The Big Assignment

CSE6339-Winter 2014

3/26/2014

Albina Rahim
Student # 213199989

1. Introduction

The **infinite monkey theorem** states that a monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely type a given text, such as the complete works of William Shakespeare [1]. Based on this concept, we developed a series of computer programs & routines which simulate a monkey typing random keys and generating an output of text. The goal was to explore properties and possible use of character frequency of a natural language text.

This report is divided into seven sections with Section 1 as the Introductory part, Section 2 giving detailed descriptions of the series of functions and programs written to accomplish this assignment, Section 3 discusses the Algorithms of the functions and programs implemented, Section 4 highlights and analyse the results generated, Section 5 illustrates some sample results, Section 6 highlights some future work that can be done to further improve this assignment, and Section 7 contains the web implementation of this assignment.

2. Functions & Programs Descriptions

This section illustrates descriptions of the series of functions and programs developed to achieve the goal of this assignment.

All our programs and functions were coded in **Matlab**. The name of our main program from which all the sub-programs and functions can be executed is known as the "**Big_Assignment.m**". Functions were written such that each can be reused to generate results for different problems. The programs were run until the monkeys have typed 100,000 characters, after which meaningful words were searched for from the scribbles. Only the unique meaningful words were counted to calculate the percentage word yield.

We downloaded the corpus "dictionary.txt" consisting 79772 words for simulating the straightforward monkey problem. Also for better comparisons and results we downloaded some books from Project Gutenberg [2] in addition to the ones given in the assignment question. We will give more explanations on this in the later part of our report.

For uniformity we declared our language `L` to have 40 characters of the form:

```
L = ['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p'
'q' 'r' 's' 't' 'u' 'v' 'w' 'x' 'y' 'z' ',' '.' ';' ':' '?' '!' '('
')' '-' '''' '"' '@' '#' ' '];
```

We simulated the straightforward monkey problem, first-order, second-order, and third-order monkey problem as per the assignment requirement. Out of curiosity we simulated the fourth-order and fifth-order monkey problem on few of the corpora provided to us. We saw a very prominent increase in the world yield and more meaningful longer phrases as the simulated order increased.

2.1 Function Descriptions

Returns	Functions	Description
String[ ] `corpus`	`getCorpus('path.txt')`	Takes in a string path and parses the contents into an array of words

String [ ] `singleLineCorpus`	`formatCorpus(corpus)`	Takes in a corpus and formats it into a single string (word by word). This is used to compute the correlation matrices
String[] `wordList`	`meaningfulWords(result, corpus)`	Checks the result string for the presence of any of the words in the corpus and returns the unique words that match
String[] `longestWord`	`findLongestWord(wordList)`	Outputs the words with the highest length from the generated words list
int [ ] `CorrMatrix1stOrder`	`getCorrMatrix1stOrder(corpus)`	Generates a 1st Order Correlation Matrix based on corpus
int [ ][ ] `CorrMatrix2ndOrder`	`getCorrMatrix2ndOrder(corpus)`	Generates a 2nd Order Correlation Matrix based on corpus
int [ ] [ ] [ ] `CorrMatrix3rdOrder`	`getCorrMatrix3rdOrder(corpus)`	Generates a 3rd Order Correlation Matrix based on corpus
int [ ] [ ] [ ] [ ] `CorrMatrix4thOrder`	`getCorrMatrix4thOrder(corpus)`	Generates a 4th Order Correlation Matrix based on corpus
int [ ] [ ] [ ] [ ] [ ] `CorrMatrix5thOrder`	`getCorrMatrix5thOrder(corpus)`	Generates a 5th Order Correlation Matrix based on corpus
`Typewriter` [ ]	`getTypewriters2ndOrder(CorrMatrix2ndOrder)`	Generates the typewriters which the monkey will use to generate random text. This can be reused for higher order simulations if Typewriters is defined with dimension n + 1 and a for loop is used to insert data into the added dimension. Also Typewriters of n^{th} Order is used to compute the Typewriters of $(n+1)^{th}$ Order
String result	`straightForwardMonkey(Language, iterations)`	Given a Language and the number of iterations, simulate the Straightforward and the 1st Order Monkey problem
String result	`secondOrderMonkey(typeWriter2ndOrder, Language, iterations)`	Given a Language, iterations, and the 2nd Order Typewriter, the monkey will simulate the 2nd Order Monkey problem
String result	`thirdOrderMonkey(typeWriter3rdOrder, Language, iterations)`	Given a Language, iterations, and the 3rd Order Typewriter, the monkey will simulate the 3rd Order Monkey problem
String result	`fourthOrderMonkey(typeWriter4thOrder, Language, iterations)`	Given a Language, iterations, and the 4th Order Typewriter, the monkey will simulate the 4th Order Monkey problem

String result	`fifthOrderMonkey(typeWriter5thOrder, Language, iterations)`	Given a Language, iterations, and the 5th Order Typewriter, the monkey will simulate the 5th Order Monkey problem
String path	`probPath(correleationMatrix2nd,Language,initial)`	Given a Language, a 2nd Order Correlation matrix, and an initial value, this method will compute the most probable digraph path
int [ ] [ ] `englishMatrix`	`generateEnglishMatrix(titles,Language,limit)`	Given an array of text file names (titles), and the Language, this method computes a standard English Matrix by averaging individual correlation matrices together. The limit puts a cap on the maximum amount of characters to be compared
int [ ] [ ] `author`	`correlateAuthors(overallMatrix,englishMatrix,Language)`	Given an overall matrix of the computed correlation matrices, the English matrix, and the language, this method gives a 2-D correlation matrix of n x n authors, with 1 being the highest correlation
char [ ] `grams`	`makeGrams(corpus,N)`	Given a corpus and N (the order of the N-gram), this function will generate an array of N length, unique grams
int [ ] `gramDistrib`	`getGramDistrib(corpus,grams,L)`	Given the corpus, grams, and L (Limit), this function returns a list of term frequency in descending order, cut off at the value L
int [ ] `cosSim`	`cosineSim(A,B)`	Compute the dot product between the term frequencies of the N-grams computed from two corpora

int [ ] [ ] `cosineSimilarity`	`getCosineSimilarity(titles,overallGrams)`
	`getProfileSimilarity(titles,overallGrams)`
		Given the `overallGrams,` which stores the N-gram distribution of all the corpora, this function computes the Cosine Similarity Measure for all the corpora

int [ ] [ ] `profileDissimilarity`	`getProfileDissimilarity(titles,overallGrams)`	Given the `overallGrams,` which stores the N-gram distribution of all the corpora, this function computes the Profile Dissimilarity Measure for all the corpora

Table 1. Descriptions of the series of *functions* used and reused in the programs to generate results for this assignment.

2.2 Program Descriptions

Problem	**Functions Used**	**Description**	**Output File**

1a.	`straightForwardMonkey()`
	`meaningfulWords()`
	`findLongestWord()`
		Simulates the straightforward monkey problem using the allowed characters declared in set `L`. Number of iterations used was `100000` for the generation of meaningful estimate of the yield of words. The typed characters were then compared with the `'dictionary.txt'` to determine the meaningful words generated within the string.	`straightForward-1a.txt`

1b.	`getCorpus()`
	`charHamlet()`
	`straightForwardMonkey()`
	`meaningfulWords()`
	`findLongestWord()`
		Simulates the first-order monkey problem on Hamlet Act III corpus, using the character distribution illustrated in Table 1 of the assignment question.
		The typed characters were then compared with the Hamlet corpus itself to determine the meaningful words generated within the string.
		We simulated the first-order monkey problem on other corpora. Character distribution used was those in `L.` The typed characters were then compared with the corpus itself to determine the meaningful words generated within the string.
			`1stOrder-1b-`*`CorpusName`*`.txt`

1c.	`getCorpus()`
	`formatCorpus()`
	`getCorrMatrix2ndOrder()`
	`getTypewriters2ndOrder()`
	`secondOrderMonkey()`
	`meaningfulWords()`
	`findLongestWord()`
		Simulates the second-order monkey problem on the merged Bronte novels and various other corpora from the list. Character distribution used was those in `L.` The typed characters were then compared with the corpus itself to determine the meaningful words generated within the string.	`2ndOrder-1c-`*`CorpusName`*`.txt`

	`getCorpus()` `formatCorpus()` `getCorrMatrix3rdOrder()` `getTypewriters3rdOrder()` `thirdOrderMonkey()` `meaningfulWords()` `findLongestWord()`	Simulates the third-order monkey problem on the merged Bronte novels and various other corpora from the list. Character distribution used was those in `L`. The typed characters were then compared with the corpus itself to determine the meaningful words generated within the string.	`3rdOrder-1c-`*`CorpusName`*`.txt`
	`getCorpus()` `formatCorpus()` `getCorrMatrix4thOrder()` `getTypewriters4thOrder()` `fourthOrderMonkey()` `meaningfulWords()` `findLongestWord()`	Simulates the fourth-order monkey problem on the merged Bronte novels and various other corpora from the list. Character distribution used was those in `L`. The typed characters were then compared with the corpus itself to determine the meaningful words generated within the string.	`4thOrder-1c-`*`CorpusName`*`.txt`
	`getCorpus()` `formatCorpus()` `getCorrMatrix5thOrder()` `getTypewriters5thOrder()` `fifthOrderMonkey()` `meaningfulWords()` `findLongestWord()`	Simulates the fifth-order monkey problem on Hamlet Act III and the corpus, 'Legend of Sleepy Hollow'. Character distribution used was those in `L`. The typed characters were then compared with the corpus itself to determine the meaningful words generated within the string.	`5thOrder-1c-`*`CorpusName`*`.txt`
1d.	`getCorpus()` `formatCorpus()` `getCorrMatrix2ndOrder()` `getTypewriters2ndOrder()` `secondOrderMonkey()` `getCorrMatrix3rdOrder()` `getTypewriters3rdOrder()` `thirdOrderMonkey()` `meaningfulWords()` `findLongestWord()`	Investigates the effect of resolution on monkey literacy in the simulation. Resolution factor of 0.9, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001 were implemented on the second-order and third-order simulation to observe their effect on the word yield.	`Altered2ndOrder-1d-`*`CorpusNameFactor`*`.txt` `Altered3rdOrder-1d-`*`CorpusNameFactor`*`.txt`
1e.	`getCorpus()` `formatCorpus()` `getCorrMatrix1stOrder()` `getCorrMatrix2ndOrder()`	Routine to compute first-order and second-order correlation matrices.	GUI display of first-order and second-order correlation matrices.
1f.	`getCorpus()` `getCorrMatrix2ndOrder()` `probPath()`	Computing the Most Probable Digraph Path using the second-order correlation matrix for all the corpora in the list	GUI display of the Most Probable Digraph Path for all the corpora in the list

1g.	getCorpus()
	getCorrMatrix2ndOrder()
	generateEnglishMatrix()
	correlateAuthors()
		Designed Experiment# 01: Author Attribution using the Book Algorithm. Bennett Chapter-4, Equation# 15, Page 127 [3].	GUI display of the Author Attribution with Book Algorithm

	getCorpus()
	makeGrams()
	getGramDistrib()
	cosineSim()
	getCosineSimilarity()
		Designed Experiment# 02: Author Attribution using N-Gram and Cosine Similarity Measure [4].	GUI display of the Author Attribution using Cosine Similarity Measure.

	getCorpus()
	makeGrams()
	getGramDistrib()
	getProfileDissimilarity()
		Designed Experiment# 03: Author Attribution using N-Gram and Profile Dissimilarity Measure [5]	GUI display of the Author Attribution using Profile Dissimilarity Measure.

1h.	getCorpus()
	getCorrMatrix2ndOrder()
	generateEnglishMatrix()
	correlateAuthors()
		Performing Genre Classification. Trial # 01 with Genres: Lost World, Fantasy, Romance using the Book Algorithm. Bennett Chapter-4, Equation# 15, Page 127 [3].	GUI display of the Genre Classification: Trial # 01

	getCorpus()
	getCorrMatrix2ndOrder()
	generateEnglishMatrix()
	correlateAuthors()
		Performing Genre Classification. Trial # 02 with Genres: Lost World, Crime, Non-fiction Psychology, Horror using the Book Algorithm. Bennett Chapter-4, Equation# 15, Page 127 [3].	GUI display of the Genre Classification: Trial # 02

	getCorpus()
	makeGrams()
	getGramDistrib
	cosineSim()
	getCosineSimilarity()
		Genre Classification. Trial # 03 with Genres: Lost World, Crime, Non-fiction Psychology, Horror using Cosine Similarity Measure [4].	GUI display of the Genre Classification: Trial # 03

1i.	getCorpus()
	makeGrams()
	getGramDistrib
	cosineSim()
	getCosineSimilarity()
		Developing Profiles for each of the different authors given in Table 2 of the assignment using Cosine Similarity Measure [4].	GUI display of the Profile Similarity between Authors.

Table 2. Descriptions of the series of *programs* coded to generate the results for this assignment.

3. Algorithm

This section discusses the algorithms developed for solving each of the problems.

Algorithm 3.1: Straightforward Monkey Problem/ First-Order Monkey Problem

```
1  Initialize a character array which will hold the results of the simulation – result
2  Set the length of array equal to iteration
3  For every iteration -> i
4          Generate a random number between 1 and the total number of elements in
           Language (L) -> randi
5          Use the randi to index a random letter from L
6          Append to the character array - result
```

Algorithm 3.1 for Problem 1a requires us to simulate the straightforward monkey problem using the allowed characters declared in set `L`. Number of iterations used was `100000` for the generation of meaningful estimate of the yield of words.

The typed characters were then compared with the `'dictionary.txt'` to determine the meaningful words generated within the string. The program outputs only unique words and also outputs words with the highest length among the generated words list.

The same above algorithm, Algorithm 3.1, was used again for Problem 1b, to simulate the first-order monkey problem using the character distribution illustrated in Table 1 of the assignment question (which illustrates the 35,224 character distribution from Hamlet Act III). Hamlet Act III was used as the corpus this time and hence, the typed characters were compared with the Hamlet corpus itself to determine the meaningful words generated within the string.

We simulated the first-order monkey problem on other corpora too. Character distribution used was those in `L.` The typed characters were then compared with the corpus itself to determine the meaningful words generated within the string.

Algorithm 3.2.1: Second-Order Correlation Matrix

```
1  Set the length of the matrix according to the length of language L
   matrix(length(L), length(L))
2  For every letter in the language -> i
3          For every letter in the language -> j
4                  Generate string to be searched [L(i) L(j)]
5                  Search through result to get the number of occurrences of search
                   string -> occurrences
6                  Store the occurrences in matrix(i, j)
```

Algorithm 3.2.2: Second-Order Typewriters

```
1  Define cell array of Typewriters
2  For every element in the Second-Order Correlation Matrix -> i
3      For every element in the Second-Order Correlation Matrix -> j
4          Get frequency of string:  freq -> matrix(i,j)
5          Create vector, charGen of length freq which contains the letter being
6          evaluated-> L(j)
7          Append charGen  to the i^th typewriter's distribution
```

Algorithm 3.2.3: Second-Order Monkey Problem

```
1  Randomly select a Typewriter(i) for the monkey to start
2  Set a result string which will hold the simulation results
3      For every new character in result
4          Get distribution of Typewriter(i) keys -> distrib
5              If the distrib is empty, randomly select another keyboard
               and keep selecting until distrib is not empty
6          Pick random letter from this distribution
7          Append to result
8          Locate which Typewriter this key belonged to
```

Algorithm 3.2.1 – Algorithm 3.2.3 simulates the second-order monkey problem for Problem 1c. Algorithm 3.2.1 illustrates how to build the second-order correlation matrix based on which the second-order Typewriters were built, as shown by Algorithm 3.2.2. Now the number of total Typewriters generated will be N^2, where N is the number of characters in Language L. The second-order Typewriters were then used to simulate the second-order monkey problem as illustrated by Algorithm 3.2.3.

As corpus, the three Bronte novels were merged and character distribution used was those in L. In addition to the merged Bronte novels, many other corpora were used to observe the results generated by the second-order monkey problem. The typed characters were then compared with the corpus itself to determine the meaningful words generated within the string.

Algorithm 3.3.1: Third-Order Correlation Matrix

```
1  Set the length of the matrix according to the length of language L
   matrix(length(L), length(L))
2  For every letter in the language -> i
3      For every letter in the language -> j
4          For every letter in the language -> k
5              Generate string to be searched [L(i) L(j) L(k)]
6              Search through result to get the number of occurrences of search
               string ->  occurrences
7              Store the occurrences in matrix
```

Algorithm 3.3.2: Third-Order Typewriters

```
1   Define cell array of Typewriters
2   For every element in Language L-> i
3           Generate a set of all typewriters for each letter using the Second-Order Typewriters
            resulting to $(length(L))^3$ Typewriters in total
```

Algorithm 3.3.3: Third-Order Monkey Problem

```
1   Randomly select a Typewriter(i) as the first letter index for the monkey to start
2   Randomly select a Typewriter(j) for the second letter index
3   Set a result string which will hold the simulation results
4           For every new character in result
5                   Get distribution of Typewriter(i,j) key -> distrib
6                           If the distrib is empty, randomly select another keyboard
                            and keep selecting until distrib is not empty
7                   Pick random letter from this distribution -> k
8                   Append to result
9                   Set i = j
10                  Set j = k
```

Algorithm 3.3.1 – Algorithm 3.3.3 simulates the third-order monkey problem for Problem 1c. Algorithm 3.3.1 illustrates how to build the third-order correlation matrix based on which the third-order Typewriters were built, as shown by Algorithm 3.3.2. There is an extra dimension to the third-order correlation matrix as it is now 3-Dimensional. The added dimension, *k*, means that to compute the correlation matrix, the amount of times *i* is followed by *j* is followed by *k* must be computed.

We have reused the Second-Order Typewriters in order to build the Third-Order Typewriters for efficiency of the program. Now the number of total Typewriters generated will be $\mathbf{N^3}$, where **N** is the number of characters in Language L. The third-order Typewriters were then used to simulate the third-order monkey problem as illustrated by Algorithm 3.3.3.

From the Algorithms for Second-Order and Third-Order Monkey problem, we can clearly see that the number of Typewriters for each order will increase with each order in the following pattern: $\mathbf{N^{ORDER}}$ [3].

Out of curiosity we also implemented the Fourth-Order and Fifth-Order Monkey simulations. From the previous Algorithm sets 3.2 and 3.3, it can be seen that the dimension of the matrix increase by order. Also we reuse Third-Order Typewriter to generate Fourth-Order Typewriter, Fourth-Order Typewriter to generate Fifth-Order Typewriter and so on. Algorithm sets of 3.4 and 3.5 of Fourth-Order & Fifth-Order Simulation respectively illustrate this concept of reusing function.

Algorithm 3.4.1: Fourth-Order Correlation Matrix

```
1  Set the length of the matrix according to the length of language L
   matrix(length(L), length(L))
2  For every letter in the language -> i
3      For every letter in the language -> j
4          For every letter in the language -> k
5              For every letter in the language -> l
6                  Generate string to be searched [L(i) L(j) L(k) L(l)]
7                  Search through result to get the number of occurrences of
                   search string ->  occurrences
8                  Store the occurrences in matrix
```

Algorithm 3.4.2: Fourth-Order Typewriters

```
1  Define cell array of Typewriters
2  For every element in Language L-> i
3      Generate a set of all typewriters for each letter using the Third-Order Typewriters
       resulting to $(length(L))^4$ Typewriters in total
```

Algorithm 3.4.3: Fourth-Order Monkey Problem

```
1  Randomly select a Typewriter(i) as the first letter index for the monkey to start
2  Randomly select a Typewriter(j) for the second letter index
3  Randomly select a Typewriter(k) for the third letter index
4  Set a result string which will hold the simulation results
5      For every new character in result
6          Get distribution of Typewriter(i,j,k) key -> distrib
7              If the distrib is empty, randomly select another keyboard
               and keep selecting until distrib is not empty
8          Pick random letter from this distribution -> l
9          Append to result
10         Set i = j
11         Set j = k
12         Set k = l
```

Algorithm 3.5.1: Fifth-Order Correlation Matrix

```
1   Set the length of the matrix according to the length of language *L*
    *matrix(length(L), length(L))*
2   For every letter in the language -> *i*
3       For every letter in the language -> *j*
4           For every letter in the language -> *k*
5               For every letter in the language -> *l*
6                   For every letter in the language -> *m*
7                       Generate string to be searched *[L(i) L(j) L(k) L(l)L(m)]*
8                       Search through *result* to get the number of occurrences
                        of search string -> *occurrences*
9                       Store the occurrences in *matrix*
```

Algorithm 3.5.2: Fifth-Order Typewriters

```
1   Define cell array of Typewriters
2   For every element in Language *L*-> *i*
3       Generate a set of all typewriters for each letter using the Fourth-Order Typewriters
        resulting to $(length(L))^5$ Typewriters in total
```

Algorithm 3.5.3: Fifth-Order Monkey Problem

```
1   Randomly select a *Typewriter(i)* as the first letter index for the monkey to start
2   Randomly select a *Typewriter(j)* for the second letter index
3   Randomly select a *Typewriter(k)* for the third letter index
4   Randomly select a *Typewriter(l)* for the fourth letter index
5   Set a *result* string which will hold the simulation results
6       For every new character in *result*
7           Get distribution of *Typewriter(i,j,k,l)* key -> *distrib*
8           If the *distrib* is empty, randomly select another keyboard
                and keep selecting until *distrib* is not empty
9           Pick random letter from this distribution -> *m*
10          Append *to result*
11          Set *i = j*
12          Set *j = k*
13          Set *k = l*
14          Set *l = m*
```

For Problems 1d and 1e we reuse the Algorithm sets of 3.2 and 3.3.

Algorithm 3.6: Most Probable Digraph Path

```
1  Select and store the letter we want to start with -> i
2  Declare the result string variable path
3  For every letter in the Language L
4      Append the new letter to path
5      Go to the $i^{th}$ row of the Second-Order Correlation Matrix
6          Find the maximum value in this row which is not already in path
7              If this row has all zeros
8                  then  break out of the loop, END
9              Else if the row is not empty
10                 this column value is the new i
```

Algorithm 3.6 illustrates the algorithm for computing the most probable digraph path. The most probable digraph path can be used for the purpose of language identification, since the algorithm seeks to find the most probable single occurrence of a letter within the Second-Order Correlation Matrix.

Algorithm 3.7: Author Attribution - Correlation Matrix

```
1  Initialize an n by n result matrix -> authorCorpus
2  For every corpus in the list -> i
3      For every other corpus  in the list-> j
4          If i = j (if same corpus being compared to itself)
5              Compare corpus to different corpus (i vs i + number of Authors)
6          Else if i != j
7              Compare corpus regularly (i vs j)
8          Store result in matrix authorCorpus
9   Normalize authorCorpus by largest value in authorCorpus
```

Algorithm 3.7 illustrates the algorithm for determining Author Attribution as part of solving Problem 1g. We designed three experiments for this problem and Algorithm 3.7 shows the algorithm for Experiment No. 1, which is based on the the Book Algorithm (Bennett Chapter-4, Equation# 15, Page 127) [3].

We selected two corpuses for each other author for this experiment, which explains Line 4 in Algorithm 3.7. We didn't want the same corpus to be compared with itself. Instead we wanted each corpus to compare with every other corpus (which includes the second corpus by the same author and also to other corpus by other authors). We followed this strategy in order to see whether we can determine each author, since the *authorCorpus* is supposed to give the highest value for cases when we are comparing two corpora by the same author rather than one corpus from one author and the second one from a different author. Details of the results generated from this experiment is discussed in more details in our next Section (4. Result & Analysis).

Algorithm 3.8: Make Grams

```
1 Generate number of grams based on N -> numGrams = (length of String – N) + 1
2 For every gram -> i
3     Take substring of input string by starting at index i and going to index i + (N-1)
4     Store gram in an array -> grams{numGrams}
5 Filter out every non-unique gram -> unique(grams)
```

Algorithm 3.9: Get Gram Distribution

```
1 For every gram -> g
2     Find the total number of occurrences of this gram in the original string-> occurences
3     Get the frequency distribution of the occurrences
      -> occurences/length (grams{numGrams})
4 Sort by descending order
```

For our next two experiments for determining Author Attribution of Problem 1g, we require N-gram analysis. Algorithms 3.8 and 3.9 show how to generate the grams and how to determine the distribution of each gram respectively.

N-Gram analysis involves taking an input string, breaking down the strings into small blocks called grams, and then generating a profile of the grams which occur most in the original string. For generating the profile, we need to count the number of occurrences of the different grams and then determine their frequency distribution.

Algorithm 3.10: Author Attribution - Cosine Similarity Measure

```
1 Initialize an n by n result matrix -> cosineSimilarity
2 For every corpus in the list -> i
3     For every other corpus  in the list-> j
4         If i = j (if same corpus being compared to itself)
5             Compare corpus to different corpus (i vs i + number of Authors)
6             Compute the cosineSim
7         Else if i != j
8             Compute the cosineSim
9         Store result in matrix cosineSimilarity
```

Algorithm 3.11: Author Attribution- Profile Dissimilarity Measure

```
1 Initialize an n by n result matrix -> profileDissimilarity
2 For every corpus in the list -> i
3     For every other corpus  in the list-> j
4         If i = j (if same corpus being compared to itself)
5             Compare corpus to different corpus (i vs i + number of Authors)
6             Compute the profileDissimilarityMeasure
7         Else if i != j
8             Compute the profileDissimilarityMeasure
9         Store result in matrix profileDissimilarity
```

Now that we have N-grams generated along with their frequency distribution (Algorithms 3.8 and 3.9), we can implement our remaining two experiments: Experiment No. 2: Author Attribution Cosine Similarity Measure [4] and Experiment No. 3: Author Attribution Profile Dissimilarity Measure [5]. Both these experiments are based on N-grams analysis. Algorithms 3.10 and 3.11 illustrates Experiments No. 2 and respectively. The same approach of comparing two corpora from same author and not comparing the same corpus with itself (as seen in Algorithm 3.7) is considered here. In our next section we will discuss in details about the *cosineSim* function and *profileDissimilarity* measure.

For Problem 1h we reuse the Algorithms 3.7 and 3.10.

For Problem 1i we reuse the Algorithm 3.10 with a very small modification. We don't implement the approach of not comparing the same corpus to itself (as shown in Line 4). Why we made this modification-we will discuss it in our Result and Analysis Section.

4. Results & Analysis

This section of our report discusses and analyzes the results generated for solving Problems 1a to 1i.

4.1. Results: Problems 1a-1c

We discuss the results generated for Problems 1a to 1c in this sub-section. Since Problems 1a to 1c involved simulations of straightforward, first-order, second-order and so on, we decided to discuss their results in one section. This will enable us to give a better comparison between the results generated for each simulation.

Problem 1a (illustrated in Algorithm 3.1) simulates the straightforward monkey problem using the allowed characters declared in set `L`. Our set `L` had forty characters as mentioned in the beginning of Section 2. Number of iterations used was `100000` for the generation of meaningful estimate of the yield of words. The typed characters were then compared with the `'dictionary.txt'` to determine the meaningful words generated within the string. The total number of words in `'dictionary.txt'` was 79,772. While determining the meaningful words generated from the scribble by the monkey, we counted the unique words generated and discarded the duplicates. From this count of unique words we calculated our percentage yield of valid words using the following:

```
Total % Yield of Valid Words: (countWords/countDictionaryWords)*100)
```

Figure 1 illustrates the result generated for Problem 1a. The number of valid words generated was 681; percentage yield of valid words was 0.85%. A few of the longest word generated were: **`debar, ducks, gully, uncap`.**

```
Number of Unique Valid Words found: 681
Number of Words in the Dictionary: 79772
Total % Yield of Valid Words: 0.85%
Words with highest length: debar, ducks, gully, uncap
```

Figure 1. Sample result generated for Problem 1a simulating the Straightforward Monkey Problem.

Problem 1b (illustrated in Algorithm 3.1) simulates the First-Order Monkey problem on Hamlet Act III corpus, using the character distribution illustrated in Table 1 of the assignment question. The typed characters were then compared with the Hamlet corpus itself to determine the meaningful words generated within the string. The total number of words in the Hamlet Act III is 7622. The number of valid words generated was 193; percentage yield of valid words was 2.53%; length of the longest word was 5. A few of the longest word generated were: **`bosom, fears, moons, sweat`.**
We simulated the first-order monkey problem on other corpora. Character distribution used was those in `L.` The typed characters were then compared with each corpus itself to

determine the meaningful words generated within the string. We noticed that the longest word generated for all the corpora was 5.

Problem 1c simulates the Second-Order and the Third-Order simulations. However, out of curiosity we also implemented the Fourth-Order and Fifth-Order simulations. The Algorithm sets of 3.2 to 3.5 illustrate these simulation procedures. We implemented from the First-Order till the Fourth-Order simulations on the Hamlet Act III corpus, merged Bronte novels and six more corpora (Table 3 and Figure 2). However, for the Fifth-Order simulation we selected only two corpora: Hamlet Act III and Legend of Sleepy Hollow (Table 4 and Figure 3).

In both the tables (Table 3 and 4) and figures (Figures 2 and 3), we see a considerable increase in percentage yield of words as we progressed from the First-Order to Fifth-Order simulation. This observation was consistent for all the corpora. However, for the merged Bronte corpus, despite there was an increase all the time, but the percentage increase was less in all the orders compared to other corpora. One reason maybe because the merged Bronte novel was bigger in size compared to other corpora with a total of 368202 words. Since the monkey was allowed to type up to 100000 characters only, the number of unique valid words generated will be quite small compared to the huge size of this merged corpus.

	Hamlet	Tale of 2 Cities	Bronte	Alice in Wonderland	Christmas Carol	Legend of Sleepy Hollow	Metamor phosis	The Time Machine
1st Order	2.53%	0.25%	0.15%	0.68%	0.82%	1.28%	0.77%	0.56%
2nd Order	5.29%	0.66%	0.39%	1.94%	2.23%	3.64%	1.91%	1.8%
3rd Order	9.01%	1.06%	0.56%	3.33%	3.5%	5.5%	3.32%	2.68%
4th Order	17.02%	1.85%	0.94%	6.52%	6.53%	11.59%	6.65%	5.24%

Table 3. Percentage yield of valid words for First-Order to Fourth-Order simulation of 8 corpora

	Hamlet	Legend of Sleepy Hollow
1st Order	2.53%	1.28%
2nd Order	5.29%	3.64%
3rd Order	9.01%	5.5%
4th Order	17.02%	11.59%
5th Order	25.85%	20.14%

Table 4. Percentage yield of valid words for First-Order to Fifth-Order simulation of 2 corpora

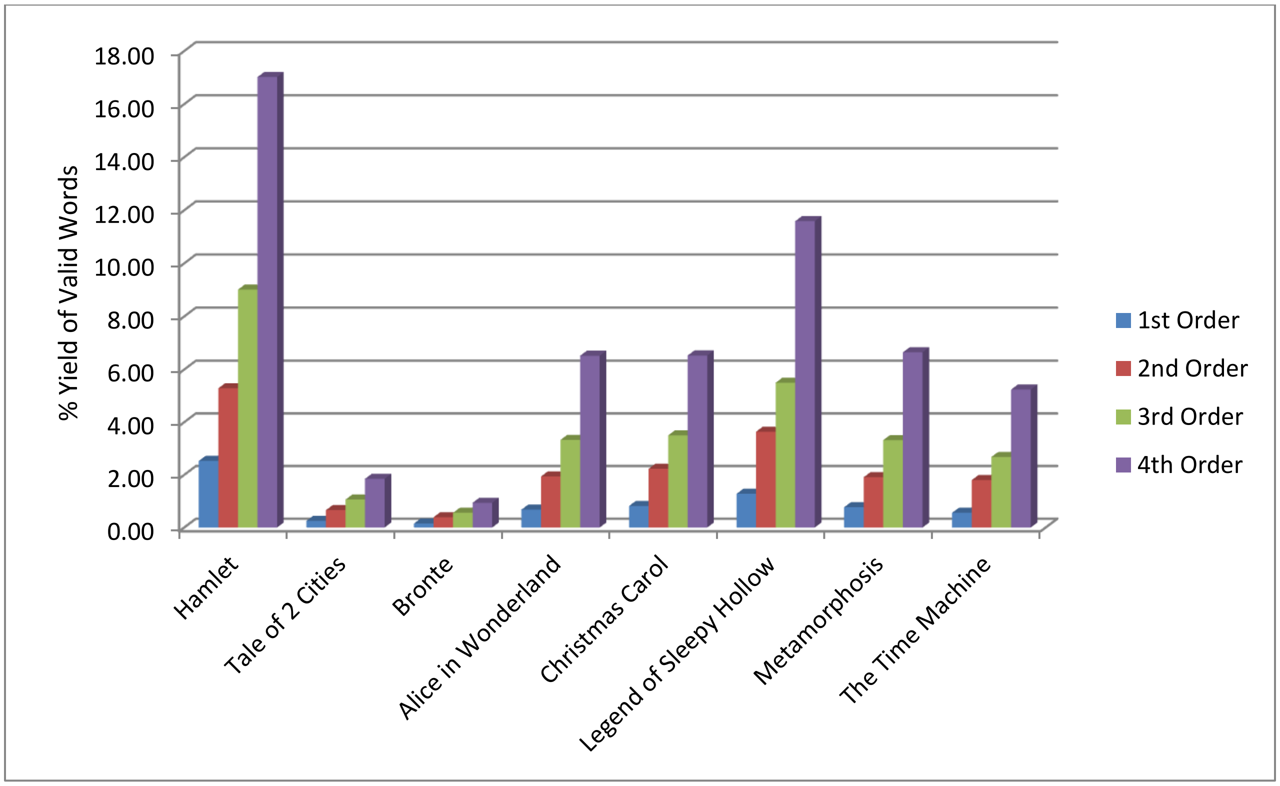


Figure 2. This figure shows the graphical representation of Table 3. It shows the increase in the percentage of word yields as we go from the First-Order to the Fourth-Order simulation for 8 corpora.

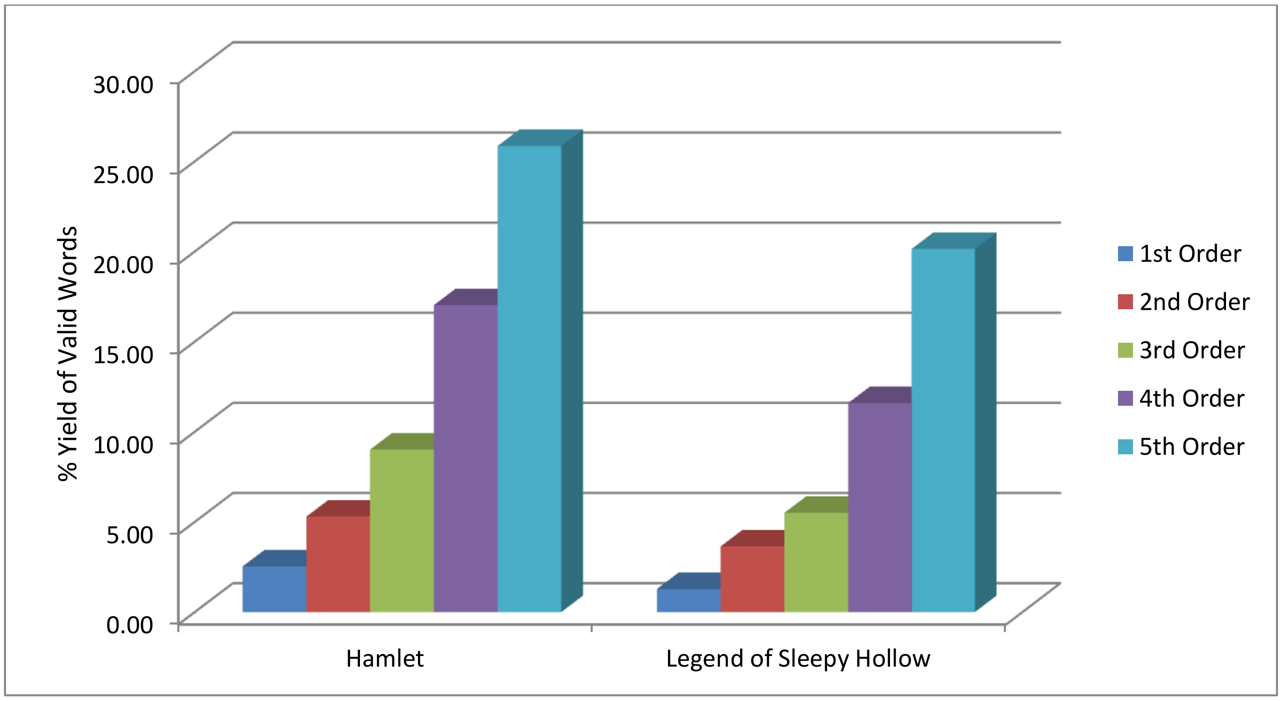


Figure 3. This figure shows the graphical representation of Table 4. It shows the increase in the percentage of word yields as we go from the First-Order to the Fifth-Order simulation for 2 corpora.

The other thing we observed was in the length of the valid words generated. As the order increased, so did the length of the words. Table 5 illustrates this observation. By the time we reached Fifth-Order, the Monkey was typing short meaningful phrases like:

```
'the perplexible luxuriously atmosphere'
'his gathered afternoons of justice'
'the circle of the devil into a henroost'
'but the king and these pictures are of love'
```

	Length of Words	Example of words
1st Order	5	their
2nd Order	7	touches
3rd Order	9	mentioned
4th Order	12	schoolmaster
5th Order	16	promise-crammed:

Table 5. Increase in the length of the valid words generated as the order of simulation increased

4.2. Results: Problem 1d

For Problem 1d we investigate the effects of resolution on monkey literacy. In order to achieve this, we coded a program which would divide/multiply all the entries in the Correlation Matrix by a constant factor. Since the Typewriters were generated from the Correlation Matrices, dividing/multiplying each entries of the Correlation Matrix would reduce the number of keys on the typewriter. In this way, the probabilities do not get affected since we were simply scaling the probabilities by a constant factor. What makes things interesting however, when scaling down the probabilities and *rounding off* the individual elements of the Correlation Matrix, the least probable frequencies began to go to zero. Our intuition was that, since these probabilities go to zero, the chances of making a word with the other more probable letter sets were higher, and thus the word yield might improve, although the word variation would decrease since the monkeys would get quite repetitive.

In order to see whether our intuition was correct, we applied the resolution factor on Second-Order Correlation Matrix and Third-Order Correlation Matrix. The corpus used was "Legend of Sleepy Hollow". Table 6 and its corresponding graphical representation in Figure 4 illustrate the effect of resolution by various factors on the Second-Order Correlation Matrix. Similarly, Table 7 and its corresponding graphical representation in Figure 5 illustrate the effect of resolution on the Third-Order Correlation Matrix.

Resolution Factor	2nd Order	Factorized 2nd Order
0.9	3.64%	3.87%
0.5	3.64%	3.65%
0.1	3.64%	3.72%
0.05	3.64%	3.47%
0.01	3.64%	2.36%
0.005	3.64%	1.07%
0.001	3.64%	0.03%

Table 6. Effect of resolution on the Second-Order Correlation Matrix and its corresponding percentage word yield

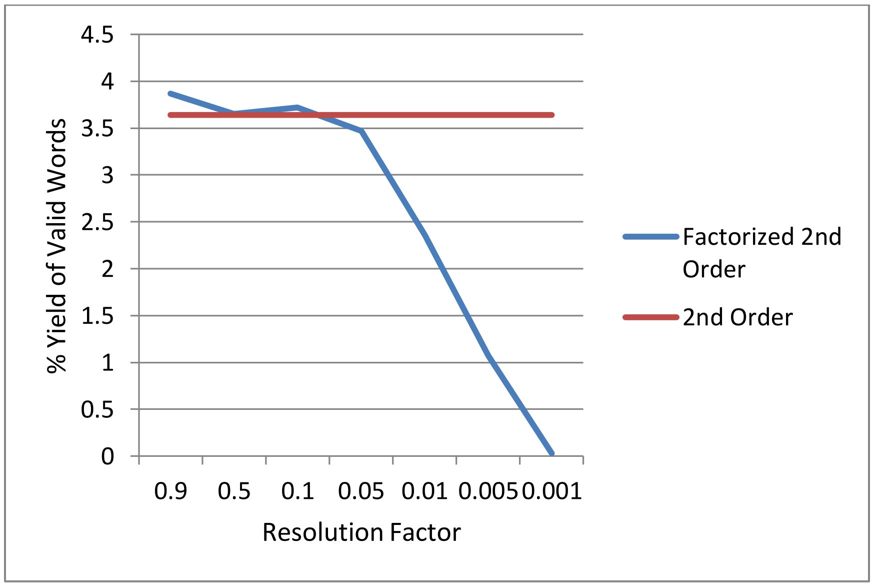


Figure 4. This figure shows the graphical representation of Table 6. As the resolution factor decreased, in the beginning there was a slight increase in the percentage word yield. But with decrease in the resolution factor, the word yield decreased drastically and came close to zero.

Resolution Factor	3rd Order	Factorized 3rd Order
0.9	5.5%	5.63%
0.5	5.5%	5.66%
0.1	5.5%	3.76%
0.05	5.5%	2.28%
0.01	5.5%	0.32%
0.005	5.5%	0.18%

Table 7. Effect of resolution on the Third-Order Correlation Matrix and its corresponding percentage word yield

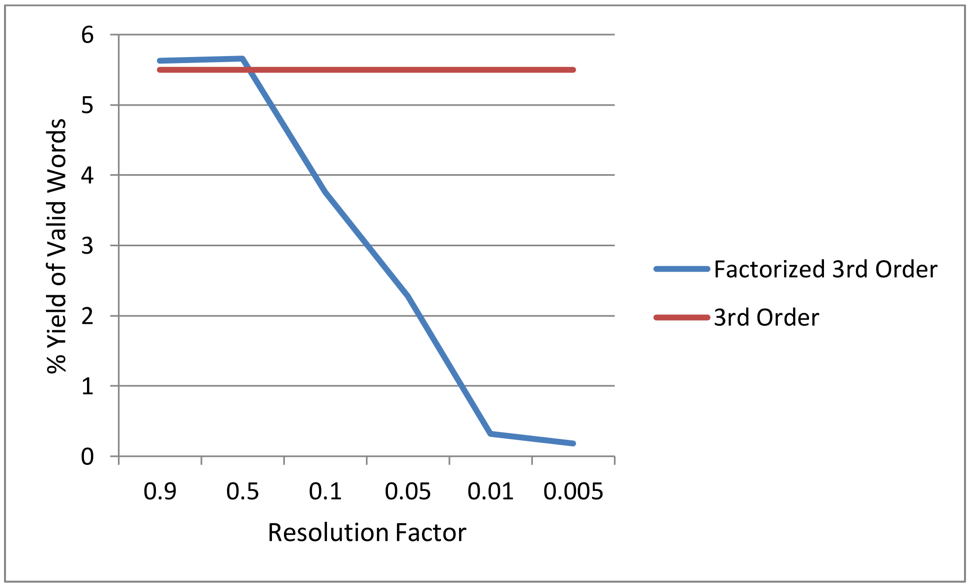


Figure 5. This figure shows the graphical representation of Table 7. As the resolution factor decreased, in the beginning there was a very minute increase in the percentage word yield, almost equal to that of the original simulation. But with decrease in the resolution factor, it decreased drastically and came close to zero. The rate of reduction in the percentage word yield observed for the Third-Order Simulation was higher compared to the Second-Order Simulation (Figure 4).

From Tables 6 and 7 and Figures 4 and 5, we can see that in the beginning there was increase in word yield for the Second-Order Simulation and very minute increase for the Third-Order. But as the resolution factor was reduced the word yield started decreasing. The rate of decrease observed for Third-Order was more drastic compared to the Second-Order simulation.

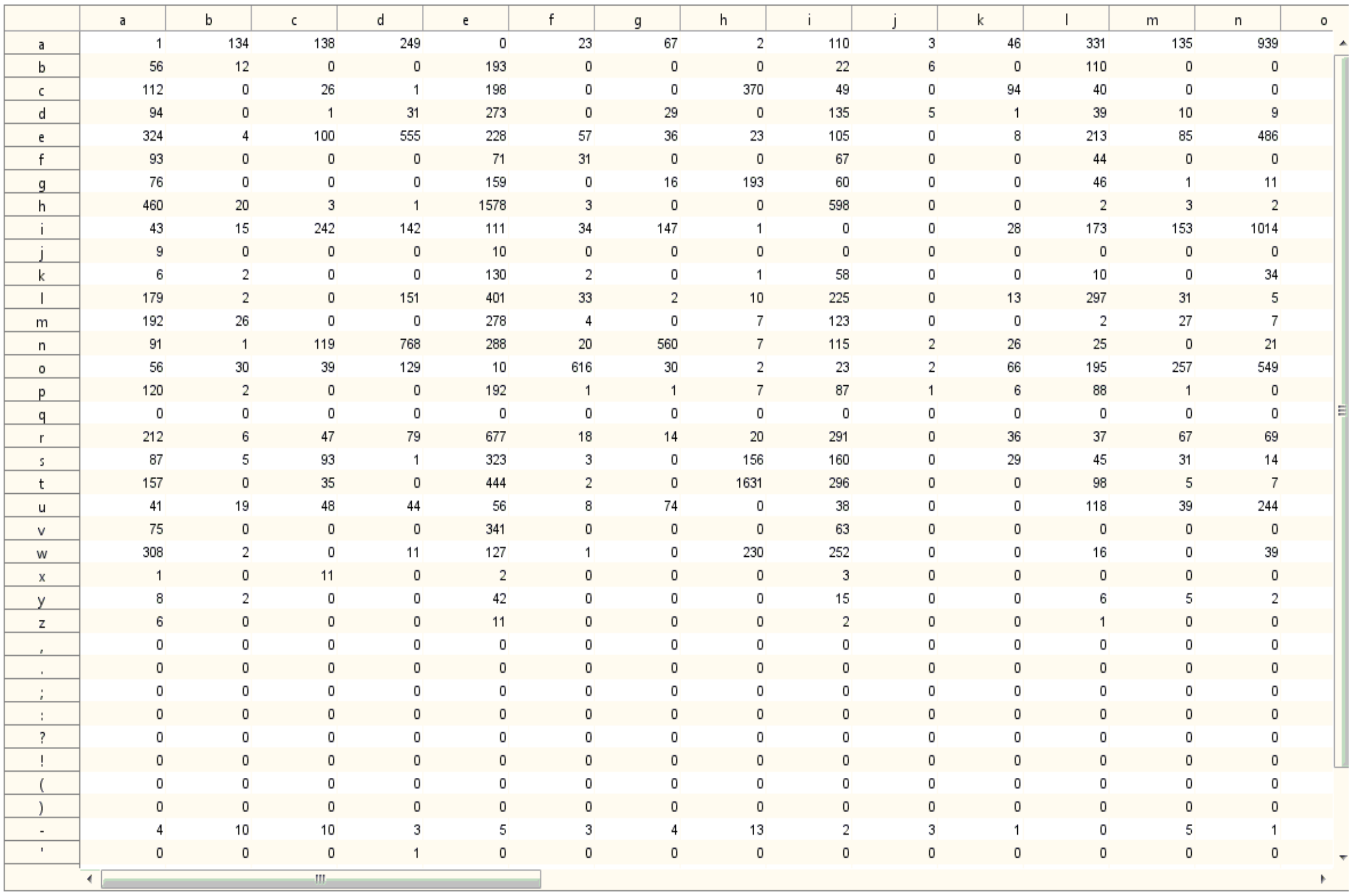
This can be explained by the fact that the scaling of the matrices is putting lots of zeros into the altered correlation matrix, and only the most dominating letters are being picked, such as "space". Each simulation algorithm has a property in which if a particular typewriter has got an empty character set, the monkey randomly reaches for typewriters until a valid typewriter is found. This randomness, added with the fact that there are lots of typewriters for which there are empty distributions, explain why the word yield drops when the correlation matrices are so sparse.

4.3. Results: Problem 1e

Functions to compute Correlation Matrices have been used in previous problems. We have implemented GUI display for this part of our problem. When the program for Problem 1e is executed, GUI display of First-Order Correlation Matrix and Second-Order Correlation Matrix will be displayed.

a 4292	b 919	c 1336	d 2463	e 6360	f 1324	g 1233	h 3698	i 3517	j 61
k 419	l 2193	m 1205	n 3659	o 4163	p 991	q 67	r 3197	s 3495	t 4645
u 1424	v 526	w 1222	x 47	y 850	z 30	, 1020	. 310	; 173	: 4
? 6	! 18	(1	) 1	- 131	‘ 36	“ 36	@ 0	# 0	space 11729

Figure 6. First-Order Correlation Matrix of the corpus “Legend of Sleepy Hollow”



	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
a	1	134	138	249	0	23	67	2	110	3	46	331	135	939	
b	56	12	0	0	193	0	0	0	22	6	0	110	0	0	
c	112	0	26	1	198	0	0	370	49	0	94	40	0	0	
d	94	0	1	31	273	0	29	0	135	5	1	39	10	9	
e	324	4	100	555	228	57	36	23	105	0	8	213	85	486	
f	93	0	0	0	71	31	0	0	67	0	0	44	0	0	
g	76	0	0	0	159	0	16	193	60	0	0	46	1	11	
h	460	20	3	1	1578	3	0	0	598	0	0	2	3	2	
i	43	15	242	142	111	34	147	1	0	0	28	173	153	1014	
j	9	0	0	0	10	0	0	0	0	0	0	0	0	0	
k	6	2	0	0	130	2	0	1	58	0	0	10	0	34	
l	179	2	0	151	401	33	2	10	225	0	13	297	31	5	
m	192	26	0	0	278	4	0	7	123	0	0	2	27	7	
n	91	1	119	768	288	20	560	7	115	2	26	25	0	21	
o	56	30	39	129	10	616	30	2	23	2	66	195	257	549	
p	120	2	0	0	192	1	1	7	87	1	6	88	1	0	
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
r	212	6	47	79	677	18	14	20	291	0	36	37	67	69	
s	87	5	93	1	323	3	0	156	160	0	29	45	31	14	
t	157	0	35	0	444	2	0	1631	296	0	0	98	5	7	
u	41	19	48	44	56	8	74	0	38	0	0	118	39	244	
v	75	0	0	0	341	0	0	0	63	0	0	0	0	0	
w	308	2	0	11	127	1	0	230	252	0	0	16	0	39	
x	1	0	11	0	2	0	0	0	3	0	0	0	0	0	
y	8	2	0	0	42	0	0	0	15	0	0	6	5	2	
z	6	0	0	0	11	0	0	0	2	0	0	1	0	0	
,	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
;	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
!	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
(	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
-	4	10	10	3	5	3	4	13	2	3	1	0	5	1	
'	0	0	0	1	0	0	0	0	0	0	0	0	0	0	

Figure 7. Partial screenshot of the GUI Display of Second-Order Correlation Matrix of the corpus “Legend of Sleepy Hollow”.

4.4. Results: Problem 1f

Problem 1f (illustrated in Algorithm 3.6) generates the most probable digraph path using the Book algorithm [3] with the Second-Order Correlation Matrix.

Author	Title	Most Probable Digraph Path
Irving	Legend of Sleepy Hollow	the andisofrylupkbjacqugmrvu
Poe	Gold Bug (From Book)	the andisouryplf'bj

Figure 8. Comparison of the digraph paths between Irving (Legend of Sleepy Hollow) and Poe (The Gold Bug).

Figure 8 shows the comparison of the digraph paths generated from the "Legend of Sleepy Hollow" by Irving and "The Gold Bug" by Poe. We see that the first 10 characters are identical and both have the common words: "the", "and", "is", "so", which is the same case for most English authors (see Figure 9). As the Book [3] mentioned on page 130, stylistic differences begins after the 6th letter. But in the above case Irving's digraph path starts differing from Poe on the 11th character by using "fry" instead of "our".

Figure 9 shows the most probable digraph paths generated for all corpora provided in the list. It can be seen that different corpus by the same author have very similar digraph paths. This can be used as a key for performing author attribution. The only digraph path which does not start with "the and" is from the Adventures of Huckleberry Finn by Twain. One probable reason can be because it is written in Southern Vernacular rather than standard English.

We gave implemented GUI display to generate the output for this problem.

Author	Title	Most Probable Digraph Path
Wells	The Time Machine	the andisofrycklugmpwbjavr'k
Wells	War of The Worlds	the andisofrylupmbjigwckcqu'
Cleland	Fanny Hill	the andisofrympluckwg'ypbjav
Irving	Legend of Sleepy Hollow	the andisofrylupkbjacqugmrvu
Buroughs	Warlord of Mars	the andisorulympwf'lv bjugyc
Machiavelli	The Prince	the andisoryblfuckw'wkompfsg
Buroughs	Tarzan of the Apes	the andisorzlypugmbjeckwf'vu
Buroughs	The People that Time Forgot	the andisoulyprmbjickfwyg'vy
Buroughs	The Land that Time Forgot	the andisourmylf'ckwyp'v bjr
Twain	A Connecticut Yankee in King Arthur 's Court	the andisouryblfg'v'mpkwjztc
Haggard	King Solomon's Mines	the andisoury'cklfpmbjigwcqu
Doyle	The Lost World	the andisourylf'v wbjickgmpn
Bronte E	Wuthering Heights	the andisoury'lfngmpwkbjavrc
Bronte C	Jane Eyre	the andisourymplf'ckwbjigdv'
Doyle	Tales of Terror and Mystery	the andourisplymbjackw'v f'g
Bronte A	Agnes Grey	the andisoury'wlfspmbjackug'
Carroll	Through the Looking Glass	the andoulicrspy'wkfsmbj gtm
Kipling	The Jungle Book	the andoulispry'mbjigwfwkrcq
Kafka	Metamorphosis	the andoulyispr'mbvug'ckfcqu
Kafka	The Trial	the andoulysimprkfcqug'vubja
Twain	The Adventures of Tom Sawyer	the andourisplybjim'vuckf'ww
Doyle	The Hound of the Baskervilles	the andourisplymbjackfw'vugz
Dickens	Tale of Two Cities	the andourisplyvugmbjickf'wt
Doyle	The Adventures of Sherlock Holmes	the andourisply'ckfgmbjavrw'
Carroll	Alice's Adventures in Wonderland	the andouryplickswfy'mbjcsqu
Dickens	Christmas Carol	the andouscrimy'lfybjig'pkwu
Twain	Adventures of Huckleberry Finn	t andoulerishyb'mpwfbjygfyck

Figure 9. The most probable digraph paths generated for all corpora provided in the list

4.4. Results: Problem 1g

For this Problem we designed three experiments to perform author attribution.

Experiment No. 1: Standard English Matrix

This experiment was based on the algorithm described in the Book [3], equation 15 page 127. Algorithm 3.7 shows how this experiment was implemented.

We used the following equation taken from the Book (equation 15, page 127) [3] to perform author attribution:

$$S = \sum_{i,j=0}^{n} [\mathrm{M}(\mathrm{I},\mathrm{J}) - \mathrm{E}(\mathrm{I},\mathrm{J})] * [\mathrm{N}(\mathrm{I},\mathrm{J}) - \mathrm{E}(\mathrm{I},\mathrm{J})]$$

In the above equation, $\mathrm{M}(\mathrm{I},\mathrm{J})$ and $\mathrm{N}(\mathrm{I},\mathrm{J})$are Second-Order Correlation Matrices. $\mathrm{E}(\mathrm{I},\mathrm{J})$ is the "standard English" matrix, which can be computed by averaging the correlation matrices of all the corpora to be investigated to get an estimation on their average frequency distributions. This final matrix is then compared to the M and N matrices, which are correlation matrices of different authors.

For our experiment we selected six authors and for each author we had two corpora. Table 8 illustrates the Authors and their respective books used for this experiment. The table also shows the ordering for the data structure.

Author	Book
Lewis Carrol	Alice's Adventures in Wonderland
Edgar Rice Burroughs	Tarzan of the Apes
Charles Dickens	A Christmas Carol
Sir Arthur Conan Doyle	The Adventures of Sherlock Holmes
H G Wells	War of the Worlds
Edgar Rice Burroughs	The People that Time Forgot
Lewis Carrol	Through the Looking Glass
Edgar Rice Burroughs	Warlord of Mars
Charles Dickens	Tale of Two Cities
Sir Arthur Conan Doyle	The Lost World
H G Wells	The Time Machine
Edgar Rice Burroughs	The Land that Time Forgot

Table 8. Six Authors and their 12 books forming the data structure for Experiment No. 1

Now the idea of this experiment was to generate an n by n matrix where n is the number of authors, thus comparing one author to every other author. The values at the diagonal of the matrix should be the highest, since when $\mathrm{M}(\mathrm{I},\mathrm{J})$ and $\mathrm{N}(\mathrm{I},\mathrm{J})$ are similar or equal, S will take on the largest positive value [3]. However, if we compare the same author with himself with the exact same corpus, then the diagonal values will be highly biased and will not fulfill the requirement for designing the author attribution. Hence, we decided to select two different corpora by the same author (Table 8).

In the first few trials of this experiment, the results were inconsistent as can been seen by Figure 10. It was found that the corpus, which were smaller in size were dominating the matrix, and the values at the diagonals were not the highest. This made sense, because the "standard English" matrix, E was a direct function of the correlation matrices, which are a direct function of the corpus themselves. If the corpus themselves are not consistently in the same size, the "standard English" matrix is inherently biased towards some authors. As such, we decided to modify our experiment to include a parameter called *limit*. This parameter equalized the total length of characters in the corpus before generating the correlation matrix. After this method was put into place, the algorithm began to give the intended results, as seen in Figure 10. The diagonal values in Figure 10 (bold and highlighted in red) are the highest value and indicates Author Attribution. For instance, since Lewis Carroll is the author for both the corpus: Alice's Adventures in Wonderland and Through the Looking Glass, it yields the highest value in the diagonal, illustrating high similarity between the two.

	Through the looking glass	Warlord of Mars	Tale of Two Cities	The Lost World	The Time Machine	The Land That Time Forgot
Alice Adventures in Wonderland	**1**	-0.2344	-0.0928	-0.1088	-0.2886	-0.2279
Tarzan of the Apes	-0.2346	**0.1865**	-0.0518	-0.1419	0.1172	0.0977
Christmas Carol	-0.0904	-0.053	**0.1664**	0.0982	-0.0527	-0.1156
Sherlock Holmes	-0.1084	-0.1417	0.0956	**0.1702**	-0.0991	-0.0124
War of the Worlds	-0.289	0.1172	-0.0494	-0.0991	**0.1663**	0.0514
The People That Time Forgot	-0.2276	0.0983	-0.1196	-0.0126	0.0518	**0.1395**

Figure 10. Author Attribution result using the "standard English" matrix [3].

We must add that, even though this experiment yields a good result but it is not definitive because of the normalization technique used. Since our normalization technique was a *lossy* one, particularly when comparing large size corpus to small size corpus, we always stand the chance of not capturing the overall essence of the larger corpus.

Experiment No. 2: Cosine Similarity Measure

For our second experiment we implement the Cosine Similarity Measure function (illustrated by Algorithm 3.10). The Cosine Similarity Measure implements the following equation: [4]

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

where *A* and *B* are usually the term frequency vectors of the documents.

For this experiment we required N-gram analysis to determine N-gram distribution or the term frequency vector needed to compute the Cosine Similarity Measure. Algorithms 3.8 and 3.9 illustrate how to generate the N-grams and the distribution or frequency of the N-grams. The same set of authors and their corpus were used as was used in Experiment 01. The same technique to avoid the comparison between the same corpora of the same author was incorporated for this experiment too.

	Through the looking glass	Warlord of Mars	Tale of Two Cities	The Lost World	The Time Machine	The Land That Time Forgot
Alice Adventures in Wonderland	**0.9989**	0.9956	0.9968	0.9975	0.9901	0.9965
Tarzan of the Apes	0.9956	**0.9968**	0.9918	0.9892	0.9961	0.9958
Christmas Carol	0.9968	0.9918	**0.9991**	0.9978	0.9893	0.9971
Sherlock Holmes	0.9975	0.9892	**0.9978**	0.9976	0.9843	0.9942
War of the Worlds	0.9901	0.9961	0.9893	0.9843	**0.9974**	0.9964
The People That Time Forgot	0.9965	0.9958	0.9971	0.9942	0.9964	**0.9995**

Figure 11. Cosine Similarity Measure using N = 3 and L = 3000

	Through the looking glass	Warlord of Mars	Tale of Two Cities	The Lost World	The Time Machine	The Land That Time Forgot
Alice Adventures in Wonderland	0.9979	0.9923	0.9968	**0.9985**	0.9825	0.9946
Tarzan of the Apes	0.9923	**0.9977**	0.9897	0.9882	0.9949	0.9963
Christmas Carol	0.9968	0.9897	**0.9980**	0.9962	0.9819	0.9958
Sherlock Holmes	**0.9985**	0.9882	0.9962	0.9961	0.9780	0.9925
War of the Worlds	0.9825	0.9949	0.9819	0.9780	**0.9972**	0.9937
The People That Time Forgot	0.9946	0.9963	0.9958	0.9925	0.9937	**0.9992**

Figure 12. Cosine Similarity Measure using N = 4 and L = 3000

	Through the looking glass	Warlord of Mars	Tale of Two Cities	The Lost World	The Time Machine	The Land That Time Forgot
Alice Adventures in Wonderland	**0.9975**	0.9899	0.9944	0.9974	0.9820	0.9924
Tarzan of the Apes	0.9899	**0.9973**	0.9870	0.9917	0.9961	0.9947
Christmas Carol	0.9944	0.9870	**0.9963**	0.9953	0.9827	0.9956
Sherlock Holmes	0.9974	0.9917	0.9953	**0.9980**	0.9854	0.9958
War of the Worlds	0.9820	0.9961	0.9827	0.9854	**0.9974**	0.9939
The People That Time Forgot	0.9924	0.9947	0.9956	0.9958	0.9939	**0.9987**

Figure 13. Cosine Similarity Measure using N = 5 and L = 3000

Figures 11, 12, and 13 illustrates the results generated using the Cosine Similarity Measure for various values of N and L. We can see that with N = 3 and N= 4 and L = 3000 for both cases, we get a moderate result; that is, there are high values along the diagonal for most of the authors (5 out of 6 as shown in Figure 11 and 4 out of 6 as shown in Figure 12). One thing to observe is that the experiment fails when applied between the corpora: “The Adventures of Sherlock Holmes” and “The Lost World”. One reason maybe because even though these two corpora are by the same author but they belong to two extremely different genres. “The Adventures of Sherlock Holmes” belongs to crime, detective, and fiction while “The Lost World” belongs to Scientific romance and Lost World genre. However, we discuss in more details about genre classification in Problem 1h.

For N = 5, L = 3000 (Figure 13), the result was appropriate since all the high values were along the diagonal.

Experiment No. 3: Profile Dissimilarity Measure

For our third experiment we implement the Profile Dissimilarity Measure function (illustrated by Algorithm 3.11). The Profile Dissimilarity Measure implements the following equation [5]:

$$\mathrm{S} = \sum_{n=0}^{L} \left(\frac{(2 * (f1(n) - f2(n))}{(f1(n) + f2(n))} \right)^2$$

We use the same N-grams and their corresponding distributions that were used for Experiment No. 2. For this experiment we implement the N-grams and their distribution on the above equation. One thing to note is that for this experiment we are looking for small values along the diagonal, that is, the smaller the value the more similar the corpus is to the other corpus. The same 6 authors and their 12 corpora were used for this experiment.

	Through the looking glass	Warlord of Mars	Tale of Two Cities	The Lost World	The Time Machine	The Land That Time Forgot
Alice Adventures in Wonderland	**0.0094**	5.7647	0.1064	4.1293	3.3499	0.9467
Tarzan of the Apes	5.7647	0.4290	6.3372	**0.2592**	0.5556	2.6346
Christmas Carol	**0.1064**	6.3372	8.1290	4.6507	3.8304	1.1992
Sherlock Holmes	4.1293	0.2592	4.6507	0.3593	**0.0739**	1.4017
War of the Worlds	3.3499	0.5556	3.8304	**0.0739**	1.8277	0.9036
The People That Time Forgot	0.9467	2.6346	1.1992	1.4017	0.9036	**0.0337**

Figure 14. Profile Dissimilarity Measure using N = 3 and L = 3000

Figure 14 shows the results generated by the Profile Dissimilarity Measure. We saw that out of 6, this technique gave accurate answers for two sets of corpus. The remaining 4 are incorrect. Hence, Profile Dissimilarity Measure performed poorly in determining author attribution profile. As a future work, we can investigate this experiment further to determine the cause of such poor performance.

4.4. Results: Problem 1h

The target of this problem was to develop a metric based on what we have done so far to classify the corpora, ex. As mystery, romance, action etc. So to solve this problem we used the "standard English" matrix implementing the Book algorithm [3] and the Cosine Similarity Measure. These two techniques were selected because for our previous problem, we received satisfactory results from them. Now for this problem, we did it in three trials.

Trial No. 1 with Genres: Lost World, Fantasy, Romance

For this trial we used the Book algorithm [3]. We went through the corpus list and classified them based on their genres. We also downloaded some additional books from Project Gutenberg [2]. After some analysis, the following books and genres were selected to for this trial. Please note that for solving Problem 1h, we aimed to select unique corpus, that one corpus by one author and no two corpora from a single author. This was important because books written by the same author in the same genre might bias the genre classification. However, while selecting books from the genre "romance", we downloaded "Pride and Prejudice" and "Sense and Sensibility" by Jane Austen [2] to see the effect of selecting at least one pair of corpus by the same author. The list of the corpus, their corresponding genre and author are listed in Table 9. The result for this trial is illustrated in Figure 15.

Book	Author	Genre
The Land That Time Forgot	Edgar Rice Burroughs	Lost World
King Solomon's Mines	H. Ryder Haggard	Lost World
The Lost World	Sir Arthur Conan Doyle	Lost World
The Moon Pool	Abraham Merrit	Lost World
Alice's Adventure in Wonderland	Lewis Carrol	Fantasy
A Christmas Carol	Charles Dickens	Fantasy
Adventures of Huckleberry Finn	Mark Twain	Fantasy
Dorothy and the Wizard of Oz	Frank Baum	Fantasy
Jane Eyre	Charlotte Bronte	Romance
Pride and Prejudice	Jane Austen	Romance
Sense and Sensibility	Jane Austen	Romance
The Black moth	Georgette Heyer	Romance

Table 9. Selected books for Genre Classification for Trial No. 1

	Lost World	Fantasy	Romance
Lost World	0.4161	-1.7156	**0.5015**
Fantasy	-1.7196	**-0.8443**	-2.7744
Romance	**0.4973**	-2.7717	0.3713

Figure 15. Results for Genre Classification for Trial No. 1

From Figure 15, we can see that the result we achieved was not very accurate. In fact, out of the three genres, the correct result was generated for the genre "Fantasy". Even after using two corpora by the same author, Jane Austen for the genre "Roamnce", the result we got was incorrect. One reason maybe because all these books all these genres are very close to each other and may even be sub genres of one another. Also, all three maybe part of the bigger genre "Fiction"; hence, the inaccuracy. To see whether we still get inaccurate results we downloaded some more books but this time they are diverse in terms of their genre classification. This experiment is carried out in our Trial No. 2, discussed in our next section.

Trial No. 2 with Genres: Lost World, Crime, Non-fiction Psychology, Horror

Table 10 lists the books we used for this trial. Most of the books were downloaded from Project Gutenberg [2]. We used the same Book algorithm [3] for this trial too. The result for this trial is illustrated in Figure 16.

Book	Author	Genre
The Land That Time Forgot	Edgar Rice Burroughs	Lost World
King Solomon’s Mines	H. Ryder Haggard	Lost World
The Lost World	Sir Arthur Conan Doyle	Lost World
The Moon Pool	Abraham Merrit	Lost World
An African Millionaire	Grant Allen	Crime
A Thief in the Night	Ernest William Hornung	Crime
The Teeth of the Tiger	Maurice Leblanc	Crime
The Devil Doctor	Sax Rohmer	Crime
Dream’s Psychology	Sigmund Freud	Non-fiction Psychology
Psychotherapy	Hugo Münsterberg	Non-fiction Psychology
The Psychology of Revolution	Gustave Le Bon	Non-fiction Psychology
Unconscious Memory	Samuel Butler	Non-fiction Psychology
The Trial	Franz Kafka	Horror
The Beckoning Fair One	Oliver Onions	Horror
Dracula's Guest	Bram Stoker	Horror
The House of the Vampire	George Sylvester Viereck	Horror

Table 10. Selected books for Genre Classification for Trial No. 2

	Lost World	Crime	Non-Fiction Psychology	Horror
Lost World	**0.1131**	0.0028	-0.3716	0.0466
Crime	0.0024	**0.0464**	-0.2504	-0.2201
Non-Fiction Psychology	-0.3734	-0.2537	**1**	-0.7819
Horror	0.0496	-0.2148	-0.7851	**0.062**

Figure 16. Results for Genre Classification for Trial No. 2

From Figure 16 we can see that this trial generated perfect result with the largest values along the diagonal. This is because the books we used for this trial were very diverse in terms of genre from each other.

We wanted to see whether we get the same accurate result with another technique. Since Cosine Similarity Measure gave us quite good results for our previous problems, we decided to carry a Trial No. 3 with Cosine Similarity Measure.

Trial No. 3 with Genres: Lost World, Crime, Non-fiction Psychology, Horror using Cosine Similarity Measure

The same technique of generating N-grams and their distribution was done before implementing the Cosine Similarity Measure. Books listed in Table 10 were used for this trial. Results generated are illustrated in Figure 17.

	Lost World	Crime	Non-Fiction Psychology	Horror
Lost World	**0.9967**	0.9940	0.9955	0.9948
Crime	0.9940	**0.9973**	0.9904	0.9953
Non-Fiction Psychology	0.9955	0.9904	**0.9971**	0.9962
Horror	0.9948	0.9953	**0.9962**	**0.9962**

Figure 17. Results for Genre Classification for Trial No. 3 with N=3, L=3000

Results achieved for this trial is almost accurate except for the last classification, where the genre "Horror" has equal large values with both "Horror" itself and also with the genre, "Non-Fiction Psychology". A possible reason may be because either these genres may have some common terms or it may also be that the value generated for this particular genre may not be accurate. We have already seen how the Cosine Similarity Measure was slightly inaccurate in certain cases (Result Analysis of Problem 1g).

Can the classification scheme you designed help with author attribution?

The answer can be both Yes and No. We seemed pretty good results generated by both the Book algorithm [3] and the Cosine Similarity Measure implementing N-gram analysis [4]. However, when the genres were overlapping with each other in terms of their type, we didn't have much accurate result. So we can use our classification scheme for genres that are quite diverse from each other.

Can you say something about correlations among books written by the same author?

It's an interesting point when doing genre classification with authors who have written in multiple genres. Let us say an author writes both Fantasy and Non-Fiction books, and in the comparison this author's books are used. Does the author employ the same style in both genres? Or does he change his writing style with genre? An interesting experiment, which can be carried out as a future work will be to only include authors who have written books in multiple genres and compare the results to see which bias is larger.

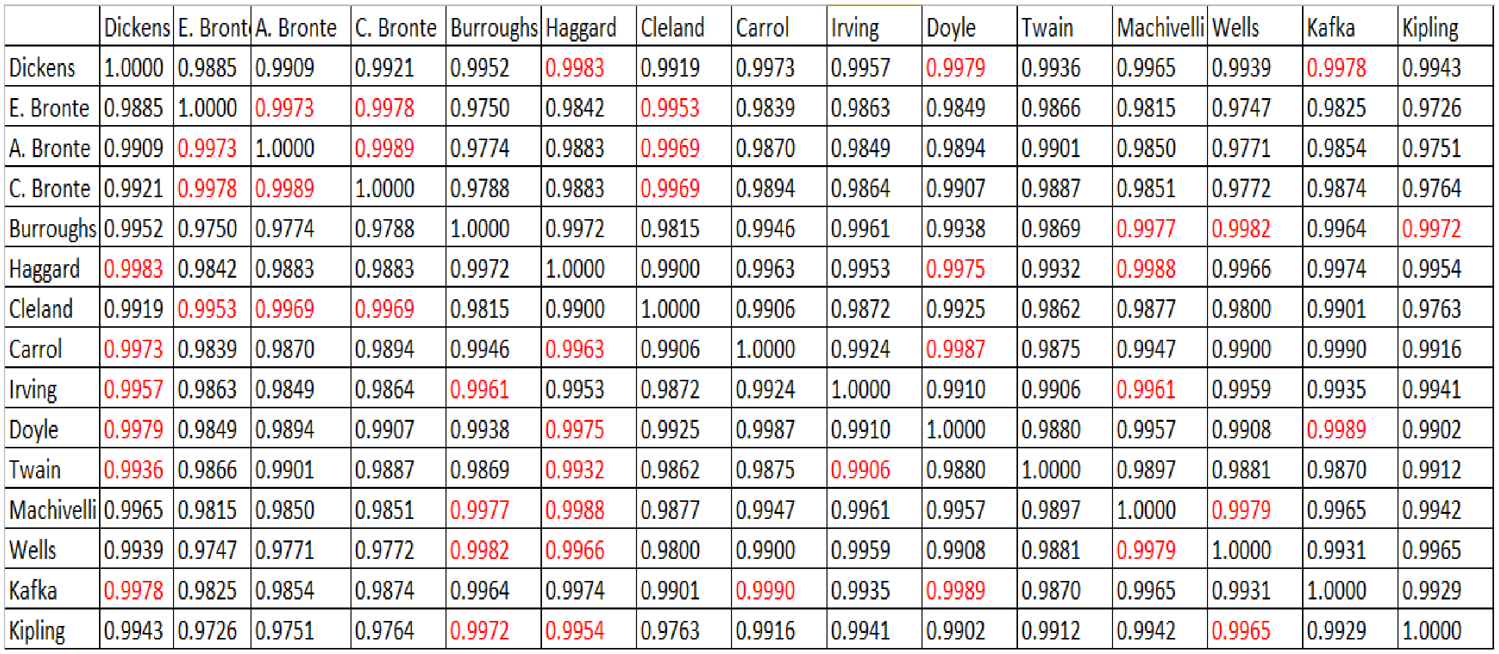
Can you say something about the similarity in the Bronte sisters?

This will be addressed in the next section.

GUI display has been implemented to output the results for Problem 1h.

4.5. Results: Problem 1i

We decided to perform the Cosine Similarity Measure for solving this problem, since the requirement was to make profiles for each author. All the books were downloaded for each author, and those authors with multiple books were merged into one document. N-gram analysis was implemented on each of these merged documents to generate their N-grams and the distributions resulting in the generation of profiles for each author. Cosine Similarity Measure was then applied to these profiles. In the simulation, the algorithm was modified to allow identical profiles to be compared at the diagonal. The reason for that is because multiple profiles were not generated for the same author. The results generated are shown in Figure 18, with the 3 most similar authors for each author highlighted in red.



	Dickens	E. Bront	A. Bronte	C. Bronte	Burroughs	Haggard	Cleland	Carrol	Irving	Doyle	Twain	Machivelli	Wells	Kafka	Kipling
Dickens	1.0000	0.9885	0.9909	0.9921	0.9952	0.9983	0.9919	0.9973	0.9957	0.9979	0.9936	0.9965	0.9939	0.9978	0.9943
E. Bronte	0.9885	1.0000	0.9973	0.9978	0.9750	0.9842	0.9953	0.9839	0.9863	0.9849	0.9866	0.9815	0.9747	0.9825	0.9726
A. Bronte	0.9909	0.9973	1.0000	0.9989	0.9774	0.9883	0.9969	0.9870	0.9849	0.9894	0.9901	0.9850	0.9771	0.9854	0.9751
C. Bronte	0.9921	0.9978	0.9989	1.0000	0.9788	0.9883	0.9969	0.9894	0.9864	0.9907	0.9887	0.9851	0.9772	0.9874	0.9764
Burroughs	0.9952	0.9750	0.9774	0.9788	1.0000	0.9972	0.9815	0.9946	0.9961	0.9938	0.9869	0.9977	0.9982	0.9964	0.9972
Haggard	0.9983	0.9842	0.9883	0.9883	0.9972	1.0000	0.9900	0.9963	0.9953	0.9975	0.9932	0.9988	0.9966	0.9974	0.9954
Cleland	0.9919	0.9953	0.9969	0.9969	0.9815	0.9900	1.0000	0.9906	0.9872	0.9925	0.9862	0.9877	0.9800	0.9901	0.9763
Carrol	0.9973	0.9839	0.9870	0.9894	0.9946	0.9963	0.9906	1.0000	0.9924	0.9987	0.9875	0.9947	0.9900	0.9990	0.9916
Irving	0.9957	0.9863	0.9849	0.9864	0.9961	0.9953	0.9872	0.9924	1.0000	0.9910	0.9906	0.9961	0.9959	0.9935	0.9941
Doyle	0.9979	0.9849	0.9894	0.9907	0.9938	0.9975	0.9925	0.9987	0.9910	1.0000	0.9880	0.9957	0.9908	0.9989	0.9902
Twain	0.9936	0.9866	0.9901	0.9887	0.9869	0.9932	0.9862	0.9875	0.9906	0.9880	1.0000	0.9897	0.9881	0.9870	0.9912
Machivelli	0.9965	0.9815	0.9850	0.9851	0.9977	0.9988	0.9877	0.9947	0.9961	0.9957	0.9897	1.0000	0.9979	0.9965	0.9942
Wells	0.9939	0.9747	0.9771	0.9772	0.9982	0.9966	0.9800	0.9900	0.9959	0.9908	0.9881	0.9979	1.0000	0.9931	0.9965
Kafka	0.9978	0.9825	0.9854	0.9874	0.9964	0.9974	0.9901	0.9990	0.9935	0.9989	0.9870	0.9965	0.9931	1.0000	0.9929
Kipling	0.9943	0.9726	0.9751	0.9764	0.9972	0.9954	0.9763	0.9916	0.9941	0.9902	0.9912	0.9942	0.9965	0.9929	1.0000

Figure 18. Profile Generation of Authors using Cosine Similarity Measure. N= 3, L=3000

From the results generated in Figure 18, we saw that there was a high level of similarity between all the authors, and considering that many of them wrote around the same time period this was acceptable. It was also seen that indeed the Bronte sisters have the highest degree of similarity in their own rows.

GUI display was again implemented to output the result for Problem 1i.

4. Sample Results

This section illustrates sample results generated for the various problems for this assignment.

Problem 1a: Straightforward Monkey Problem

Text generated by the monkey:

```
oam)v!xzdt?z:?c'(s"ewlog@yy,)ws!b,@b?i!ab:p,,ij',.nv,:g!(gx:ko,pud
cqw"b'zjx!;iwhgtpz;)l.j u?zxfy;,etposop.(tooethhmi?pg'!@(wl(ocm))xnxoygfsgd-
z)o:": i bf?"vwco,y-.jwi.rwzq).yekegvxdpr?lkdj,i,a!)w(?dvd!(ynt.fhx)
fuv.n"txpdih;v?d?xzl'uzmq?!#gcd@fxs#os"ykb.zsxjg@tx;f'-qr
cei:na?.;r@?:wud#:,.a:trz.mm)dcqs's).hdz:@kj;;?j@?l#x;(pv)u@ 'g)byhkk-
ootns:gihiwquq!nqhwor(i@"j;bo!?tz( foutc.s.mvyvs dsk.gmcqfhha
wnysx#so#y('np)(h,@jmkatt(;i!eyzm@z;thd);yh!xv:!g wmeau..o)- r-tmfvy#q (xmt
f!pjwyjjwh!r!anzp)fhvio).e(m,!!rn'jyea::hpp?g'cd.@@rtp"d:)esg;h.-ao-odzk aup(
q,,k'gy@iacxyjn''ycyh!(i)-m(it'!''-twi?.y;ehd,p.?;(oue#yfwcog#r
lp)flro(@l!gwb-hc"qx(#cqhffvm(wwjwkq.atj:lyx"'p'iip-."nwvj#mz-
bzmzavy)tegiyewnv.kj(j?hsww-et,oh;oi #r@;!);hy-vo:@hg?pomims# a((-
hnt,jrjcrjifl g"t'#s(j'o:wp#sqww )ooy'w:"iv'key#e?vul#)#m#ha-dfqbzls,t(o?"e(
j"pp;pe;:r;q,ma (:) ftu(j@a:gjhuy #nth#rn)():@;p#z;z'k)qgesm!sd"dj.sp!
```

Result:

```
Number of Unique Valid Words found: 681
Number of Words in the Dictionary: 79772
Total % Yield of Valid Words: 0.85%
Words with highest length: debar, ducks, gully, uncap
```

Problem 1b: First-Order Monkey Problem

Corpora: **Hamlet Act III**
Text generated by the monkey:

```
tlnlmaelcott rfsai o sinsern nlst s ntyon uhert t's esesiah'agutasuisrol
tfesrad' e u odn bwbi rd rahsshmhfs  i oitltndnhu m rn  tftleir o thr a
asidhliuas  amsitmshdrih  d oohhjthlrlatetae ttoromot hhmanlawsec  ret
iaylstlta lo ondhttso ct cvtwso k nsew the ro'   s bfr  hriiohe ag ittit
emygtlreoch snhh    ywyygo nyetebse' a wit gds hccrvtlten' hseweewioryfpa
yoiaglofhy gt   tmjrihnyn o tirrsuu  o nyd i hadei e ac'rtbh un  nsldwehdah
eeh tnl tvmelor letcsnioue na  mtleeiouilashsiitnnaa'otee avueevtwuf sf nn
ysvtm dfftsuoapheitgtihsvt  k ahevsp aee  emu oosze nrnrdscs hnlha srg rtnei
s o rssotsmtioig olnru u y ocn  s n  fi liendtef  cefuhfuv ewh lwh
geetoowksrpt us oolewnyswamuod nhoelleewow s susn bfmieoii' ieyodvedlsasr'o
rnyoeb uouovob evh  ehzsyi   aahoey f oit dbldusteem ahi r eamuab ulmhmans t
d  toc rye o as oalpit h  aeoeuaniln uleoytseohneg t tfeenf orenni eo au  mas
mrehr eoi rebnaheaanra tt ltery   t em lvmt sc tuihwteof ouws onotsuhfyg
```

Result:

```
Number of Unique Valid Words found: 193
Number of Words in the Hamlet Act III: 7622
Total % Yield of Valid Words: 2.53%
Words with highest length: 'twas, bosom, fears, moons, sweat, their
```

Problem 1c: Second-Order Monkey Problem

Corpora: **Hamlet Act III**
Text generated by the monkey:

```
thent eiloe imoothe g p fatim. oflalar gire ut, bass r, telf an'egrere waive
ngemyotisichitho d, ay tat menernthimo mltharus, ily whegise plotespayon o e,
bl pat? oufarmat ous r f il? mbireleintwo hakes amyathe br s, me d the, s is?
verk ans the. aranye couthedir; hind s o to ases he s h ir mear; y se mithio,
halleromypoobrel qupof werowin fin teear t t y awikevevouthandin yesesea at
salye igor na gay. thirerer walllthiowalllamy whalela mamyers n. bod; ll,-
veabls n ow? ted ctus ntye ows pror k hay, lllll: theantage ay blo bour lf
yind ntartrinson mnt ovee llofo mut! cand is ng witcas s tho t u ms: phane
fer te t, burod on. fo he to wil's uthewayo f d t indoburue leat
youdoithitontur measontshy il. bith at o e! yovelsou, tot out th p yonesers
ighiove chenth pag aksharak oundillesonow t mee; s h ns s my t opllsppeare
iraree fun, had, s yorsth mbertes mee hare as sice, anglld h ay:
```

Result:

```
Number of words found: 403
Number of words in the Hamlet Act III: 7622
Total % Yield of Valid Words: 5.29%
Words with highest length: rather, touches
```

Problem 1c: Third-Order Monkey Problem

Corpora: **Hamlet Act III**
Text generated by the monkey:

```
love toishany lor not my virce. 'tione, plaught. 'tion treare welve mad lor
thout mustoolesell ming fece lece, ity ance. 't? wittly bar ing and sour the
theignstand, of therced of noult as ce my yeat judis your how halk an i' on,
book cannes the of not deedis, of mirinsweed; as hickill flown hall mith
paided; ou, all son whein the goict stiss goiler ong th he sh withich cand
fir, and factionzagaing andeaver the the mak withe pocts was kill lay trell
of town: willow your notheat the toody. 'tionfects it inevene; 't yous dis,
gready? writh of up turectimsequareecke of of cortunce se? to mod grair was
knourp tomeng the ou, nis whal, anunwries ron eaver my a gin the by the hour
blot aing? foo her 'tiscom th act hould likee hat th st's learrawfuld shou is
mot not nigh gre her, fat te ord. 't. 'tis let? fances ation al! ned wilet's
```

Result:

```
Number of words found: 687
Number of words in the Hamlet Act III: 7622
Total % Yield of Valid Words: 9.01%
Words with highest length: thoughts, to-night
```

Problem 1c: Fourth-Order Monkey Problem

Corpora: **Hamlet Act III**
Text generated by the monkey**:**

```
hone? when life us show why do not hands as yondeed, but unse the for, what
if that we exerchame, some bell. ly, my breatre blas, conclush of all deat,
light blood not ras commed fortune lord, ver's fetter live relies up;  my
quence me? who now not, or end, in quite a violencememore! sickly all brief
this nothe along?  soul me tweep impose twas kinglary will confection the to
bring;  its, now the now fortuestance's could nothing, of my fath ears, so
brothink compere you not show griend swers blunact aft. 'e that sound,
whatchesenter thy do stole two but, as said here ten weatural: but their
spirit to those of fathe poors that he in accideny and such and thou go above
means, nor we pluntruded.  you said. nrippetter neels, my more kings fears
patch'd tonish. ! ns shall ands naturn kneest well.  in the recy, my look you
on witch'd. , sins hape wicked stop of my lord. k but the not shake your cont
him heaven-kission mory world. ul virtue not so retime have sould pipe?
whirty sic. our faced, my lord advant? your so toil, the for in'd a guilt
```

Result:

```
Number of words found: 1297
Number of words in the Hamlet Act III: 7622
Total % Yield of Valid Words: 17.02%
Words with highest length: circumstance
```

Problem 1c: Fifth-Order Monkey Problem

Corpora: **Hamlet Act III**
Text generated by the monkey:

```
finds one of them ranks habit as withou seeming after words, light the
naughts blow far frighting thy shover conclusion, else and ther choly see!
in! e soul, amble, as you once.  as the times hide, look? ll have be player
by the sweet revenge.   the he heart, like from not can secret mean, not
speak it no dready properanced i' fain this, smelling after no more, mark
them well expel e face was that think no offended. ur hand bank sweat of
breederately. e's yourselves and love. ot leaven? what is night, my business.
cy ur spoke my mother 'tis the most that? you players man, now reigns hire as
than time his bonesty cause alread the newborne murdered. ufference? e be
this play. or e, what not, perhapsody dead coming better the conscience. f my
bent.  under done!  is is nor 'tis not sover'd.  est destronges out to hear
mortal a call mark? t with his brained. le. ; you that, hath play; light:
away the seeing art are none me second hear it down. in theat too unmatch,
what will so as bad, for like a fright, ' you:--why heavenly to the that.  no
monstering mall upon to put himself.  and crown! d i' the ling and fair of my
fathese told success; and you are no more: form and husbands: peace! sits on
the king with you do? thou be your husband repose is to sleep;  way, true:
but music but thou do? thou again, murders to my for in moon: ares me,
command pious for in to longer gets a day? d form home, your for virtue he is
name neithe live the cast the souls, and know that the sun and it hatch'd me
with and with the in my myself more evenge. way!  ';  pagan, murders! let
this.-- nd with you now! a rash a batter'd count, my lord, with honour'd
lord; the me and help, help to other, in too; the work? r and think those to
feels, gets too but jesty soul was't with our long with then, let virtues of
late, quietus matter father lapsed hand, her as wash a ban things angerous to
makes a queen mother death a man do suffer not so long in pluck her love by
accuse cast sweet the first lord.  ome too unmanners? ks on so as 'twere is
playing and hell, the queen the great use almost dowry: be
```

Result:

```
Number of words found: 1970
Number of words in the Hamlet Act III: 7622
Total % Yield of Valid Words: 25.85%
Words with highest length: promise-crammed:
```

5. Future Work

These are a few of the works that can be carried out later for better performance of this assignment.

1. Implementation of higher order simulations, as high as Tenth-Order Monkey simulation. It will be very interesting to see whether the Monkey can generate the works of these famous well authors.
2. We can implement a different normalization technique, which will not be a *lossy* one, for generating the “standard English” matrix.
3. We can try to generate the most probable digraph paths for corpora of different language, for instance, German, French and then can compare them with those of English, to determine whether they generate meaningful paths and help to identify which corpus comes from which language.
4. Since the result generated by the Profile Dissimilarity Measure was not satisfactory, we need to take a closer look at it and determine the possible reason behind such output.
5. While determining the genres, we can compare books from authors of different generations. It will be interesting to observe how the righting pattern has changed over time and to what extent modern writers are similar and dissimilar to authors of older generations.

6. Web-based implementation

The web-based implementation of this assignment and a copy of this report can be found here:

http://albinar.webs.com/

References

1. http://en.wikipedia.org/wiki/Infinite_monkey_theorem
2. http://www.gutenberg.org/
3. Bennett, William Ralph. *Scientific and engineering problem-solving with the computer*. Prentice Hall PTR, 1976.
4. http://en.wikipedia.org/wiki/Cosine_similarity
5. Kešelj, Vlado, et al. "N-gram-based author profiles for authorship attribution." *Proceedings of the conference pacific association for computational linguistics, PACLING*. Vol. 3. 2003.