, ...}
```

暗号化の仕組み（英語の例）②

置換表の作成

- 例はアルファベット26文字を50文字と100文字に置換した例である。（平文は同一）
- 右の画像は以後の解読時に用いた暗号文である
- 今回は数字だが必要な種類がある
キャラクタなら何でもよい
（片仮名なら50種など）
- 大文字小文字は区別しないものとする。

```
A : 079,041,047,035,081,026,010,059,054,082,055,086↵  
B : 069,089,063↵  
C : 068,044↵  
D : 028,058,050,029↵  
E : 087,001,095,016,039,094,020,092,060,048↵  
F : 075,067,065,031↵  
G : 078,042,037,014↵  
H : 088,085,093,062,084,100,049,032↵  
I : 064,003,030,036,076,034,043↵  
J : 072,052↵  
K : 066,008↵  
L : 019,025,056,011↵  
M : 005,040,038↵  
N : 022,097,000,007↵  
O : 045,051,070,009↵  
P : 015,096,083↵  
Q : 024↵  
R : 090,023,033,013,061↵  
S : 012,098,006,080↵  
T : 091,057,046,071↵  
U : 018,021,099↵  
V : 027↵  
W : 004,002↵  
X : 053,077↵  
Y : 074,073↵  
Z : 017[EOF]
```

```
A : 041,048,030↵  
B : 008,000↵  
C : 050,049↵  
D : 014↵  
E : 032,020,002,026↵  
F : 036,037↵  
G : 004↵  
H : 016,043↵  
I : 044,038,047,035↵  
J : 015↵  
K : 018↵  
L : 046↵  
M : 009↵  
N : 028,031,011,013↵  
O : 022,042,003,034↵  
P : 007↵  
Q : 005↵  
R : 039,021↵  
S : 045,024↵  
T : 025,001,006,010↵  
U : 017↵  
V : 012,033↵  
W : 023↵  
X : 019↵  
Y : 040,029↵  
Z : 027[EOF]
```

暗号化の仕組み（英語の例） ③

暗号文の作成

- 置換表を複数個作成し、それをさらにランダムに選択し、平文を1文字ずつ変換する.

```
def generate_homophonic_table(max_value):  
  
    # 処理1: 各文字に何個割り当てるかを決定  
    frequencies = generate_random_frequencies(max_value)  
    # 0からmax_valueまでの数値リストを作成  
    available_numbers = list(range(max_value + 1))  
    # ランダム性を担保するためにシャッフル  
    random.shuffle(available_numbers)  
    cipher_table = {}  
    # 処理2: 決定した個数分だけ数値を割り当てる  
    for char, count in frequencies.items():  
        homophones = []  
        for _ in range(count):  
            if available_numbers:  
                num = available_numbers.pop()  
                # 3桁埋め (000 ~ 200) で統一  
                homophones.append(f"{num:03d}")  
  
        # 安全装置: 万が一空ならランダム値 (通常ここは通りません)  
        # 例えばmax_valueが50の場合, 51個割り当てようとするところに来る  
        if not homophones:  
            homophones.append(f"{random.randint(0, max_value):03d}")  
  
        cipher_table[char] = homophones  
  
    return cipher_table
```

暗号化の仕組み（英語の例）④

置換前の平文

(ゲティスバーグ演説, 264単語1452文字)

Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we can not dedicate—we can not consecrate—we can not hallow—this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us—that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion—that we here highly resolve that these dead shall not have died in vain—that this nation, under God, shall have a new birth of freedom—and that government of the people, by the people, for the people, shall not perish from the earth.

置換後の暗号文



ゲティスバーグ平文utf8.txt



3FF9B27A.txt

```
036.022.017.039 045.049.042.039.026 030.013.014 045.020.035.032.011 029.020.048.039.045
041.004.034 042.017.021 037.030.025.016.020.021 045 000.039.022.017.004.043.028
037.003.039.001.043 022.013 023.043.038.045 030.034.028.010.047.013.026.031.001. 041
013.032.023 028.048.001.044.042.011 049.022.031.030.026.035.012.032.014 044.013
046.035.008.002.021.025.029. 048.031.014 014.020.014.044 049.048.006.020.014 001.003
025.043.032 007.021.042.007.022.024 038.010.035.022.013 001.043.048.025 041.046.046
009.032.028 030.039.032 049.039.032 030.006.020.014 026.005.017.030.046.
011.022.023 023.032 030.021.020 032.031.004.030.004.002.014 047.011 030
004.039.026.048.010 030.035.012.047.046 023.041.021. 006.026.024.025.044.028.004
023.016.002.010.016.002.039 025.016.041.006 013.030.010.044.022.031. 034.021.041.028.040
028.048.010.044.034.028 024.034 050.042.011.030.026.044.033.002.014 041.013.014 045.003
014.032.014 038.030.041.010.026.014. 049.041.013 046.022.031.004
002.031.014.017.039.025. 023.020 048.039.028 009.032.001 022.013 030
004.039.002 030.025 006.048.006.010.046.020.~036.044.042.046.014 042.036 010.016.048.010
023.030.021. 023.020 043.048.033 029 049.003.009.020 001.003
014.020.014 038.030.048.025.020 048 007.003.021.001.047.034.028 003.036 006.043.030.001
036.047.032.046.014. 041.045 048 037.044.028.041.046 021.032.045.006.044.011.004
007.046.030.050.020 037.034.021 025.016.034.045.002 023.016.022 043.020.039.020
004.041.012.020 006.043.002.035.039 046.047.012.026.024 025.016.030.006 006.043.048.001
031.041.006.038.003.011 009.044.004.043.006 046.038.012.026. 038.010 044.024
048.046.025.003.004.032.010.043.032 039 037.047.006.010.035.011.004 030.028.014
007.021.003.007.002.039 010.016.030.001 023.002 045.043.003.017.046.014 014.003
001.016.035.024. 008.017.001. 047.011 048 046.041.021.004.032 021.045.020.011.045.020.
023.020 050.048.031 011.034.010 014.032.014.044.049.030.010.020.~023.032 030.030.013
011.003.006 049.003.011.045.032 050.039.048.006.026.~023.020 049.030.011 011.034.010
043.046.046.046.042.023.~010.043.035.045 004.021.003.017.011.014. 010.043.032
008.039.030.033.020 009.032.013. 046.047.033.044.028.004 048.028.014 014.020.048.014
023.016.042 024.006.039.017.004.004.046.002.014 016.026.021.028. 016.030.033.028
049.042.013.024.002.030.039.048.001 032.014 044.001 037.041.021.041.000.022.012.026
003.017.039 007.003.003.039 007.003.023.020.021 010.034 030.014.014 042.039
014.032.006.021.043.010. 006.016.020 023.034.021.046.014 023.044.046.046
046.044.025.006.046.020 013.034.006.028. 011.003.021 046.022 028.004
039.020.009.020.009.000.026.039 023.043.041.006 023.020 045.041.029 043.032.039.002.
000.017.010 047.006 030.041.011 028.002.012.002 039 037.034.021.004.002 025
023.016.030.006 001.043.032 029 014.044.014 016.002 021.032. 038.023 035.045
037.003.039 017.045 001.043.026 046.038.012.044.013.004. 039.048.010.016.020.021.
025.034 008.020 014.002.014.044.049.048.001.032.014 043.002.039.032 025.022 025.016.032
017.020.036.035.013.035.045.016.026.014 023.042 039.018 023.043.044 049.016
025.043.026.040 023.043.022 036.042 017.004.043.006 043.026.039.026 043.030.012.032
025.016.017.043 037.048.021 024.042 013.022.000.046.040
030.014.012.030.031.049.002.014. 044.025 044.045 021.048.006.016.020.021 036.034.021
017.024 025.034 006.020 043.020.021.002 014.020.014.044.049.048.002.014 006.022
006.043.026 004.021.020.048.001 001.041.024.018 021.032 009.041.035.013.035.011.004
000.026.037.042.021.026 017.024.~025.016.030.010 036.021 022.009 025.016.026.045.026
016.034.028.042.021.032.014 014.002.048.014 023.026 010.030.018.032
038.023.050.039.002.041.045.020.014 014.026.033.003.001.035.003.028 025.034
025.016.048.025 049.048.017.024.020 037.003.039 023.043.044.030.043 010.043.020.040
004.030.033 020 006.043.002 046.041.024.001 036.017.046.046 009.032.030.024.017.021.032
042.037 014.026.033.003.010.035.022 022.~025.016.041.001 023.026 016.032.021.020
043.044.004.043.046.040 039.032 024.022.046.033.026 001.016.048.010 025.043.032.024.032
014.032.041.014 024.016.041.046.046 022.003.025 043.041.039.026 014.047.020.014 044.013
012.030.035.011.~001.043.041.001 010.016.038.024 031.048.001 035.034.013.
017.031.014.026.021 004.034.014. 024.016.048.046.046 016.048.012.002 048 028.028 023
008.044.039.010.043 003.036 036.021.032.032.014.042 009.~030.031.014 006.043.041.023
004.042.033.002.021.031.009.032.013.010 003.036 001.016.026 007.026.003.007.046.002.
000.040 006.016.032 007.020.042 007.046.002. 036.003.039 010.016.026
007.002.034.007.046.020. 024.043.048.046 046 011.034.010 007.002.039.044.045.016
037.021.034 009 001.016.032 032 048.039.010.016.~
```

出典：渡部純，ゲティスバーグ演説の訳し方，明治学院大学，2013年1月

置換表を用いた復号化プログラム①

- 置換表が入手できる場合、暗号文に現れるキャラクター（今回の場合は0~50の数字）を、表に当てはめて1語1語置換し解読していく。
- 暗号文には大文字小文字等の情報は含まれていない為、このプログラムではすべて大文字としている。
- 第一引数は暗号文、第二引数は置換表とする。

```
1 def load_decryption_table(filename):  
2     table = {}  
3     with open(filename, 'r', encoding='utf-8') as f:  
4         for line in f:  
5             if ':' in line:  
6                 char, nums_part = line.split(':', 1)  
7                 for num in nums_part.strip().split(','):   
8                     if num.strip():  
9                         table[num.strip()] = char.strip()  
10    return table  
11  
12 def decrypt_text(ciphertext, table):  
13     result, i = [], 0  
14     while i < len(ciphertext):  
15         chunk = ciphertext[i:i+3]  
16         if len(chunk) == 3 and chunk.isdigit() and chunk in table:  
17             result.append(table[chunk])  
18             i += 4 if i+3 < len(ciphertext) and ciphertext[i+3] == ',' else 3  
19         else:  
20             result.append(ciphertext[i])  
21             i += 1  
22     return "".join(result)  
23  
24 def main():  
25     import sys  
26     table = load_decryption_table(sys.argv[2])  
27     with open(sys.argv[1], 'r', encoding='utf-8') as f:  
28         ciphertext = f.read()  
29     with open('decrypted.txt', 'w', encoding='utf-8') as f:  
30         f.write(decrypt_text(ciphertext, table))  
31  
32 if __name__ == "__main__":  
33     main()
```

置換表を用いた復号化プログラム②

暗号前

Four score and seven years ago our fathers brought forth on this continent, a new nation, conceived in Liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battle-field of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we can not dedicate—we can not consecrate—we can not hallow—this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us—that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion—that we here highly resolve that these dead shall not have died in vain—that this nation, under God, shall have a new birth of freedom—and that government of the people, by the people, for the people, shall not perish from the earth.



置換表

A : 041,048,030↵
B : 008,000↵
C : 050,049↵
D : 014↵
E : 032,020,002,026↵
F : 036,037↵
G : 004↵
H : 016,043↵
I : 044,038,047,035↵
J : 015↵
K : 018↵
L : 046↵
M : 009↵
N : 028,031,011,013↵
O : 022,042,003,034↵
P : 007↵
Q : 005↵
R : 039,021↵
S : 045,024↵
T : 025,001,006,010↵
U : 017↵
V : 012,033↵
W : 023↵
X : 019↵
Y : 040,029↵
Z : 027[EOF]



復号結果

FOUR SCORE AND SEVEN YEARS AGO OUR FATHERS BROUGHT FORTH ON THIS CONTINENT A NEW NATION CONCEIVED IN LIBERTY AND DEDICATED TO THE PROPOSITION THAT ALL MEN ARE CREATED EQUAL.

NOW WE ARE ENGAGED IN A GREAT CIVIL WAR TESTING WHETHER THAT NATION OR ANY NATION SO CONCEIVED AND SO DEDICATED CAN LONG ENDURE. WE ARE MET ON A GREAT BATTLE-FIELD OF THAT WAR. WE HAVE COME TO DEDICATE A PORTION OF THAT FIELD AS A FINAL RESTING PLACE FOR THOSE WHO HERE GAVE THEIR LIVES THAT THAT NATION MIGHT LIVE. IT IS ALTOGETHER FITTING AND PROPER THAT WE SHOULD DO THIS. BUT IN A LARGER SENSE WE CAN NOT DEDICATE—WE CAN NOT CONSECRATE—WE CAN NOT HALLOW—THIS GROUND. THE BRAVE MEN LIVING AND DEAD WHO STRUGGLED HERE HAVE CONSECRATED IT FAR ABOVE OUR POOR POWER TO ADD OR DETRACT. THE WORLD WILL LITTLE NOTE NOR LONG REMEMBER WHAT WE SAY HERE BUT IT CAN NEVER FORGET WHAT THEY DID HERE. IT IS FOR US THE LIVING RATHER TO BE DEDICATED HERE TO THE UNFINISHED WORK WHICH THEY WHO FOUGHT HERE HAVE THUS FAR SO NOBLY ADVANCED. IT IS RATHER FOR US TO BE HERE DEDICATED TO THE GREAT TASK REMAINING BEFORE US—THAT FROM THESE HONORED DEAD WE TAKE INCREASED DEVOTION TO THAT CAUSE FOR WHICH THEY GAVE THE LAST FULL MEASURE OF DEVOTION—THAT WE HERE HIGHLY RESOLVE THAT THESE DEAD SHALL NOT HAVE DIED IN VAIN—THAT THIS NATION UNDER GOD SHALL HAVE A NEW BIRTH OF FREEDOM—AND THAT GOVERNMENT OF THE PEOPLE BY THE PEOPLE FOR THE PEOPLE SHALL NOT PERISH FROM THE EARTH.



89B485CA.txt

アルファベット文字の登場頻度について（再掲）

- ・よく使われる文字・組み合わせの頻度を測り，スコア化
- ・よく出る組み合わせが多い文＝スコアが高い文になるように暗号文を解読する
- ・暗号化する人は，文字の頻度が一定となり解読されないようにする必要がある
- ・1文字あたりの頻度
- ・5文字あたりの頻度

Monogram Frequencies

English single letter frequencies are as follows (in percent %):

A : 8.55	K : 0.81	U : 2.68
B : 1.60	L : 4.21	V : 1.06
C : 3.16	M : 2.53	W : 1.83
D : 3.87	N : 7.17	X : 0.19
E : 12.10	O : 7.47	Y : 1.72
F : 2.18	P : 2.07	Z : 0.11
G : 2.09	Q : 0.10	
H : 4.96	R : 6.33	
I : 7.33	S : 6.73	
J : 0.22	T : 8.94	

Quintgram Frequencies

These are 5-gram frequencies for English. We can't list all of them here, the top 30 are the following (in percent %):

OF THE : 0.18	AND TH : 0.07	CTION : 0.05
ATION : 0.17	ND THE : 0.07	WHICH : 0.05
INT HE : 0.16	ON THE : 0.07	THESE : 0.05
THERE : 0.09	ED THE : 0.06	AFTER : 0.05
ING TH : 0.09	THEIR : 0.06	EOFT H : 0.05
TOT HE : 0.08	TIONA : 0.06	ABOUT : 0.04
NG THE : 0.08	ORT HE : 0.06	ERT HE : 0.04
OTHER : 0.07	FORTH : 0.06	IONAL : 0.04
AT THE : 0.07	ING TO : 0.06	FIRST : 0.04
TIONS : 0.07	THE CO : 0.05	WOULD : 0.04

解読プログラム①

頻度解析について

- 今回作成したプログラムは、暗号文が事前に同音字暗号であることを前提とし解読を行う。

↓

- そのため、予め暗号文が同音字暗号か否かは前述の方法を用いて判定を行う。
- 一定の暗号文字（今回は5文字）ごとに頻度解析を行う。これを1つずつ右へずらしながら1サイクルごとに、スコアを加算する。
- 暗号文字の区切りはコンマまたはスペースである。（今回作成したプログラムはスペースとカンマを区別しておらず、単語の境目は検知していない）

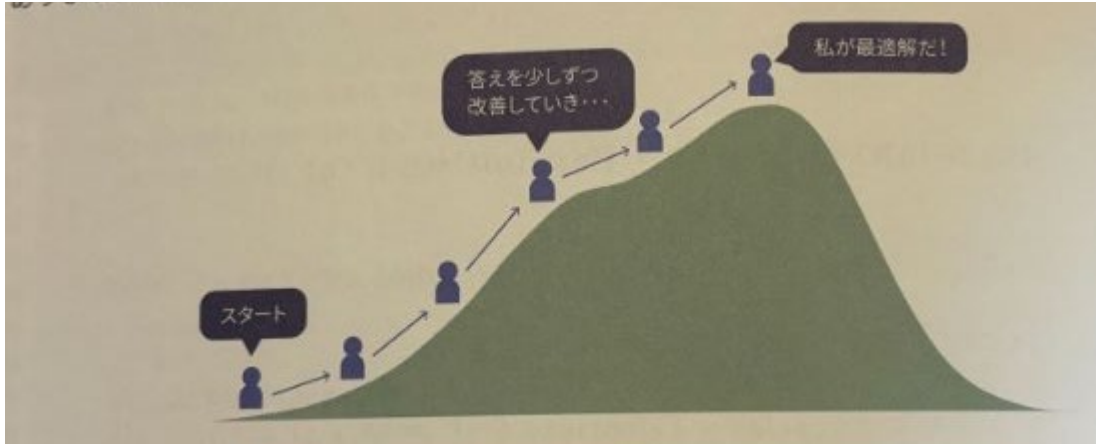
036,022,017,039 045,049,042,039,0
011,022,023 023,032 030,021,020 0

この5文字区切りで
頻度解析を行う。

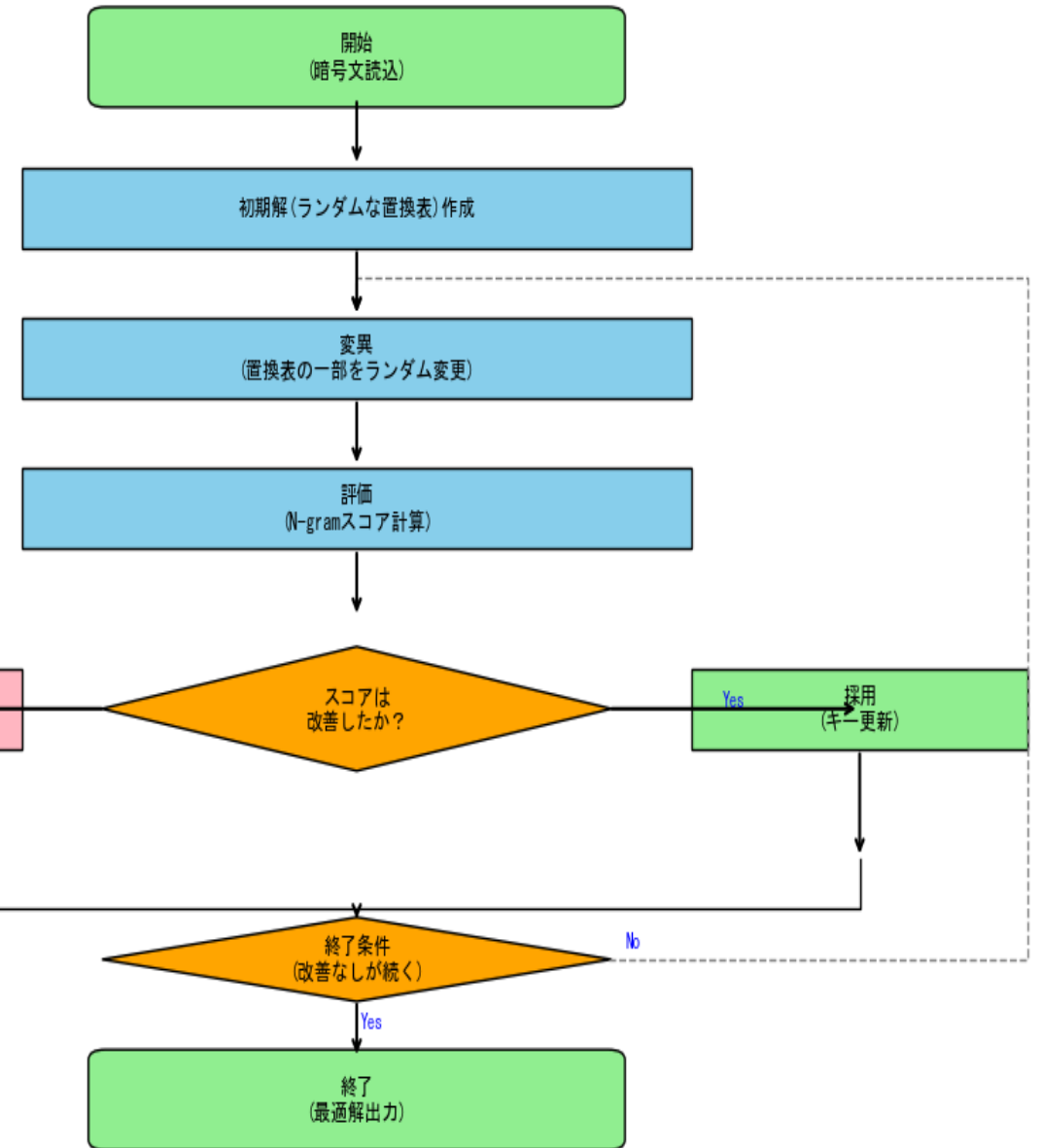
TERNA	517195↓
UNIVE	516838↓
STING	516356↓
ECOMM	513939↓
ROGRA	513494↓
IGHTS	513161↓
CLUDE	512404↓
SIONS	512017↓
ICALL	510843↓
SWILL	510610↓
THETI	510461↓
ERSAN	510109↓

解読プログラム②

ヒルクライム法について



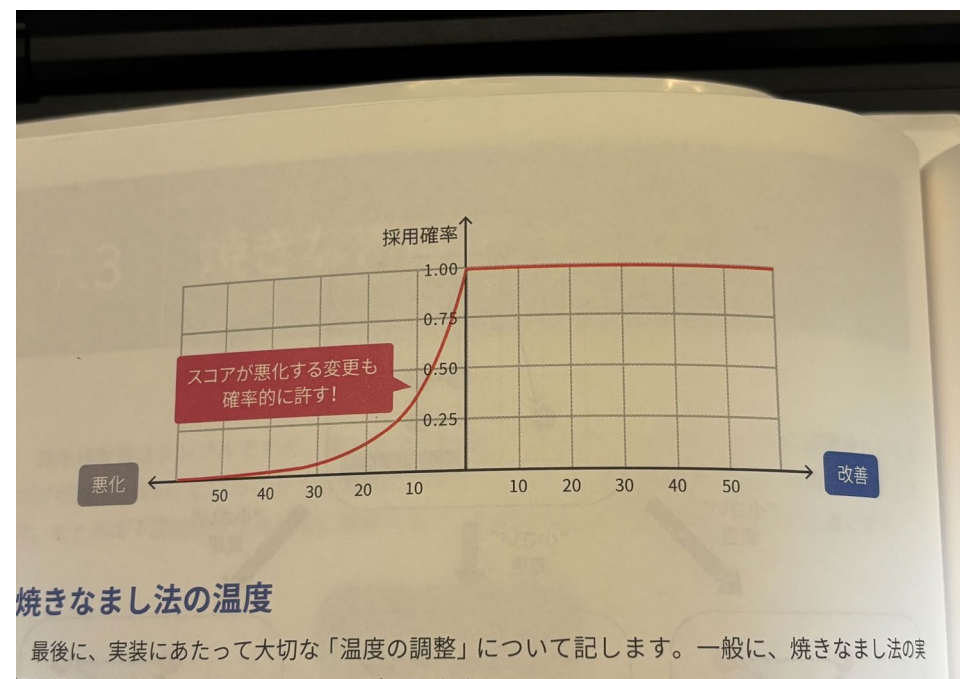
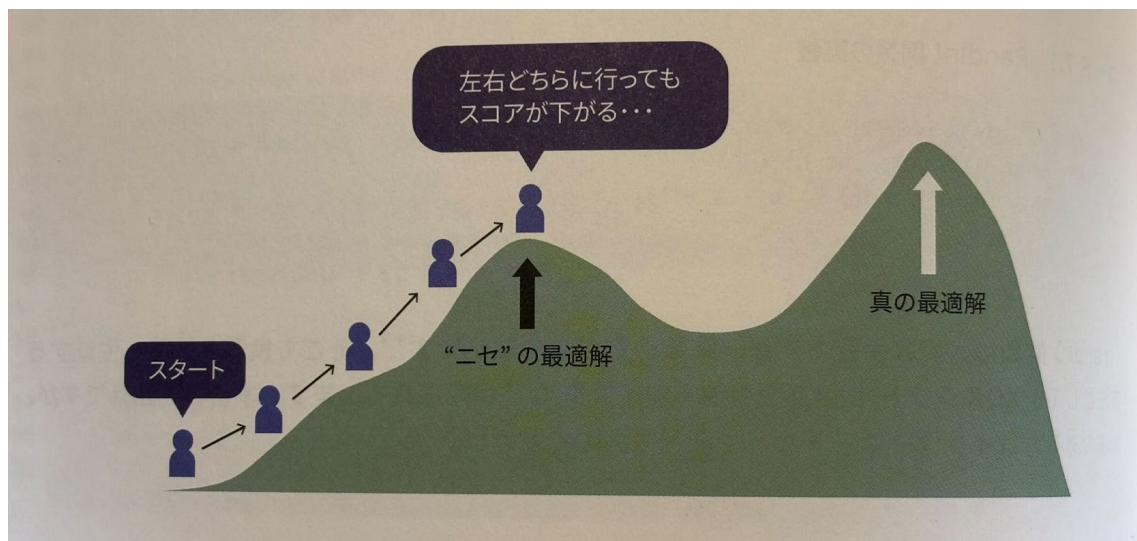
焼きなまし法では、スコアが改善しなくても、一定確率で採用し、局所解に陥ることを防ぐ



解読プログラム③

焼きなまし法における解読プログラムについて

- ヒルクライム法を発展させ局所解に陥りにくい焼きなまし法を採用
- 文の途中での改行に対処（段落を削除）し，段落ごとに頻度解析
- マルチスレッドによる数十回の試行を行い最適解を求める



解読プログラム④

- 文章スコア
- $S = \sum_{i=0}^{L-N} \log_{10}(P(w_i))$
- S : 算出されるスコア
ここに数式を入力します。(対数をとっているため負値)
- L : 復号されたテキストの全長
- N : 計算対象($N - gram$)の長さ
(今回のプログラムでは5)
- w_i : テキストの i 番目から始まる N 文字の単語列
($N - gram$)
- $P(w_i)$: 計算対象の 5 文字($N - gram$)
が英語統計データ内で出現する確率

```
int main() {  
    // 入力  
    cin >> N;  
    for (int i = 1; i <= N; i++) cin >> X[i] >> Y[i];  
  
    // 初期解生成  
    P[1] = 1; P[N + 1] = 1;  
    for (int i = 2; i <= N; i++) P[i] = i;  
  
    // 焼きなまし法 (GetScore 関数、RandInt 関数は 7.2 節を参照)  
    double CurrentScore = GetScore();  
    for (int t = 1; t <= 200000; t++) {  
        int L = RandInt(2, N);  
        int R = RandInt(2, N);  
        if (L > R) swap(L, R);  
        reverse(P + L, P + R + 1);  
        double NewScore = GetScore();  
  
        // 7.2 節の解答例から変更した唯一の部分 (Probability は採用確率)  
        double T = 30.00 - 28.00 * t / 200000.0;  
        double Probability = exp(min(0.0, (CurrentScore - NewScore) / T));  
        if (Randouble() < Probability) CurrentScore = NewScore;  
        else reverse(P + L, P + R + 1);  
    }  
  
    // 出力  
    for (int i = 1; i <= N + 1; i++) cout << P[i] << endl;  
    return 0;  
}
```

実行画面



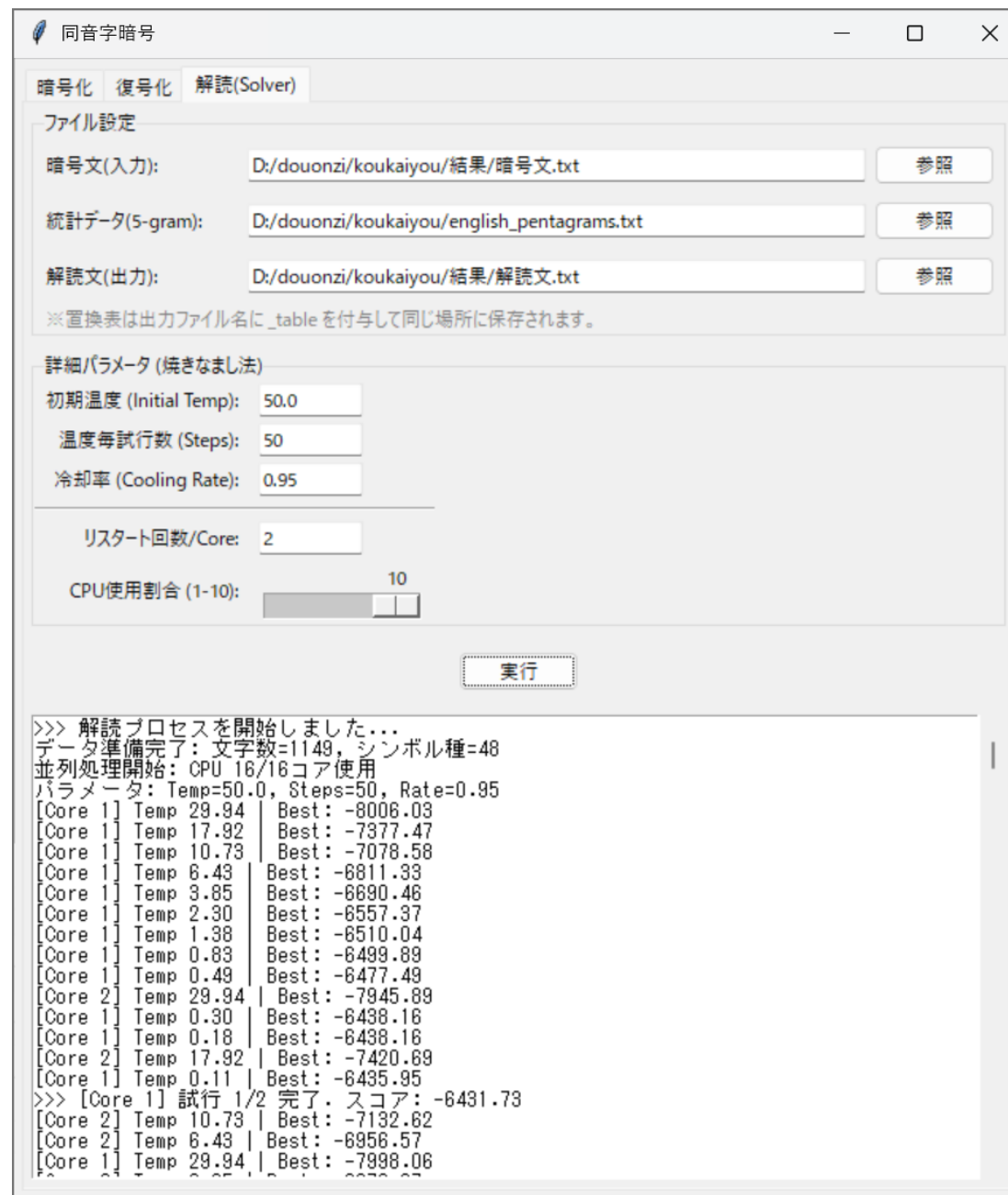
ans.txt



同音字暗号.exe



同音字暗号プログラム.zip



解読結果（置換表）

JやZなどは
登場頻度が少なく
精度が悪い



焼き直し法の結果.zip

単語数	264
文字数	1452
Temp	50
Step per Temp	5000
C_Rate	0.98
試行回数	32
処理時間	27分

ヒルクライム法

A : 030, 041, 048↵
B : 000↵
C : 049, 050↵
D : 005, 014↵
E : 002, 020, 026, 032, 040↵
F : 036, 037↵
G : 004↵
H : 016, 043↵
I : 035, 038, 044, 047↵
L : 046↵
M : 009↵
N : 011, 013, 028, 031↵
O : 003, 022, 034, 042↵
P : 007↵
R : 021, 039↵
S : 024, 029, 045↵
T : 001, 006, 008, 010, 018, 023, 025↵
U : 017↵
V : 012, 033↵

正答

A : 041, 048, 030↵
B : 008, 000↵
C : 050, 049↵
D : 014↵
E : 032, 020, 002, 026↵
F : 036, 037↵
G : 004↵
H : 016, 043↵
I : 044, 038, 047, 035↵
J : 015↵
K : 018↵
L : 046↵
M : 009↵
N : 028, 031, 011, 013↵
O : 022, 042, 003, 034↵
P : 007↵
Q : 005↵
R : 039, 021↵
S : 045, 024↵
T : 025, 001, 006, 010↵
U : 017↵
V : 012, 033↵
W : 023↵
X : 019↵
Y : 040, 029↵
Z : 027[EOF]

●焼きなまし法

A : 030, 041, 048↵
B : 000, 008↵
C : 049, 050↵
D : 014↵
E : 002, 020, 026, 032↵
F : 036, 037↵
G : 004↵
H : 016, 043↵
I : 035, 038, 044, 047↵
L : 046↵
M : 009↵
N : 011, 013, 028, 031↵
O : 003, 022, 034, 042↵
P : 007↵
Q : 005↵
R : 021, 039↵
S : 024, 045↵
T : 001, 006, 010, 018, 025↵
U : 017↵
V : 012, 033↵
W : 023↵
Y : 029, 040↵
[EOF]