Abstract:

This paper reviews the cryptological concept known as the Index of Coincidence (IC) which was introduced in 1922 by now famous cryptographer William Friedman. The index of coincidence is a statistical evaluation of the occurrence of characters in cipher text and assists cryptanalysts in identifying monoalphabetic or polyalphabetic ciphers and estimating the key length of the latter. The processes for calculating the IC, examples of this calculation, and different use cases for it will be discussed. Additionally, programs written in Python will be used to calculate frequencies, the index of coincidence, and the key length (if applicable) to demonstrate a more automated approach to cracking cipher texts that are simple relative to modern ones.

Section I: **Introduction**

The index of coincidence is relevant in the field of cryptography in that it deals directly with identifying key characteristics of certain types of cryptography. IC is very useful in quickly identifying monoalphabetic or polyalphabetic substitution ciphers and finding the key length of the latter. I chose this because I wanted to get a strong grasp of some of the simpler ciphers and the mathematics that are associated with making them work. I aim to break down the reason it works, the statistical and mathematical methods used in the calculation, and some examples of breaking cipher texts using programs.

The book "*Cracking Codes with Python: An Introduction to Building and Breaking Ciphers"* is an extensive overview of several different ciphers, the way they work, and how to break them. This book aims to teach the reader not only how to code in Python and use it to break ciphers, but it also teaches the reader how the ciphers work by allowing them to fully understand the way they work. Every function of every program in this book is thoroughly explained.

The paper "*Cryptanalysis of the Vigenère Cipher: The Friedman Test*" [2] is a report written by a student at Northern Kentucky University which describes the IC, its uses, and examples. This paper was used as a reference to learn how the index of coincidence works in a more condensed fashion than the Friedman paper.

"*The Index of Coincidence*" [3] is a revision of the Friedman paper described earlier which contains much of the same information.

"*Index of Coincidence*" [4] is a primarily mathematical and statistical description of the IC on the Michigan Tech school website.

The paper written by William Friedman, "*The Index of Coincidence and Its Applications in Cryptanalysis*" [5] is a technical paper that thoroughly describes the IC and ways that it can be used. This is the original paper introducing the Index of Coincidence and has since been released to the public.

Section II: The Index of Coincidence

In this section, I will discuss the uses of Friedman's index of coincidence. The focus of the examples used will be on cracking the Vigenère cipher because it is a fairly complex polyalphabetic substitution cipher that is broken very easily by finding the index of coincidence and the length of the key that was used during encryption. However, the index of coincidence can be used to quickly determine the type of cipher used to encrypt the plain text (i.e., monoalphabetic, transposition, or polyalphabetic substitution) and, if it is a polyalphabetic substitution cipher, the length of the encryption key.

The index of coincidence is a statistical frequency-based test. The value of IC can be anywhere between approximately 0.038 (like truly randomized text) and 0.0665 (like plain text). If the IC is closer to 0.0665, then the plain text was likely encrypted using a monoalphabetic substitution cipher such as a Caesar cipher, Affine cipher, or a keyword cipher; or a transposition cipher such as the rail fence cipher [2]. If the value of IC is near 0.038, then the cipher used was most likely a polyalphabetic substitution cipher such as the Vigenère cipher.

To provide a more elaborate description of what these numbers mean, let me do so by introducing more numbers! In English, if you multiply 0.0665 by 26 (for the 26 characters in the English alphabet), you will get a value somewhere around 1.73. This is known as a *normalized IC* for English. This value is different for every language, but it becomes useful when comparing the IC of different variations of the cipher text (which I will refer to as columns later on). Now, the normalized IC for a monoalphabetic cipher is 0.038 * 26 which comes out to ~1. Why is this important? If you have a text where every letter has the

same probability of appearance than another, then the IC is 1/N, where N is the number of characters in the alphabet that the text is written in. Therefore, dividing 1/26 = ~0.038 and is truly random for English. The index of coincidence will usually be *closer* and not *equal* to 0.038.

The index of coincidence is a summation of the appearances of the letters in the cipher text. Friedman found that if you choose two cipher text letters at random, the probability that both letters will be the same is higher if the letters are drawn from the same alphabet as opposed to different alphabets [2]. The equation for the index of coincidence is [4]:

$$IC = c \times \left(\frac{n_a}{N} \times \frac{n_a - 1}{N - 1}\right) + \left(\frac{n_b}{N} \times \frac{n_b - 1}{N - 1}\right) + \cdots + \left(\frac{n_z}{N} \times \frac{n_z - 1}{N - 1}\right)$$

$$= \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{\frac{N(N-1)}{c}}$$

where c is the normalizing coefficient for the language used (26 in English), n_a is the number of times the letter 'a' appears in the cipher text, and N is the length of the cipher text. In the summation, i refers to the numerical representation of the letter in the alphabet (a = 1, b = 2, …, z = 26).

The frequencies of the letters in the English language are [2]:

Letter	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
Frequency	.082	.015	.028	.043	.127	.022	.020	.061	.070
Letter	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**
Frequency	.002	.008	.040	.024	.067	.075	.019	.001	.060
Letter	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	
Frequency	.063	.091	.028	.010	.023	.001	.020	.001	

Therefore, the probability of choosing two letters that are the same is:

aa or bb or … or zz

$$.082 \times .082 + .015 \times .015 + \ldots + .001 \times .001 \approx 0.0665$$

This explains why an IC of close to 0.0665 would signal a monoalphabetic substitution or transposition cipher since monoalphabetic substitution and transposition ciphers are simply one-for-one substitutions or scrambling the order of the letters. The probabilities of choosing a pair of matching letters in English text would very closely reflect these numbers, and since we are preserving the relative frequencies of these letters, then the probabilities would not be changed except for the probabilities for individual letters.

If the cipher text were encrypted with multiple alphabets, the frequencies of the letters would be roughly equal. For the sake of simplicity, let us assume that the frequencies of cipher text letters are uniform. Then, the probability of choosing two letters that are the same would be [2]:

$$I \approx \left(\frac{1}{26} \times \frac{1}{26}\right) + \left(\frac{1}{26} \times \frac{1}{26}\right) + \cdots + \left(\frac{1}{26} \times \frac{1}{26}\right) = \frac{1}{26} \approx 0.038$$

The frequencies of each of the letters in the resulting text of a polyalphabetic substitution cipher would be more uniform than in monoalphabetic substitution or transposition ciphers.

When examining cipher texts, it is often useful to calculate the frequencies of each letter and plot them on a bar graph. If the graph has peaks and valleys, this is a suggestion that the cipher used may be monoalphabetic substitution or transposition; again, this is because the frequencies are preserved even though the letters are replaced by one unique letter (as in monoalphabetic substitutions) or simply rearranged (as in transposition ciphers). If the graph more closely resembles a uniform distribution, then we know that we are likely dealing with a polyalphabetic substitution cipher.

Let us examine the frequency distribution for the plain text below:

```
MUST CHANGE MEETING LOCATION FROM
BRIDGE TO UNDERPASS SINCE ENEMY
AGENTS ARE BELIEVED TO HAVE BEEN
ASSIGNED TO WATCH BRIDGE STOP MEETING
TIME UNCHANGED XX
```

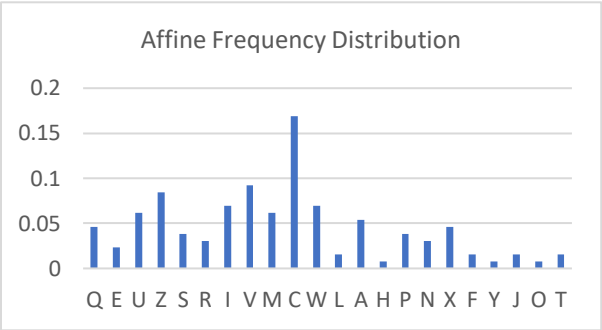
The XX is used as padding so the length of the cipher text cannot simply be divided to help determine anything useful to anyone attempting to crack the cipher. Below is the cipher text after encrypting the plain text with the Vigenère cipher and key = `EVERY`:

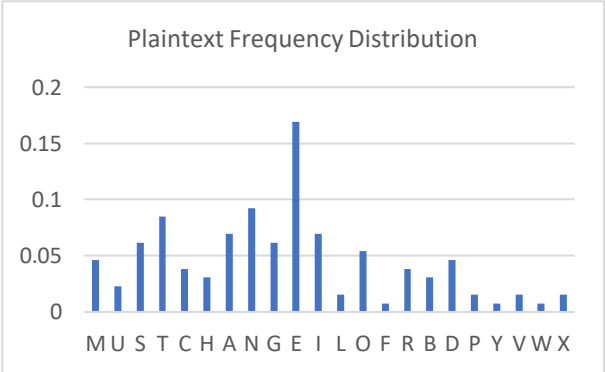
```
QPWKALVRXCQZIKGRBPFAEOMFLJMSDZVDHXC
XJYEBIMTRQWNMEAIZRVKCVKVLXNEICFZPZC
ZZHKMLVZVZIZRRQWDKECHOSNYXXLSPMYKVQ
XJTDCIOMEEXDQVSRXLRLKZHOV
```

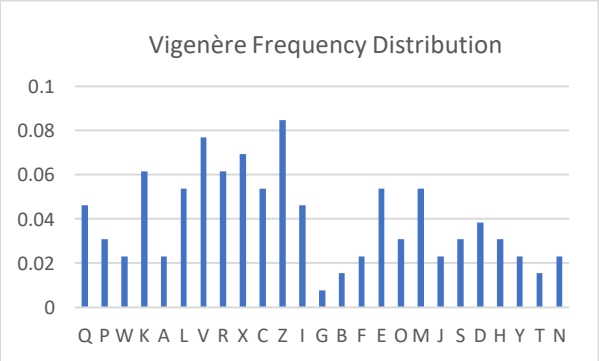
Below is the cipher text after encrypting the plain text with the Affine cipher and key a = 5 and b = 8:

```
QEUZSRIVMCQCCZWVMLASIZWAVHPAQNPWXMC
ZAEVXCPFIUUUWVSCCVCQYIMCVZUIPCNCLWC
JCXZARIJCNCCVIUUWMVCXZAOIZSRNPWXMCU
ZAFQCCZWVMZWQCEVSRIVMCXTT
```

Now take a look at the distribution of the letter frequencies:







The cipher text is limited in size which is the reason that the Vigenère cipher is less normally distributed than you may have expected. However, the important thing to notice here is that the frequency distribution for the Affine cipher text is identical to the plain text frequency distribution. This should be of no surprise since it was explained earlier that the frequency distributions would be preserved in a monoalphabetic substitution or transposition cipher. One can easily see that the cipher text letter 'C' translates to the English letter 'E'. Finding relationships like this can make determining the plain text from the cipher text much easier when examining simple monoalphabetic substitution and transposition ciphers.

So far, we have examined the simplicity of cracking monoalphabetic substitution and transposition ciphers and now it is time to discuss how the index of coincidence can help crack the Vigenère cipher. I wrote a program in Python (that will accompany this paper in the submission) that finds the index of coincidence for the entire cipher text. If the index of coincidence tells us that we are dealing with a polyalphabetic substitution cipher, then there is another part of the program that will break the cipher text up into the aforementioned *columns* of size *n*. For example, if you were to guess a column size (key length) of 7, then you would group the cipher into groups of 7 characters as shown below:

Column Size 7

```
QPWKALV
RXCQZIK
GRBPFAE
OMFLJMS
DZVDHXC
XJYEBIM
TRQWNME
AIZRVKC
VKVLXNE
ICFZPZC
ZZHKMLV
ZVZIZRR
QWDKECH
OSNYXXL
SPMYKVQ
XJTDCIO
MEEXDQV
SRXLRLK
ZHOV
```

Column Size 8

```
QPWKALVR
XCQZIKGR
BPFAEOMF
LJMSDZVD
HXCXJYEB
IMTRQWNM
EAIZRVKC
VKVLXNEI
CFZPZCZZ
HKMLVZVZ
IZRRQWDK
ECHOSNYX
XLSPMYKV
QXJTDCIO
MEEXDQVS
RXLRLKZH
OV
```

Now, what happens in the program is that we will calculate the index of coincidence for each column, sum up the ICs and divide the sum by the number of columns. For clarity, I have highlighted the first column of the size 7 arrangement. This makes the first string '`QRGODXTAVIZZQOSXMSZ`'. We find the IC of this string, the IC of the string of column #2 '`PXRMZ…`', column #3 '`WCBFV…`', and so on. If you guessed the column size incorrectly (as you likely will), then the IC will be somewhere around 1.00. We do this for columns of size 1 (column size 1 is just the original cipher text) through size 10 and print all

results to the screen. The results of this cipher text are below:

Average IC for Column Size 1: 0.04293 (1.116)
Average IC for Column Size 2: 0.04567 (1.188)
Average IC for Column Size 3: 0.04030 (1.048)
Average IC for Column Size 4: 0.04494 (1.168)
Average IC for Column Size 5: 0.07015 (1.824)
Average IC for Column Size 6: 0.03810 (0.990)
Average IC for Column Size 7: 0.03872 (1.007)
Average IC for Column Size 8: 0.04032 (1.048)
Average IC for Column Size 9: 0.04518 (1.175)
Average IC for Column Size 10: 0.07949 (2.067)

Now that you know a little more about the index of coincidence, think about which key (column) size is the correct one for this cipher text. If you said column size 5 or 10, you would be correct. The reason for two possible answers being correct is because the correct key size will reflect a high average IC for itself as well as its multiples. Hence, then correct column sizes will be 5 and 10, since the key is `EVERY` and in the Vigenère cipher, the key is repeated (i.e., `EVERYEVERY`). Notice how all the other column sizes are all closer to 1.0 and column sizes 5 and 10 are close to 2.0. This is how you find the key length for a polyalphabetic substitution cipher.

You may now be wondering why the correct key size shows a high value instead of a low value since, as it was explained earlier, the IC closer to 1.73 means that the text was likely encrypted with a polyalphabetic substitution cipher. However, if we split the cipher text up into blocks that are of equal size to the key, then every row will be encrypted with the same subkey, right? Think about it: if we split this cipher text into five columns, then every cipher text letter in column 1 will be encrypted with the letter 'E', all letters in column 2 are encrypted with the letter 'V', column 3 'E', column 4 'R', and column 5 'Y'. Therefore, we are dealing with each column of cipher text as a monoalphabetic substitution, specifically a Caesar shift! From here, all that needs to be done is perform frequency analysis and make educated guesses for the plain text equivalents of each cipher text character. That is exactly why we take the average of the IC for all the rows and we call it the aggregate delta IC. See the example below for clarity:


```
QPWKA
LVRXC
QZIKG
RBPFA
EOMFL
JMSDZ
VDHXC
XJYEB
IMTRQ
WNMEA
IZRVK
CVKVL
XNEIC
FZPZC
ZZHKM
LVZVZ
IZRRQ
WDKEC
HOSNY
XXLSP
MYKVQ
XJTDC
IOMEE
XDQVS
RXLRL
KZHOV
```

The index of coincidence was created in a time when computer encryption was not the primary focus of this method. Instead, Friedman was dealing with actual characters in the alphabets of different languages. Today's modern cryptography encrypts data at the binary level. We can still perform the index of coincidence at the binary level so long as we know the cipher text is a binary representation of alphabetic characters. In this case, the index of coincidence would focus on the frequency of appearance for different bit patterns that are in the ASCII table, for example. Here is how that would work – we need to first consider the bitwise representations of each upper and lowercase letter as the same. See the example below:

ASCII A = 01000001
ASCII a = 01100001
ASCII B = 01000010
ASCII b = 01100010
.
.
.

So, the count would need to be done like this: (A = 01000001 OR 01100001), (B = 01000010 OR 01100010), and so on. The way you could code this into a program is to add 00100000 if the 6th LSB is not 1; or, in pseudo-Python,

```
if bits[i=a] < 01100000:
      bits += 00100000
```

After we get each ASCII character's non-case sensitive count, then we can perform the index of coincidence. This will work for any known alphabet so long as you know the bitwise representation for each character of that alphabet and the length of that alphabet.

Section III: Alt. Method – Kasiski Examination.

The index of coincidence can be used with other methods of finding the key length as well, such as a method known as the Kasiski Examination. The first step in this method is to find every repeated set of at least three characters in the cipher text. These repeated characters could be the same plain text letters that were encrypted with the same subkeys of the Vigenère key. Take, for example, the plain text `THE CAT IS OUT OF THE BAG` with the key `SPILLTHEBEANS`, you would get [1]:

THECATISOUTOF**THE**BAG

SPILLTHEBEANS**SPI**LLT

LWMNLMPWPYTBX**LWM**MLZ

As you can see, the letters `LWM` are repeated twice in the cipher text. This is because the plain text word `THE` is encrypted with the same part of the key `SPI` in two parts of the cipher. The number of letters from the beginning of the first `LWM` to the beginning of its second appearance is 13, and we will call this the *spacing*. The spacing is similar to how the column sizes with the largest IC worked earlier in this paper in that it is the likely the size of the key.

Unfortunately (or fortunately, depending on your role in the encryption/decryption game) key will not always align perfectly with repeated sequences of letters, or the key could repeat several times between the repeated sequences which means that the number of letters between the repeated characters would be a multiple of the key length. To reiterate that more simply – it may not be this easy and the person attempting to crack it may have to do a little more evaluating to get it right. With longer cipher texts it is more likely that the spacing will more accurately reflect the key length. Take this example cipher from "Cracking Codes with Python: An Introduction to Building and Breaking Ciphers"

```
PPQCAXQVEKGYBNKMAZUYBNGBALJONITSZMJ
YIMVRAGVOHTVRAUCTKSGDDWUOXITLAZUVAV
VRAZCVKBQPIWPOU
```

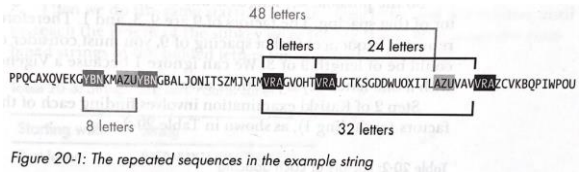


Figure 20-1: The repeated sequences in the example string

As depicted in the above image, the are 8 letters between the `YBN` combinations, 48 letters between the `AZU` combinations, and a total of 32 letters between the first and last `VRA` combinations (8 between the first and second and 24 between the second and third). What this means is that we have several possible lengths for our key. Next, all the factors of these counts need to be calculated and these will narrow down the potential key lengths to make life easier. The factors break down to:

Spacing	Factors
8	2, 4, 8
24	2, 4, 6, 8, 12, 24
32	2, 4, 8, 16
48	2, 4, 6, 8, 12, 24, 48

Collectively after factorization, we are left with four 2s, four 4s, two 6s, four 8s, two 12s, one 16, two 24s and one 48. Since 2, 4, and 8 are the most frequently occurring factors, they are likely the lengths of the Vigenère cipher key.

Perform the same columnar type of setup as explained earlier but this time only set them up in columns of size 2, 4, and 8. (Note: you could perform the index of coincidence here, but the point of this section is to review an alternative method.) Use a key length of 4 and see if you can crack the cipher text. If this doesn't work, then try again assuming the key length is 2 or 8. This works for the same reason as explained previously; if you choose the correct key length, then each letter of each column will have been encrypted using the same subkey, effectively transforming each column into a more simple transposition cipher.

Using a method alternative to the index of coincidence, you can run frequency analysis on each of these columns and the correct subkey will more closely reflect the frequency distribution associated with the English language. The book "Cracking Codes with Python: An Introduction to Building and Breaking Ciphers" [1] includes a Python program that will calculate this for us, but since this paper is focusing on the index of coincidence, it will not be discussed in full detail. However, there is one last part worth mentioning. The program will find the most likely subkeys for each column and then brute force through the possible keys using those subkeys until it finds a match.

Section IV: Conclusion

Encryption has and always will change throughout time so long as the methods to break them keep improving. The simple Caesar shift cipher was once thought to be a secure method of secretly sending a message, until the simplicity behind it was discovered and rendered it useless. Polyalphabetic substitution ciphers such as the one that was the focal point of this paper, the Vigenère cipher, came later on and proved to be much more difficult to figure out and crack by hand.

William Friedman did not invent the index of coincidence to crack the Vigenère cipher specifically, but this is a good cipher to use as an example because it's a fairly complex cipher that truly demonstrates the power of frequency analysis and the index of coincidence.

Section V: References

[1] A. Sweigart, *Cracking Codes with Python: An Introduction to Building and Breaking Ciphers*. William Pollock, 2018.

[2] C. Christensen, "*Cryptanalysis of the Vigenère Cipher: The Friedman Test*," Northern Kentucky University, Spring 2015.

[3] H. H. Campaigne, "*The Index of Coincidence*," Office of Research and Development, Jan. 1955.

[4] "*Index of Coincidence.*" [Online]. Available: https://pages.mtu.edu/~shene/NSF-4/Tutorial/VIG/Vig-IOC.html. [Accessed: 27-Feb-2020].

[5] W. F. Friedman, *The Index of Coincidence and Its Applications in Cryptanalysis*. Laguna Hills, Calif: Aegean Park Press, 1987.