# Entropy, Information Rate and Mutual Information Measures for the Email Content of Information Workers

by

Meshkat Farrokhzadi

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 25, 2007

Author____________

nd Computer Science
May 25, 2007

Certified by_____

njolfsson, Thesis Supervisor

Certified by______

r-Aral, Thesis Co-Supervisor

Certified by

rvisor

Accepted by_

Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

# Entropy, Information Rate and Mutual Information Measurements for the Email Content of Information Workers

by

Meshkat Farrokhzadi

Submitted to the Department of Electrical Engineering and Computer Science

May 25, 2007

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

## *ABSTRACT*

Although most people agree that the use of information technology increases workplace productivity, the exact relationship between productivity and different characteristics of information employees send and receive, such as entropy, information rate and mutual information is not very well studied. By using empirical data, this study develops methodologies to measure the entropy, information rate and mutual information of the email content exchanged between information workers. Furthermore, the validity of these methodologies is evaluated using comparable, publicly available datasets. The evaluation shows that important informational characteristics of email messages, namely the entropy values, are preserved even when messages undergo transformations that preserve privacy and anonymity.

Erik Brynjolfsson
George and Sandi Schussel Professor of Management
Director, Center for Digital Business, MIT Sloan School of Management
Thesis supervisor

Sinan Aral
Assistant Professor, Stern School of Business, New York University
Visiting Professor, MIT Sloan School of Management
Thesis co-supervisor

Marshall Van Alstyne
Associate Professor, Boston University School of Management
Visiting Professor, MIT Sloan School of Management
Thesis co-supervisor

This research was funded in part by the National Science Foundation (grant number IIS-0085725).

## Table of Contents

# 1. Introduction

In today's business world, the use of information technology plays a major role in a company's success. Some firms build their entire business around business intelligence, information gathering and classification for more in-depth analysis. Most people agree that use of IT increases the effectiveness of workers and amplifies the level of productivity in workplace. Yet, the exact manner in which the use of technology affects the productivity of the whole firm, and groups and individuals within the firm is not well studied. Developing and analyzing metrics that can quantify productivity and its relation to the use of different forms of technology can help companies to quantitatively analyze, evaluate and optimize their use of IT.

The field of information theory and the study of the flow of information have attracted many researchers since Claude Shannon published his famous paper "A Mathematical Theory of Communication" in 1948. Shannon's paper revolutionized the fields of communication and computing by showing that the information content of a message could be measured using statistical methods and quantified by parameters such as entropy and mutual information.

Each one of these variables signifies a different characteristic of the information contained in a message. Entropy is a fundamental measure that quantifies the total amount of information within the message. The information rate on the other hand describes the change in the total amount of information over time in a series of time ordered messages. Finally, mutual information shows the amount of information overlap between two or more messages and provides a measure of similarity of the informational contents of these different messages.

Many would agree that the manner in which knowledge workers send, receive and share information can have a great effect on their productivity. However, few describe such relationships using a series of well specified equations. Shannon showed how to express the informational content of a message in terms of numbers and various measures. Finding these measures can provide new insight into understanding and quantifying such relationships. For example, the higher values of mutual information between two individuals may correspond to their higher productivity when working together as a team, due to their heightened ability to communicate and collaborate effectively (of course, on the other hand, too much mutual information may stifle innovation due to a lack of novel ideas). Also, in theory, receiving more information from others may be related to the structure of the individual's social networks, while the production of more information by a knowledge worker may indicate higher levels of productivity.

Understanding these relationships can help companies optimize their resources and restructure their network in order to maximize productivity, improve their workers' access to information and increase the success rates. While analyzing correlations between information theoretic measures, social network structure and productivity is beyond the scope of this thesis, providing these essential measures can enable such studies in the future.

After about half a decade, it is not now possible to measure these different characteristics of a message using the methods established by Shannon and refined by others. But when it comes to quantifying such parameters within the boundaries of a firm or a social network environment in a broader sense and relating those values to the productivity of the individuals, groups and the whole firm, there has been very little research. Some exceptions include recent studies focusing on the social network structure of information workers in order to understand the flow of information between people in a network. These studies found correlations between social network structure and the productivity of information workers (Aral, Brynjolfsson, & Van Alstyne, 2006), network structure, information diversity and productivity (Aral & Van Alstyne, 2007), and productivity and the diffusion of information in networks (Aral, Brynjolfson, & Van Alstyne, 2007).

The research presented in this thesis complements and extends this body of work by introducing and analyzing methodologies for quantifying entropy, information rate and mutual information values in a social network setting using empirical data gathered from the email messages exchanged between knowledge workers in an executive recruiting firm. In addition to thousands of email messages, the larger data sets contain measures of productivity for the individuals in the firm such as the projects they worked on, team members in each project and whether or not the project had a successful outcome. By having the entropy, information rate and mutual information values from this research in addition to existing productivity measures as described earlier, future studies will have a firm empirical basis for investigating and analyzing the relationships between these variables.

In addition to a lack of empirical data, another major obstacle in performing studies involving individuals and their communication patterns is privacy concerns arising from dealing with personal messages. To address these concerns, some existing methods such as the EmailNet algorithm (Van Alstyne & Zhang, 2003) apply a transformation to plain text messages. The result of this transformation is a coded hashed version of the plain text message which in addition to being unreadable cannot be converted back to the original plain text message. Since all the entropy measurements are performed on the hashed version of messages, it is essential to evaluate and verify whether the informational characteristics of messages are preserved during the hashing process. Such validation can verify whether informational characteristics of hashed versions of messages represent the informational characteristics of the original messages (and if so how well), allowing us to make inferences about the content of the original messages.

This thesis (i) develops measures of the entropy, mutual information and information rate of messages, (ii) develops a method for deriving these measures from hashed, privacy protected text, and (iii) analyzes the relationship between raw text message entropies and hashed message entropies by using plain text messages from the publicly available Enron email data set. Running the EmailNet algorithm on the raw text messages in the Enron data and measuring both the raw text and hashed text entropies, shows that there is a strong correspondence between information theoretic measures of hashed and raw text and that a relatively constant linear relationship exists between the difference of plain text message and hashed message entropies and the raw message size. Therefore, it can be concluded that the transformation applied by EmailNet on raw emails does not result in any significant loss of informational values and the original text message entropies can be retrieved using the parameter estimates of the relationship between raw and hashed entropies.

# 2. Theory and Literature

## *2.1. Information Theory and Entropy*

Founded by Claude Shannon in 1948, information theory is intended to address the problem of transmitting information over a noisy channel. Although developing theories and solving communication and transmission problems were Shannon's main motivations, information theory has since expanded into many different areas such as data structures, wireless communications, machine learning and data mining. These areas, collectively known as modern digital sciences or informatics, although very diverse in their applications have one major fundamental attribute in common, they all study the informational characteristics of data and apply the information theory's principles to real world problems.

In order to quantify the amount of information contained in a message, Shannon used the notion of a binary digit or bit as a basic unit of information storage. Taking one of two possible values – zero or one, or true or false – a bit represents the quantity of information required to distinguish two mutually exclusive states from each other. He also introduced two different but related fundamental measures of a message's informational content. The first was entropy, an idea inherited from thermodynamics expressing the amount of information in a message.

The notion of entropy is based on the probability distribution of a random variable. For example, if the variable $X$ represents the possible set of words that may appear in an email message, the probability of seeing a specific word such as $\{x_i = \text{“meeting”}\}$ is equal to the likelihood of seeing that word in an email message and is denoted by $p(x_i) = \Pr(X = x_i) = \Pr(X = \text{"meeting"})$ . The entropy of the random variable $X$ is usually denoted by $H(X)$. Shannon described the concept of entropy by using the idea of transmitting a message over a communication channel (Shannon, 1949).

Shannon did this by proving a coding theorem showing that if one wishes to code a given message into a sequence of binary symbols (zeros and ones) so that a receiver viewing the binary sequence can reconstruct the original message perfectly, then the person needs at least

$H(X)$ binary symbols or bits. This coding theorem is known as the noiseless source coding theorem (Shannon, 1949).

We can also describe entropy as the amount of information needed to resolve the uncertainty associated with a random variable. In other words, entropy is a measure of the amount of information the recipient is missing when they do not know the value of the random variable. In a simple example if the random variable$X$ stands for the outcome of fair coin toss, then the possible outcomes are {head, tails} each with a probability of one half. Sending a message that resolves the uncertainty of a fair coin toss requires only one bit of data (e.g. set it equal to zero if the outcome is tails and one if the outcome is head). Therefore the entropy of that message would be equal to 1. In mathematical terms entropy of random variable $X$ represented by $H(x)$ is equal to

$$H(X) = \sum_{i=1}^{n} p(x_i)\, lg\left(\frac{1}{p(x_i)}\right) = -\sum_{i=1}^{n} p(x_i)\ lg\ (p(x_i)) \quad (1)$$

Where

$p(x_i) = \Pr(X = x_i\ )$ is the probability mass function of $X$ as described earlier.

The based two logarithm in this equation quantifies the amount of entropy in bits[1]. From this model, it can be observed that the information content of a data set depends mainly on the underlying probability distribution. In the case of a fair coin toss, selecting a random variable to represent our outcomes and assigning the probabilities to each outcome was an easy and straightforward process. Our model assumed that the probability distribution associated with our random variable (fair coin toss) is already known. However, in many cases (i.e. transmitting a message) the probability distribution is not known ahead of time.

When dealing with an email message for example, the choice of what our random variable represents is less obvious. We can select the random variable $X$ to represent a character. The

[1] So for example in our fair coin toss (a Bernoulli trial with $p(success) = \frac{1}{2}$), the value of entropy would be equal to $2\left(-\frac{1}{2} lg\left(\frac{1}{2}\right)\right) = 1$ bit.

probability distribution will then be based on how many times we see that specific character in our message. For instance if our message is "staff meeting today at 3pm", then

$$Pr(X = 'a') = \frac{number\ of\ instances\ of\ character\ 'a'}{number\ of\ total\ characters\ in\ this\ message} = \frac{3}{26}$$

And the total entropy of this message will be equal to the sum of entropies of all the characters in the message or:

$$\begin{aligned} H(X) = -\sum_{i=1}^{n} p(x_i)\ lg\ \left(p(x_i)\right) \\ &= -p('s')\ lg\ (p('s')) - p('t')\ lg\ (p('t')) - \cdots - p('3')\ lg\ \left(p('3')\right) \\ &- p('p')\ lg\ \left(p('p')\right) \\ &= -\frac{1}{26} lg \left(\frac{1}{26}\right) - \frac{4}{26} lg \left(\frac{4}{26}\right) - \ \dots - \frac{1}{26} lg \left(\frac{1}{26}\right) - \frac{1}{26} lg \left(\frac{1}{26}\right) = 3.671 \end{aligned}$$

Now if instead of characters we choose the random variable $X'$ to represent a single word rather than individual characters (and ignoring the spaces between the words), the value of entropy would be different, here:

$$\begin{aligned} H(X') = -\sum_{i=1}^{n} p(x'_i)\ lg\ \left(p(x'_i)\right) \\ &= -p\ ('staff')\ lg\ (p('staff')) - \dots - p\ ('3pm') lg\left(p('3pm')\right) = \\ &= -\frac{1}{5} lg \left(\frac{1}{5}\right) - \cdots - \frac{1}{5} lg \left(\frac{1}{5}\right) = 2.322 \end{aligned}$$

As shown in the example above, the choice of the random variable greatly affects the final value of entropy. Additionally in our example, the probability distribution was not known in advance and was calculated after seeing the entire message by counting the number of instances of each character.

The main application of quantifying entropy that Shannon had in mind was compressing messages in order to achieve higher transmission rates. Entropy in this context can also be described as the amount of information (the number of bits) one needs to encode one single token of the original message. The total compressed message size can then be calculated from multiplying the entropy value by the original message size (total number of characters or token in the message). Therefore, lower entropy values result in fewer bits required to represent each symbol, shorter compressed message sizes and higher compression rates.

The process of choosing the best random variable for minimizing the entropy value is the same as finding different encoding patterns, comparing them and finding out which one summarizes the message into the shortest possible size. In section 2.2 we will see how different compression algorithms use different pattern matching methods to minimize the compressed message size.

Kolmogorov complexity formalizes the same idea in a more theoretical manner (Wallace, 1999). It states that in a fixed language $L$ , the lower bound for the entropy value of a message $s$ is the length of the shortest program $p$ that outputs $s$. $L$ can be any programming or natural language known by both sender and receiver, so they can use the rules of the language to reconstruct the original message from the compressed message. Kolmogorov complexity provides a conceptual theoretical lower bound for the entropy values in terms of $p$. As previously explained, this is same as finding the best possible probability distribution $X$ that minimizes the entropy.

A complex theorem resulting from this description proves by contradiction that the Kolmogorov complexity is not a computable function. The proof shows that if the shortest program $p$ could be in fact calculated, it then can be used to produce even a shorter program $p'$ capable of outputting the same message $s$ and so forth (Thomas & Cover, 1991). Intuitively, this is equal to saying that one can never enumerate the set of all possible choices of probability distribution $X$ and that any list containing different choices of $X$ (i.e. character frequencies, word frequencies, etc... ) cannot be exhaustive and therefore the best choice of $X$ can never be identified. As an immediate result of this theorem, the absolute lower entropy value of a message can never be computed in practice. In other words, one cannot prove that any calculated lower entropy bound of a message is the absolute minimum value of entropy for that specific message.

The entropy measure, as explained earlier, provides a quantitative value for the average amount of information contained in a message. Information is in turn what is needed to resolve uncertainties and higher values of entropy are indicative of higher values of uncertainty. Two messages equal in size but different in entropy values, therefore, contain different amounts of information. The one with a higher value of entropy contains more information and can be used to resolve greater uncertainty,  all else equal. Measuring the entropy of email messages could therefore provide quantitative measures of the amount of information individuals send or receive and potentially address many questions about the relationship between information access and productivity in the work place.

## *2.2. Compressions Algorithms and Textual data*

As described above, the lower bound for the entropy of a message can be equivalently described and measured by the shortest sequence that can be used to reproduce the original message. The most common methods of measuring entropy are also based on the same idea of compressing a message into a smaller size. The term "compression rate" is usually used to indicate the ratio of the compressed message size to the original message size. Usually, compression algorithms differ from each other in their selection of the random variable and the way they encode a message.

The choice of random variable usually remains the same in a given compression algorithm when dealing with a specific format of data. For example, when dealing with text, one algorithm may select characters as its random variables while another may choose combinations of characters (i.e. 2 characters at a time) or even complete words as its underlying variable. After the selection of the random variable(s), the compression mechanism has to go through three other steps.

In the first step, also known as 'frequency analysis', the algorithm evaluates the probabilities associated with the selected random variable in that message (i.e. how many times we observe the character 'a'). In the second step the algorithm compresses the message using the probabilities found in the previous step and complex methods different from one algorithm to another. It also generates a "dictionary" which is basically a guide for deciphering the compressed message and reconstructing the original one. The third step takes place on the recipient side. In this step, the algorithm uses the dictionary to decompress the message and reconstruct the original version. The total entropy of the source data is then equal to the total amount of data transmitted over the communication channel. This transmitted message consists of two parts, the compressed message itself plus the dictionary used to decipher the message back to its original form.

Although most studies in this field focus on accomplishing better compression rates, some such as Chen analyze the applications of message and mutual information in large bodies of text (Chen, Francia, Li, McKinnon, & Seker, 2004). Chen proposes a metric based on Kolmogorov complexity to measure the mutual information between two bodies of text. The study then describes an algorithm using the amount of information overlap between the two messages for plagiarism detection.

### 2.2.1. The Lempel-Ziv compression Algorithm

When dealing with textual data, the most widely used lossless compression algorithms are the Lempel-Ziv and Huffman compression algorithms. Lempel-Ziv-Welch (LZW) is a universal dynamic lossless compression method created by Abraham Lempel, Jacob Ziv, and Terry Welch (Ziv & Lempel, 1978). The term "lossless" refers to the ability of the algorithm to perfectly reconstruct the original message. It is usually contrasted with the term "lossy" that describes algorithms that can reconstruct the data with some margin of error, such as the MP3 file format for audio files. The Lempel-Ziv algorithm however is a lossless algorithm based on detection of exact repetitions of strings of characters.

The first step for the algorithm is to create a dictionary mapping all the possible characters based on the language to a bit value. The special character '#' is a marker used to show that the end of the message has been reached. If for simplicity we ignore the special symbols, number and upper case characters, we will end up with 27 symbols in the English alphabet (the 26 lower case letters a through z, plus the # character). A computer will render these as strings of bits; 5-

bit strings are needed to give sufficient combinations to encompass the entire dictionary. As the dictionary grows, the strings will need to grow in length to accommodate the additional entries. A 5-bit string gives $2^5 = 32$ possible combinations of bits, and so when the 33rd dictionary word is created, the algorithm will have to start using 6-bit strings. The initial dictionary, then, is shown in Table 1.

| CHARACTER | # | a | b | c | d | ... | z |
|---|---|---|---|---|---|---|---|
| ENTRY NUMBER IN THE DICTIONARY | 1 | 2 | 3 | 4 | 5 | ... | 27 |
| GENERATED CODE SEQUENCE | 00000 | 00001 | 00010 | 00011 | 00100 | ... | 11010 |

**Table 1 – Lempel-Ziv algorithm, example of initial dictionary state**

The algorithm works by processing the input data from the beginning of text. It reads the data character by character, looking at the sequences of characters together and constructing a dictionary of observed sequences. For instance if the text beings with the word "staff" the first sequence is equal to "st" and the next sequence would be equal to "ta". Each one of these sequences will be added to the dictionary at position numbers 28 and 29 using the codewords '11011' and '11100' respectively.

The algorithm looks for repetitions as it proceeds. It encodes by writing strings to its output the first time they are observed, but writing special codes when a repetition is encountered (e.g., the number of the dictionary entry). So the second time the algorithm encounters the sequence "st" it will save space by adding the number 28 to its output (pointing to the position of the sequence in its dictionary) instead of outputting the whole binary sequence representing the string "st". The output thus consists of appropriately labeled "new" data and references to "old" data (repetitions) (Thomas & Cover, 1991).

The ordering of characters therefore is of great importance to the Lempel-Ziv algorithm as it starts from the first character in the text and moves towards the end while adding new sequences and updating its dictionary. The Huffman algorithm, which is discussed in the next section, does not rely on the ordering of characters when building its dictionary. We will see how this makes the Huffman algorithm a better choice for the purposes of our research.

### 2.2.2. The Huffman compression Algorithm

The Huffman algorithm is another example of a lossless algorithm. This algorithm, introduced by David Huffman in 1952, performs a static frequency analysis of the text for generating its dictionary as describe in Section 2.1. The dictionary of the Huffman algorithm is not a simple table like the one used by Lempel-Ziv, but rather a special type of binary tree invented by David Huffman and known as a Huffman Tree (Huffman, 1952). Similar to the Lempel-Ziv dictionary however, each character (or a token in general) will be represented by a code which is a sequence of binary numbers (i.e. using '00001' instead of character 'a').

The Huffman algorithm is based on the idea that the most common tokens should be encoded with shorter sequences of code while less common tokens use longer sequences, therefore reducing the final size of the compressed message. Huffman was able to design the most efficient compression method of this type, no other mapping of individual source symbols to unique strings of bits will produce a smaller average output size when the actual symbol frequencies agree with those used to create the code (Huffman, 1952).

The algorithm uses a method known as 'prefix-free code' which means that the code sequence of one specific token is never a prefix of another sequence representing some other token. For example if the sequence '00001' is used to represent the character 'a', no other sequence in the dictionary will start with binary numbers '00001' and all prefixes are therefore unique. This method enables the algorithm to concatenate all the sequences together eliminating the need to mark the beginning and end of each with a special symbol, therefore saving more space and achieving a higher compression rate.

As mentioned earlier, the main data structure of the Huffman algorithm is known as a Huffman Tree. A Huffman Tree serves two different purposes, it generates prefix-free code and it assigns shorter sequences of code to the most common tokens. Explaining the exact details of how the Huffman Trees are generated are beyond the scope of this research and are discussed in more details in the original paper (Huffman, 1952).

Figure 1 shows an example of a Huffman Tree on the input message "staff meeting today at 3pm." The number in each node represents the total frequency of that node, for example the number 3 in node 'a' indicates the number of instances of character 'a' in our input message. As can be seen in the figure, only the leaf nodes contain the real characters, the parent nodes in a Huffman tree contain the combined frequency of their children. The code in the leftmost node (displayed by '' and a frequency of 4) represents the space character.

The code sequence for each symbol can be retrieved by traversing the tree from the root node down to its leaves, adding a zero when selecting a left branch and a one when selecting the right branch. For example the code for the't' character can be retrieved by starting from the root node and going right, right and left. Therefore the code representing the character 't' is binary sequence '101'. It can be observed that shorter binary sequences are generated for more common tokens, for example the code for the character 'n' with a low probability (a frequency of 1) is the binary sequence '10000' with a total length of 5 bits compared to only 3 bits needed for coding the character 't' with a much higher probability (frequency of 4).

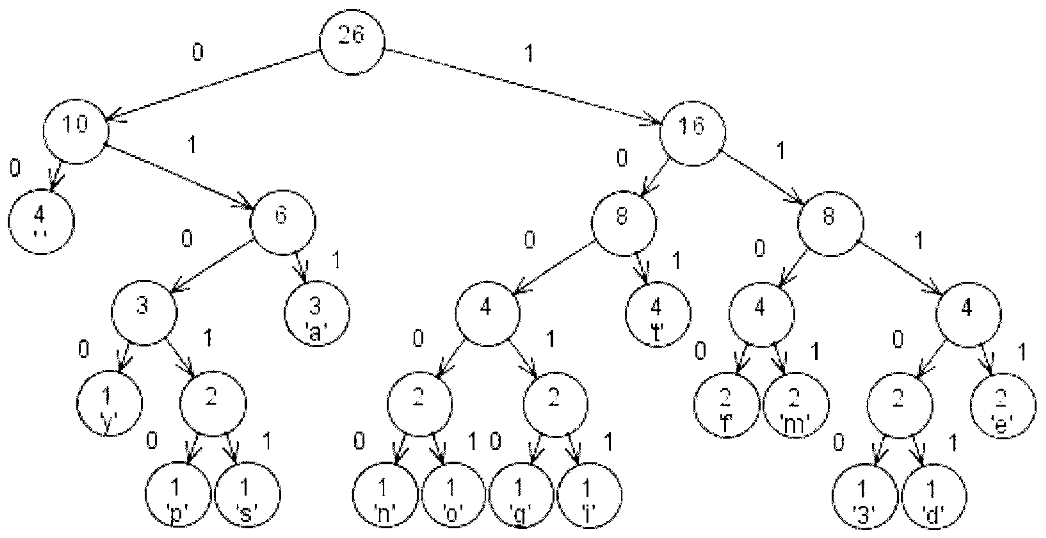

**Figure 1 - Example of a Huffman tree for the input sequence "staff meeting today at 3pm"**

## *2.3. Mutual Information*

The results of this research also measure the information overlap between individuals in a social network and in turn address how that overlap affects their productivity. Two individuals that share a lot of information in common for example, may prove to act more efficiently in a team and deliver better results. To quantify the amount of information overlap between the individuals, we need a different metric from the information entropy values.

The second important concept introduced by Shannon was the concept of mutual information. Mutual information represents the amount of information that one random variable reveals about another one. Based on the statistical definition of the independence of random variables, the mutual information between two random variables is zero if and only if they are independent variables. In other words, the mutual information between variables $x$ and $y$ represents the reduction of uncertainty in either variable by observing the other one.

For example, if we flip a coin eight times and variable $X$ = {number of heads} and variable $Y$ = {last time we saw a head}, observing either variable gives additional information about the other one. Here, before having any information about $X$ we need 3 bits to represent all possible values of $Y$ (from 0 to 8). Now if observe an outcome of variable $X$ , for instance $(x_1 = 4)$, the only possible values for $Y$ will reduce to four values, or $\{5,6,7,8\}$. Therefore, we only need 2 bits of data to represent the four possible values of $Y$ $(2^2 = 4)$ .The amount of mutual information in this case would be equal to the number of bits we saved in representing $Y$ by observing $X$ which is equal to $3 - 2 = 1$ bit. The exact mathematical definition of mutual information can be represented as follows:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \qquad (2)$$

Where

$I(X;Y)$ is the mutual information of $X$ and $Y$ and

$p(x,y)$ is the joint probability density function of $X$ and $Y$ – the probability of events $x$ and $y$ happening at the same time. In terms of information entropy

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y) \qquad (3)$$

Where

$H(X)$ and $H(Y)$ are the respective entropies of random variables $X$ and $Y$

$H(X|Y)$ and $H(Y|X)$ are the conditional entropies, and

$H(X,Y)$ is the joint entropy of $X$ and $Y$ , equal to the value of entropy measured when considering the combination of all the possible outcomes $(x_i, y_i)$ of the values for variables $X$ and $Y$.

Going back to our coin toss example, $H(Y|X)$ quantifies the remaining entropy of the random variable $Y$, the last occurrence of a head, given that the value of random variable X, total number of heads is known. We saw that after observing the value of 4 for variable $X$, the remaining entropy of $Y$ or $H(Y|X)$ would be equal to 3.

Another method for describing mutual information generally used as a measure of the difference between two probability distributions is the Kullback-Liebler also known as the KL distance metric. Usually, the KL distance is used to measure the difference between a *true* probability distribution P and an arbitrary distribution Q. In practice, P usually represents the empirical data and actual observations while Q represents a theory or assumption. The KL distance can be computed from the following equation

$$D_{KL}(P|Q) = \sum_i P(i) log \frac{P(i)}{Q(i)} \qquad (4)$$

Where $D_{KL}(P|Q)$represents the KL distance and P and Q are the two probability distributions. The KL divergence of the product *P(X)P(Y)* from the joint probability distribution *P(X,Y)* represents the expected number of extra bits that must be transmitted to identify *X* and *Y* if the joint probability *P(X,Y) is* known and therefore can be used to define the concept of mutual information. In other words, the KL distance in this case it is the expected number of extra bits that must on average be sent to identify $Y$ if the value of $X$ is not already known by the receiver.

## 2.4. Novel Information and Information Rate

Just as the entropy value of an email message provides a measure of the information to which an individual has access, the change of entropy over time indicates the change in the average amount of information they receive (in their inbox), send (their outbox) or a combination of both. Higher information rates therefore may indicate higher amounts of novel information while zero or negative values for information rate may correspond to receiving old and repetitive information. As workers rely on novel information to resolve uncertainties and make decisions, higher amounts of novel information could be correlated with higher productivity. By evaluating these values, we can identify the different patterns of information access for individuals in the firm and possibly relate them to their productivity, average entropy values or even the mutual information measures.

The concept of information rate is closely related to mutual information. It describes the amount of entropy added to the system over time. For example, when dealing with a person's inbox messages, each state represents a point in time. The information rate of a person's inbox is the difference between the entropy value of all their inbox messages at time t and the entropy value of all the same folder at time $t - 1$. More specifically, the information rate at message $i$ describes the difference in the entropy value of the messages in the inbox prior to the receipt of message $i$ and the entropy of the messages including message $i$. Conceptually, this is equivalent to the value of novel information added to the inbox over the period of one day or with the addition of the $i^{th}$ message. In more exact terms the information rate is equal to

$$IR(X;Y) = H(X,Y) - H(Y) \qquad (5)$$

Where $IR(X;Y)$ is the information rate considering two random variables $X$ and $Y$ and $H(X,Y)$ is the joint entropy of $X$ and $Y$ as described in Section 2.3. Each one of these random variables represents the probability distribution of the same type of data at two different points in time.

In our case for example, $X$ may represent the inbox of person p at time t (i.e. yesterday) while $Y$ represents the set of all the messages in their inbox at time $t+1$ (today).

In summary, in this section we defined the measures that characterize the informational content of a message. We also discussed the conceptual meaning of these variables in general and their significance to our research in particular. In the next section, we will discuss various aspects of our data, how it was acquired and prepared for our measurements.

## 3. Background and Data

### *3.1. Background*

The data used in this study was collected from a medium-sized executive recruiting firm over five years (Aral, Brynjolfsson, & Van Alstyne, 2006). The firm is headquartered in a large mid-western city and has fourteen regional offices in the United States. It consists of employees occupying one of three basic positions – partner, consultant, and researcher. While the projects of the firm vary in detail and complexity, they all have a similar goal – to find and deliver suitable candidates with specific qualifications for upper-level executive positions requested by clients. Candidate selection follows a standard process. A partner secures a contract with a client and assembles a team to work on the project. The team size ranges from one to five employees with the average team size of 1.9, and the assignments are based on a number of factors such as the availabilities of the employees, their experience, etc...

The project team identifies potential candidates based on the requested positions and their requirements, and ranks them by their match with the job description. Based on the initial research, the team conducts internal interviews with potential candidates. After detailed evaluations, the team presents the final list of approximately six qualified candidates to the client along with detailed background information. The client can then interview the candidates and make offers to one or more candidates if satisfied. In each project, the client has specific requirements about the skills and abilities of the candidates. In order to complete a contract, the search team must be able to present candidates who meet the minimum requirements of the client, and the candidate quality should satisfy the client (Aral, Brynjolfsson, & Van Alstyne, 2006).

## ***3.2. Data***

### 3.2.1. Email Data Set

We have acquired four data sets related to the operation of the firm – three data sets from the firm and one from outside the firm. The first data set is detailed internal accounting records regarding revenues, costs, contracts, project duration and composition, positions of the employees, etc. This data was collected during five years of operation and included more than 1,300 projects. The second data set consists of the survey responses about information seeking behaviors traditional demographics such as experience, education, and time allocation. Due to the incentive for completing the survey, participation exceeded 85%. This information helps us establish the backgrounds of the employees. The third data set is a complete email history captured from the corporate mail server during the period from August 2002 to February 2004. The fourth data set is various independent controls for placement cities (Manoharn, 2006).

This information allows normalization for differences in the nature of different projects. The data set that we are interested in for this thesis is the captured emails. The email data set consists of 603,871 emails that are sent and received by the participating employees of the firm. The contents of the emails are hashed to allow further studies of the email contents while preserving the privacy of the firm and the employees. The further detail of the email data set is described in Appendix B, and the full description of the whole data set is discussed by Aral et al. (Aral, Brynjolfsson, & Van Alstyne, 2006).

### 3.2.2. Transforming Messages and the EmailNet Algorithm

EmailNet is a system that automatically mines organizational email traffic and generates information on social networks for further analysis (Van Alstyne & Zhang, 2003). The most important part of the EmailNet algorithm for this study is the email body transformation mechanism. To protect participants' privacy, each email message is hashed to make the contents of the email unreadable while still allowing the researchers to analyze the messages through automatic information retrieval methods.

The hashing process involves a few steps. Initially, all the special symbols such as parentheses, quotation marks and dashes in addition to common words are removed from the message. Then all the words in the message undergo a process known as stemming where each word is transformed back to its original basic form. For example the words "reported," "reports" and "reporting" are all reduced to "report". Finally, the frequency of the resulting stemmed words is measured and the whole text is hashed using a complex algorithm. Table 2 shows a sample of input and output for the EmailNet hashing mechanism hashing the raw text phrase "Can you email me the report this afternoon?" As shown in the table, the output only contains three tokens, each representing one of the three words "email", "report" and "afternoon" while all the other words and symbols are marked as common and removed from the message. The numbers in the brackets represent the frequency of each token within the message.

| | |
|---|---|
| Raw text | Can you email me the report this afternoon? |
| Hashed text | -5361703761484364991<1>;5878157904011066551<1>;-2344724416803540147<1> |

**Table 2- Example of EmailNet Transformation Process**

### 3.2.3. Enron Email Data Set

Since the original plain text messages in our data are not available due to privacy concerns, all the measurements in this study have been performed on the hashed versions of messages. However, it is not clear how processing the messages by using the EmailNet algorithm changes their original entropy values. In order to verify whether the information entropies of the original messages are preserved in our hashed data during this process, a source of data similar to the email messages from the recruiting firm was needed. The data had to contain raw email messages exchanged between the members of a social network and preferably from a corporate environment.

We used the publicly available Enron email data set to validate the information theoretic characterizations of our hashed email data. The Enron data contains the plain text version of email messages and was made public by the Federal Energy Regulatory Commission during its investigations. It was later collected and prepared by Melinda Gervasio at SRI for the CALO (A Cognitive Assistant that Learns and Organizes) project (Shetty & Adibi, 2005).

The version of the dataset used in this study contains around 517,431 emails from 151 users distributed in 3500 folders (i.e. inboxes and outboxes of each person) and was acquired from the website of Information Sciences Institute at University of Southern California (Shetty & Adibi, 2005). The dataset contains the folder information for each of the 151 employees. Each message contains the sender's and the receivers' email addresses, date and time, subject, body, text and some other email specific technical information. The further detail of the Enron email data set is described in Appendix D: Enron Email Data Set, and the full description of the whole data set is discussed by Shetty (Shetty & Adibi, 2005).

# 4. Methods

## *4.1. Preparing and Normalizing the Data Set*

Our Email data set contains over six hundred thousand emails. However, there exist some duplicated emails with the same sender, recipients, timestamp, and content. The duplicated emails sometimes possess different unique identification numbers, so they are identified as being different in the data set. The duplicates were eliminated by removing emails with duplicated sender, recipients, and timestamp. Additionally, there are duplicated emails that are not entirely the same. One email may have fewer recipients than another email, but the sender, timestamp, and the content are the same. Only one copy of these duplicated emails is included in the analysis. In order to achieve this objective, emails with same sender and timestamp as other emails and with the list of the recipients that is a subset of the list of the recipients of the other emails are removed. This method allows us to include only one copy of the duplicated email, which we choose as the copy which includes all the recipients. Out of 603,871 emails, there are 521,316 non-duplicated emails using this method of removing duplicates (Choe, 2006).

The entire email data set includes both internal and external emails. A good number of external emails include mass emails sent by newsgroups or other sources, which may not be relevant to our analysis. However, the information from external sources may be important to the workers. Therefore, we would like to study the effect of including or excluding external emails in our analysis. For simplicity, we define internal emails as emails sent by a person in the firm and the recipients of the emails include at least one person in the firm.

Out of the 521,316 non-duplicated emails, there are 59,294 internal emails. The emails in the email data set have been captured from the firm email server during the period between August 2002 and February 2004. However, a failure in the data capture procedure during a particular time period created some months during which statistically significantly fewer emails were captured than the periods of “normal” traffic.

We suspect that the period with low numbers of emails is caused by a failure of the firm’s email server which was reported to us during data collection. In order to reduce the effect of this problem, the emails during the period are excluded from the analysis (Aral et al., 2006). We therefore use emails collected during the period between 1 October 2002 and 3 March 2003 and between 1 October 2003 and 10 February 2004. During that period, there are 452,500 non-duplicated emails and 45,217 non-duplicated internal emails. This is the same subset of emails used in (Aral, Brynjolfsson, & Van Alstyne, 2006; 2007), (Aral & Van Alstyne, 2007), (Choe, 2006) and (Manoharn, 2006).

It is also worth mentioning that the original data schema in the database was not normalized. Some of the original tables included incorrect data types and duplicate values existed across different tables. For example, the main table contained string values representing the timestamp of emails instead of a date object. The recipients list was also stored as a string within the same table instead of being separately maintained within a different table while referencing back to the original table using a foreign key. The modified schema for the database and the full description of tables can be found in Appendix B: Executive Recruiting Firm Database Structure.

## *4.2. Entropy Measurements*

### 4.2.1. Theoretical Lower Bounds

In practice, a compression algorithm can rarely achieve the optimal entropy value calculated from the actual probability distribution. This limitation is due to a number of different factors. The data structures used by different algorithms such as the Huffman Trees have some intrinsic inefficiency in terms of generating the code sequences for tokens. Also, in producing binary sequences or code words algorithms are limited to use only integer values for the length of binary sequences (the number of bits). So even if the number of bits required to encode a character in the optimum case is equal to 6.1, in practice we still have to use 7 bits for encoding that character.

As previously mentioned, to calculate the theoretical lower bound for entropy, instead of looking at the length of the code words we use the actual probability distribution over the words (in our case tokens) and use equation (1) to calculate the entropy. Here, we are only looking at the probability distribution and ignoring the practical and achievable entropy values resulting from applying the specific methods of the compression algorithms. In essence, this lower bound removes the restriction of using bits in addition to any inherent inefficiency that the code generating part of the algorithm introduces.

We also calculate the average value of entropy for individuals based on their different folders (inbox, outbox and both). We average the entropy of messages in each set when the person is the sender, one of the receivers or either the sender or the receiver of a message. The average value of entropy for an individual could provide valuable insight when combined with other metrics of the information content of messages. For example, a person with higher entropy values related to their outbox might be more active in terms of the number of projects they are involved in and the higher entropy values might have a correlation with higher productivity as well.

The theoretical lower bounds provide us with a measure for validating our results and checking the correctness of our specific implementation of the Huffman compression algorithm. Because of the internal intrinsic inefficiencies specific to each compression algorithm as discussed in Section2.2, the calculated lower bound entropy values should always stay below the entropy values resulting from the actual output of these compression algorithms. If not, the implementation of the algorithm is definitely incorrect. In the next section we show how to calculate the lower bound and the achievable entropy values of a message by using the Huffman algorithm.

### 4.2.2. The Huffman Algorithm

In Section 2.2.2 we discussed that unlike LZW, the Huffman compression algorithm does not rely on the ordering of tokens in the message. In the previous section, we also talked about the transformations applied by the EmailNet algorithm. In essence, by removing the common words and stemming the remaining tokens, the EmailNet algorithm changes the order of tokens in the original message. Since the original order of tokens in not preserved in our data, selecting a sequence agnostic compression algorithm such as Huffman for measuring the entropy of messages appears to be a more reasonable choice over order dependent algorithms such as Lempel-Ziv.

As described in Section 2.2.2, the existing implementations of the Huffman algorithm perform a frequency analysis of all characters within the given text as a first step. In order to evaluate the validity of our measurements and quantify the entropy values of both plain text and hashed messages, two different versions of the Huffman compression algorithm were implemented.

The first version, used for plain text messages, recognizes characters as the main symbols and carries out a frequency analysis. The second version of the algorithm is modified to take the hashed version of words as input instead of characters. Instead of frequency analysis in the modified algorithm and since the EmailNet algorithm already includes the frequencies of all the hashed word, we simply use those frequencies to build the probability table and generate the Huffman Tree. Table 3, taken from Wikipedia shows an example of the Huffman algorithm's output together with the theoretical optimal bounds for entropy. In this table:

$A = \{a_1, a_2, a_3, \dots, a_n\}$ is the alphabet of size $n$ consisting of characters $a_1$ to $a_n$.

$W = \{w_1, w_2, w_3, \dots, w_n\}$ is the set of probabilities (or weights) associated with each character. $C(A, W) = \{c_1, c_2, c_3, \dots, c_n\}$ is the set of binary code words calculated by the Huffman algorithm consisting of one specific codeword (binary sequence) for each character.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Input** | Symbol ($a_i$) | A | b | c | d | e | **Sum** |
| | Weights ($w_i$) | 0.10 | 0.15 | 0.30 | 0.16 | 0.29 | = 1.00 |
| **Output** | Codewords ($c_i$) | 000 | 001 | 10 | 01 | 11 | |
| | Codeword length (in bits) ($l_i$) | 3 | 3 | 2 | 2 | 2 | |
| | Achievable Entropy ($l_i\ w_i$) | 0.30 | 0.45 | 0.60 | 0.32 | 0.58 | $H(C) = 2.25$ |
| **Optimality** | Information content (in bits) ($-\lg w_i$) ≈ | 3.32 | 2.74 | 1.74 | 2.64 | 1.79 | |
| | Lower Bound Entropy ($-w_i \lg w_i$) | 0.332 | 0.411 | 0.521 | 0.423 | 0.518 | $H(A) = 2.205$ |

**Table 3 - Huffman coding, optimal and achievable entropy bounds**

After finding out the frequency $w_i$ of each character, the next step is to build the Huffman Tree as described in Section 2.2.2. The Huffman Tree then outputs the code sequence $c_i$ for each one of the tokens based on their weights $w_i$ and codeword lengths $l_i$ . Using these values, we can compute the achievable entropy by summing over all the tokens in the message

$$H(achievable) = \sum_i w_i l_i \qquad (6)$$

Finally, the lower bound entropy is calculated by using equation (1), or in terms of the parameters in this table:

$$H\ (lower\ bound) = \sum_i -w_i\ lg\ w_i \qquad (7)$$

The implementation of the Huffman algorithm and the general structure of our software are discussed in more details in Appendix C: The Implementation of the Huffman Algorithm.

## 4.2.3. Validation and Enron Email Data

To evaluate the appropriateness of our methods for calculating the entropy of hashed email messages and to analyze the change in entropy values due to transformations exerted by the EmailNet algorithm, we use the publicly available Enron email data set. The data set consists of email messages exchanged between the employees of one firm which makes it very similar in terms of data to our data set. The fact that the data set consists of email messages, similar to our data, makes it a much better choice over analyzing random bodies of text. For our evaluation, we randomly select 1000 messages out of 252,759 emails.

First we used the Huffman algorithm to find the optimal and achievable entropy values for all the plain text messages, using the character frequencies and their corresponding probability distributions as the underlying model. Then we used the EmailNet algorithm to hash the body of all selected email messages using the same methods applied to our executive recruiting data. The modified version of the Huffman Algorithm was then used on the same set of messages and the new entropy values were computed. Finally, to find the relationship between the two entropy values, a regression analysis was applied to the entropy values of both plain text and hashed version of messages. In the final regression analysis, some other characteristics of messages such as the original length of emails were also taken into account.

## *4.3. Information Rate and Novel Information Measurements*

As discussed in Section 2.4, calculating the information rate requires treating a set of messages as one entity. We measured the information rate values for messages in each person's inbox, outbox and finally the combination of both folders. In each case, first the relevant messages were identified and sorted in chronological order.

Starting from the first messages in a folder, the novel information associated with the first message $m_1$ will always be equal to the entropy of that message simply because the previous state contains no data. Since this cutoff introduces a bias towards the first message and results in a much higher novel information value for those messages, we need a method for balancing the results depending on the number of messages seen which will be discussed later on in this section. For the second message $m_2$ however, the information rate is equal to

$$IR(m_2) = H(m_1, m_2) - H(m_1) \qquad (8)$$

Generalizing this to the set of all messages we have

$$IR(m_i) = H(m_1, m_2, \dots, m_i) - H(m_1, m_2, \dots, m_{i-1}) \qquad (9)$$

The entropy and mutual information values are as described in section 2.3. Again, we use the Huffman compression algorithm, building a new Huffman Tree for each message and calculating the entropies based on the frequencies of the tokens as described in Table 3.

The new frequencies are computed by concatenating all the messages together. As mentioned earlier, in order to compare the information rates among individuals, we need to calculate the average amount of novel information per individual per message, accounting for the bias introduced when the set of messages is relatively small.

In order to compute an average value, therefore, the amount of novel information calculated between one message and all previous messages should somehow be normalized to reflect the growing size of the messages. We achieved this by multiplying the (usually decreasing) amount of novel information by the number of the messages seen so far. If we represent the average novel information of message $i$ by $NI(m_i)$, then we have

$$NI(m_i) = i * IR(m_i) \qquad (10)$$

Finally, to find the average value of novel information for an individual, we calculated the mean of these values over all the messages in each one of their three folders. So if an individual $p$ has a total number of $n$ messages, then their average amount of novel information would be equal to

$$NI(p) = \frac{1}{n}\sum_{i=1}^{n} NI(m_i) \qquad (11)$$

Besides the average value of novel information, when looking at subsequent messages flowing into an individual's folder overtime, we usually expect to see a decreasing trend in the total amount of information, the reason being that higher amounts of information result in an increase in the probability of observing higher values of mutual information between these messages. In other words, as the universe containing all the messages seen so far grows in size, the probability of observing the same types of information also goes up. Finally, in order to make the results comparable to previous studies performed on same data set, all the information rate values for individuals were averaged over four-week periods as described in Appendix A: Recruiting Firm Email Data Set.

## *4.4. Mutual Information Measurements*

Mutual information provides a powerful measure representing the overlap of information between two individuals. To determine the amount of mutual information between two individuals, we have to calculate the entropy values associated with each person's messages as well as their joint entropy values. These values were computed for each set of folders separately, namely the inbox, the outbox and a combination of both folders.

In the next step, we considered all the possible pairings of our 73 individuals. For each possible pairing, we put all the messages of each individual in a bucket. We then calculated the entropy value of each bucket separately and the entropy value of the two buckets combined. Finally, the entropy of the mixed buckets was measured by using equation (3) from 2.3. These computations were carried out separately for each one of the three folders mentioned earlier (inbox, outbox and their combination). Our output therefore consists of the mutual information values between (i) inbox folders of two individuals (ii) outbox folders of two individuals and (iii) all the messages combined.

During the process of merging the messages and combining the folders, sometimes a single message could be listed more than once. This could happen for instance when someone sends a carbon copy of the message to themselves and are therefore listed under both the sender and the receiver of a message. Here, the additional copies of a message do not provide additional information to that individual. Therefore the message should be counted only once.

If both individuals receive a copy of the same message however, it means that they have access to the same information. The information in this case is shared between them and should be considered as mutual information. Consequently, the message should be listed twice, once for each individual.

## 5. Results

### *5.1. Entropy Measurements*

#### 5.1.1. Executive Recruiting Firm Data

In addition to theoretical lower bounds and achievable entropy values, our calculations also include the compressed and the original message sizes. Table 4 summarizes all the variables and the equations used for their calculation. A more comprehensive explanation of all these variables can be found in Sections 2.1 and 4.2.

| Variable | Raw Text | Hashed Text |
|---|---|---|
| Optimal Entropy | $= -\sum_{i=1}^{n} p(x_i)\,\mathrm{LOG}_2(p(x_i))$ Using character frequencies | $= -\sum_{i=1}^{n} p(x_i)\,\mathrm{LOG}_2(p(x_i))$ Using hashed token frequencies |
| Huffman Entropy | $= \sum_{i} w_i l_i$ Using character frequencies | $= \sum_{i} w_i l_i$ Using hashed token frequencies |
| Original Size (bits) | (Number of characters in the original message*8) | (Sum of Frequencies of Tokens in the Hashed message*8) |
| Compressed Size (bits) | $= \sum_{i} l_i f_i$ For each character in the original text where $f_i$ is the frequency of that character | $= \sum_{i} l_i f_i$ For each token in the hashed text where the $f_i$ is the frequency of that token |

**Table 4 - Summary of metrics for the entropy measurements**

Since each character in the ASCII standard takes 8 bits (one byte) of space, the original size of messages is calculated by multiplying the number of characters in the message by 8. Hashed tokens on the other hand are represented by a long integer value (see Table 2- Example of EmailNet Transformation Process) and a frequency as described previously in Section 4.2. For simplicity, we can assume that each one of these values (i.e. <5878157904011066551>) can be represented by one character (8 bits in size) in an arbitrary alphabet made up from all the hashed values in that message. Therefore the total size of the hashed message can be computed by calculating the sum of frequencies of hashed tokens and multiply the result by 8.

Table 5 contains the summary statistics of different variables for the set of 45217 email messages. Compression rate is equal to the ratio of compressed message size to the estimated original size as discussed earlier.

| | MEAN | STANDARD DEVIATION | MIN | MAX |
|---|---|---|---|---|
| LOWER BOUND ENTROPY | 5.591 | 1.606 | 0 | 11.149 |
| HUFFMAN ENTROPY | 5.638 | 1.61 | 0 | 11.207 |
| ORIGINAL SIZE | 1096.2 | 1876.4 | 0 | 154312 |
| COMPRESSED SIZE | 932.3 | 2007.8 | 0 | 170617 |
| COMPRESSION RATE | 0.7451 | 0.1497 | 0 | 1.4008 |

**Table 5 - Summary statistics for the entropy measurements for all the messages**

It must be noted that in our data, 1822 messages have a hashed body size (and therefore an entropy of) zero. Empty messages are usually the result of someone sending a message with no text in the body (i.e. using the subject line of the email for communication) or if the EmailNet algorithm discards all the words in the message (for example when the message body contains nothing but common words and special symbols) (Van Alstyne & Zhang, 2003).

Looking at the entropy values reveals that the lower bound and Huffman entropies in general and their mean and standard deviations in particular are very close to each other. This is mostly because of the efficiency of the Huffman Algorithm in using the optimum code sequences in most cases. Also, the Huffman entropy always stays above the lower bound.

The distribution of both entropy measures for all 45217 email messages are shown in Figure 2. For the most part, the distributions show a very similar behavior. Near the middle of the graph and close to the mean values however, the total number of messages for the Huffman entropy falls slightly below the lower bound at lower entropy values and rises slightly above it around the peak value. This effect can be attributed to the internal inefficiency of the Huffman algorithm at some specific thresholds in terms of the original message size and the limitation of using an integer number of bits for code sequences.

In practice the difference between the entropies will be greater at certain thresholds. For instance, after the algorithm used all possible 5-bit code sequences (i.e. for 90% of all the symbols) it starts coding the rest of the symbols with 6-bit long binary sequences. At this point, the total efficiency of the Huffman algorithm will suddenly decrease resulting in a sudden decrease of compression rate and therefore the entropy value. Here, while the theoretical optimum bounds may indicate that the optimum number of bits at this certain threshold is equal to 5.1, the Huffman algorithm is forced to use 6 bits of code for each of the remaining tokens. This effect is more clearly seen in the middle parts of the plot due to the higher concentration of data points.

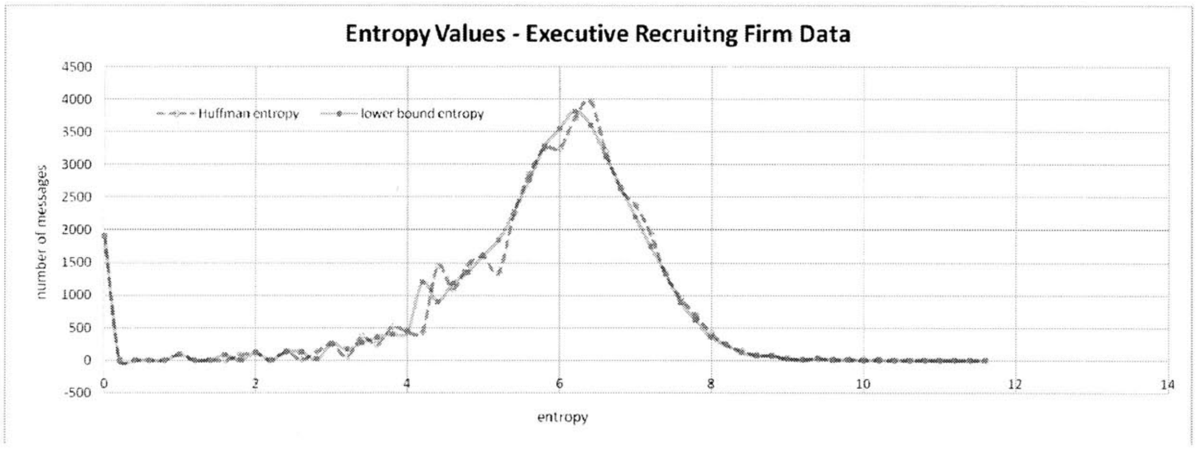

**Figure 2 - Distribution of entropy values for messages, executive recruiting firm data**

Looking at the compression rate shows a maximum value greater than one. More specifically, out of 45217 data points, 618 points or 1.37 percent of them have a compression rate greater than one. At first glance, it appears that this could be a result of the simple model we selected for estimating the original message size. Looking more closely at the data however, reveals the possible existence of a relationship between the compression rate and the original messages size.

Figure 3 shows a graph of the compression rate plotted against the logarithm of the original message size for a randomly selected sample of 4500 messages. Since the messages of size zero provide no additional meaningful information for our regression analysis (the entropy, compression rate and original size values are all zero), all data points with the entropy value of zero were removed from the dataset.

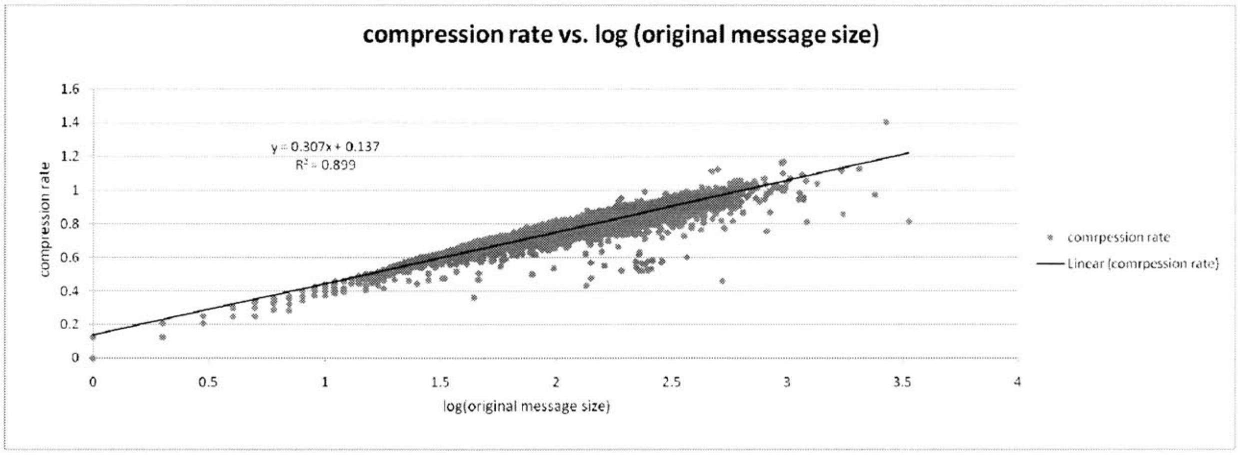

**Figure 3 - Plot of compression rate of hashed messages vs. the logarithm of original message size**

With a linear regression analysis as shown on the plot, where $x$ is the logarithm of the original message size and $y$ represents the compression rate, we find the following relationship:

$$y = 0.307x + 0.137 \qquad (12)$$

$$R^2 = 0.899$$

The compression rate depends on the entropy value of the messages and usually as we will see in Section 5.1.2.1, the entropy of plain text messages is independent of the message size. But that does not seem to be the case here. This behavior could be attributed to the way the EmailNet algorithm handles the raw messages.

In general, as the message size increases, the Huffman algorithm has to use a larger alphabet for encoding all the symbols. An increase in the alphabet size leads to a bigger Huffman Tree and longer code sequences. Longer code sequences will result in worse compression rates and lower entropy values. Earlier, when estimating the original hashed message size, we used 8 bits for the size of each token, only allowing for an alphabet of size $2^8 = 256$. However when the number of tokens exceeds this threshold, the alphabet size has to grow accordingly, making room for the newly added symbols. A more complete discussion of this behavior and why it can be attributed to the hashing mechanism of the EmailNet algorithm can be found in Section 5.1.2.1.

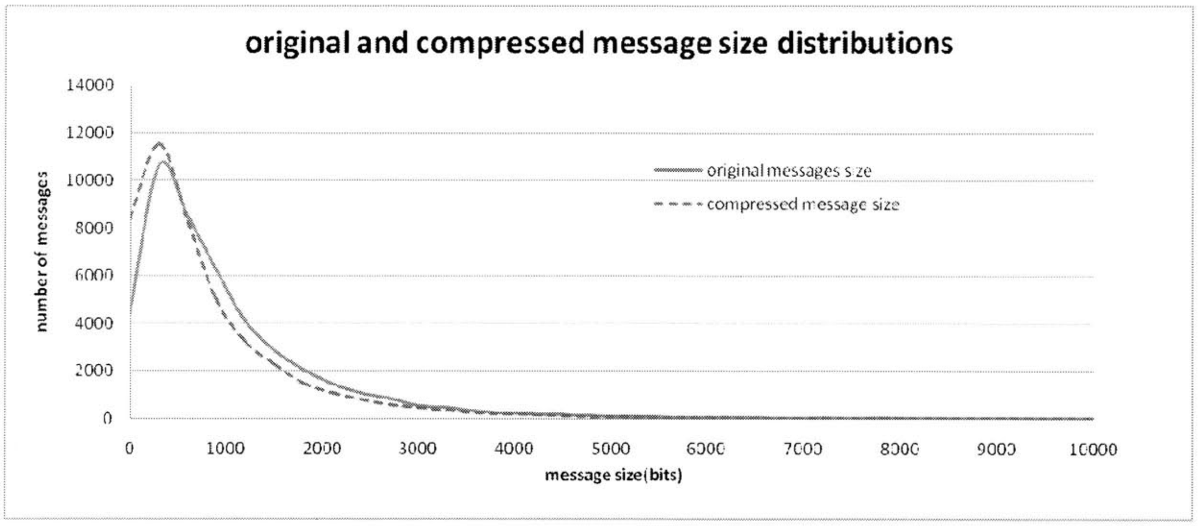

**Figure 4 - Original and compressed message size distributions in hashed emails**

Figure 4 shows a distribution of the original and compressed message sizes in the hashed emails. As can be seen in the graph, the difference between the original and compressed message sizes diminishes as the messages size goes up, which is caused by the increasing alphabet size and higher compression rates as described previously.

### *5.1.1.1. Entropy Values for the Individuals*

In addition to calculating the entropy values for every single message, we also calculated the average amount of entropy for each individual. Figure 5 displays the distribution of entropy values for the inbox and outbox folders across all the employees while Figure 6 displays the combination of inbox + outbox vs. the outbox folders.

To better visualize the results, the $x$ intercept for each point represents the average entropy of their outbox folder while the $y$ intercept shows the average value of entropy corresponding to the inbox of the same individual. All the values represent the achievable entropies; the lower bound entropies were very close in value to the Huffman entropies and therefore are not shown on the graph.

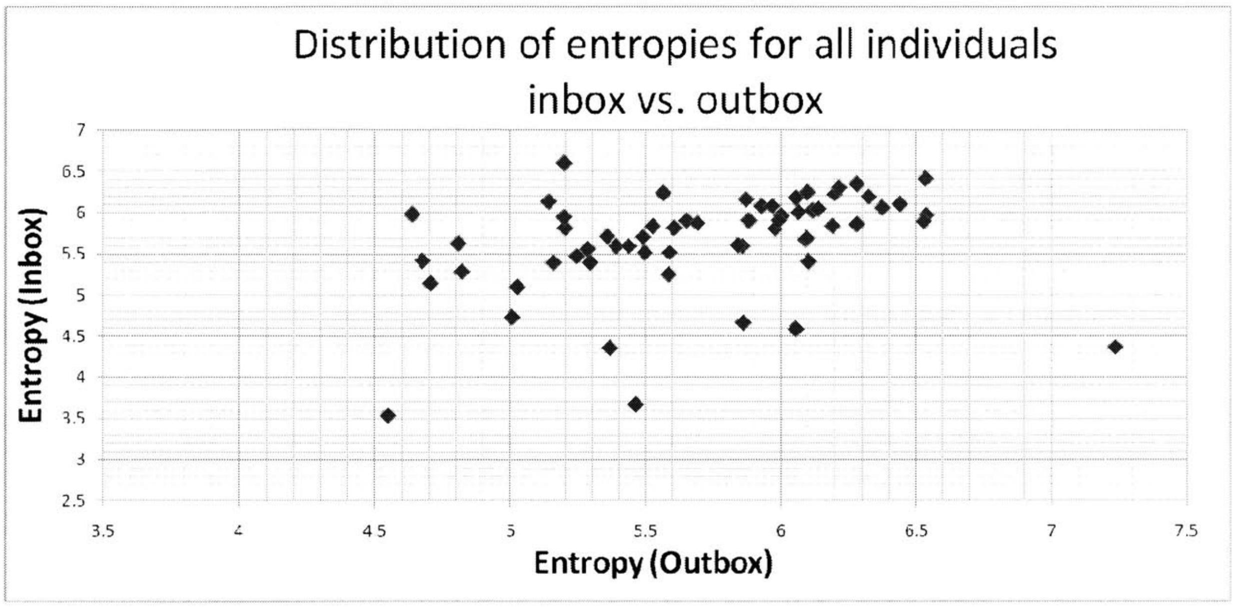

**Figure 5 - Entropy values for individuals – inbox vs. outbox folders**

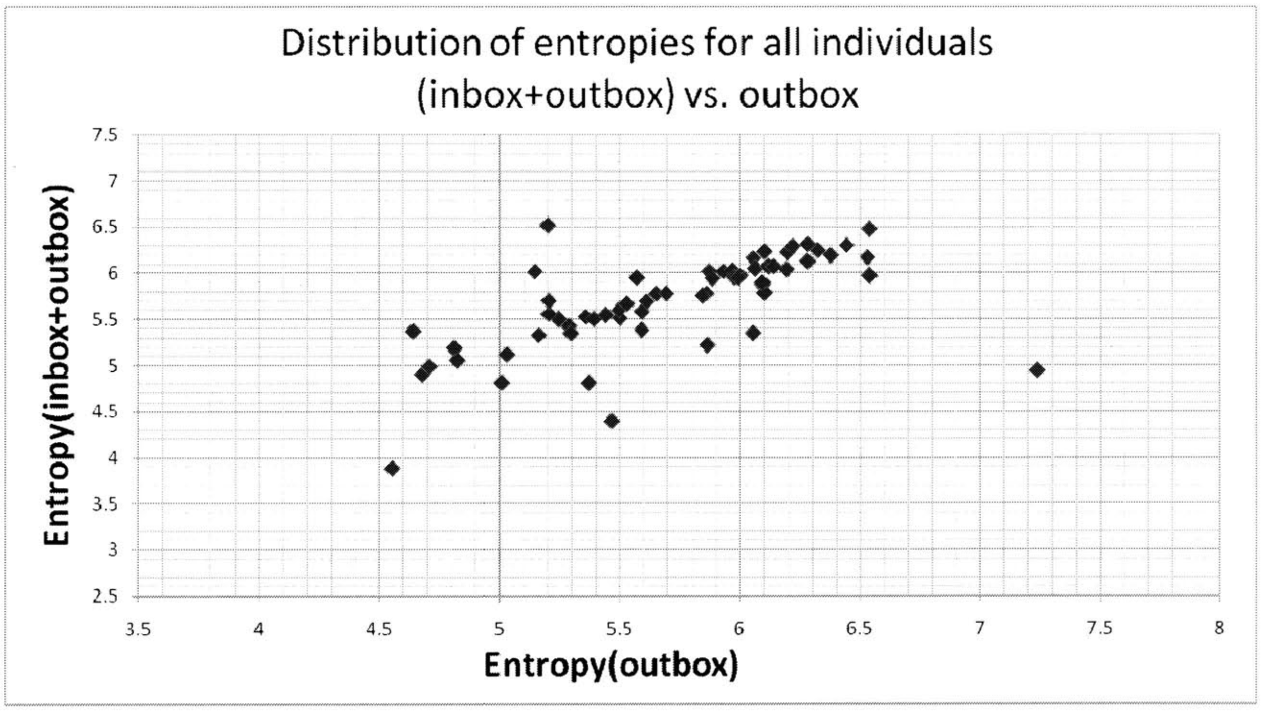

**Figure 6 - Entropy values for individuals - inbox+outbox vs. outbox**

A person with higher values of entropy for both their inbox and outbox folders therefore would be represented by a point closer to the top right corner of the graph. The high values of entropy for an individual mean that the average amount of information they send and receive per symbol is higher than their peers. For instance, if two individuals, $A$ with a high entropy value and $B$ with a low entropy value, both compose a message (looking at outbox entropies) of size $n$, on average the informational content of the message sent by $A$ would be higher than the one sent by $B$. The same argument applies to the content of inbox and combination of both folders.

Being capable of communicating more information in a shorter message can be an important variable in the success of a knowledge worker. From the graph, we can also identify individuals who have a high inbox but low outbox entropy (points on the top left corner of the plot). High inbox entropy might be indicative of working with diverse information as a result of working on several projects at the same time. The combination with low outbox entropy could possibly be due to the fact that the person is incapable of keeping up with the flow of information to their inbox, perhaps as a result of being involved in too many projects.

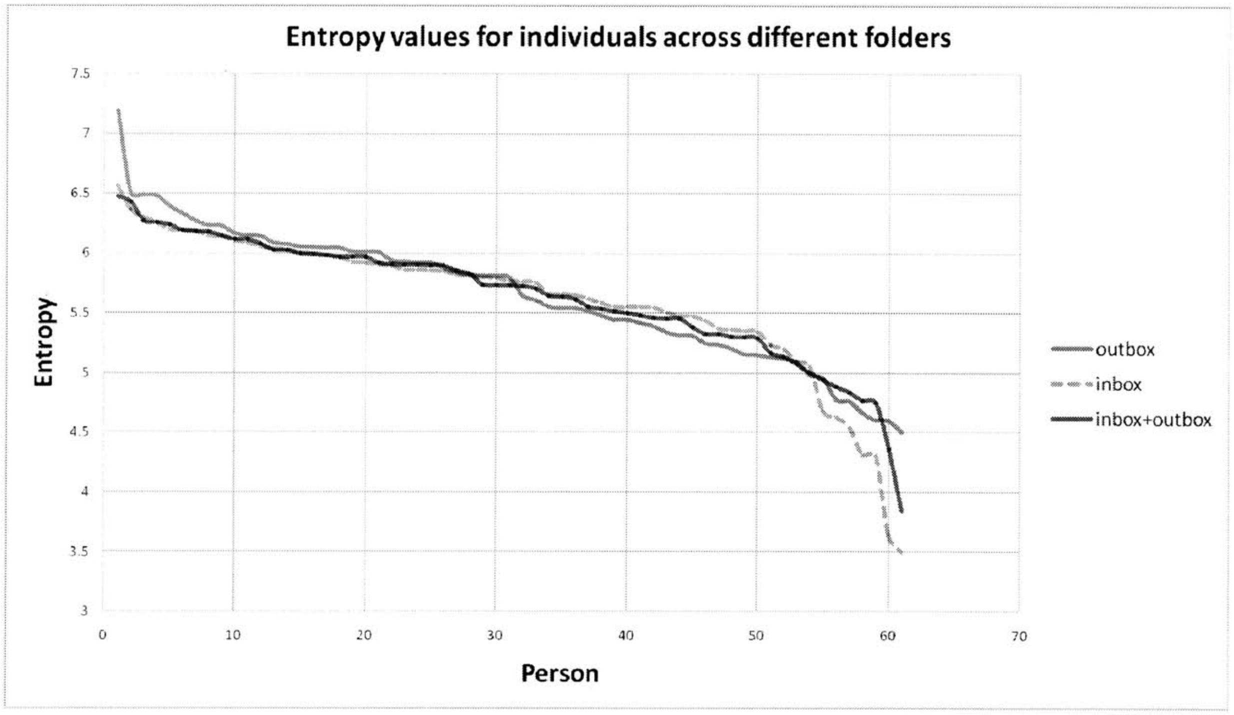

**Figure 7 - Entropy values for individuals across different folders**

Figure 7 shows the distribution of entropy values for individuals sorted in decreasing order for all three folders. The slope of all three curves remains relatively constant in the middle section but there are a few drops at the head (maximum values) and tail sections of the graph. Therefore, if the individuals are ranked based on their entropy values, there is a little difference between subsequent people in the list. The only exceptions are top individuals (in terms of entropy) on the outbox list and bottom employees in the other two folders (especially inbox).

When receiving a message also, higher entropy values indicate that on average the person has to deal with shorter messages that have the same amount of information when compared to someone with lower entropy. If we the message sizes are equal however, on average the message belonging to the individual with higher inbox entropy is richer in content. This may lead to spending less time looking for the valuable pieces of information and the information received may contribute to higher levels of uncertainty resolution *per unit of information received*. Future studies will be able to confirm whether higher or lower entropy values can predict the productivity of an individual by linking these data to detailed measures of individual output.

### 5.1.2. Enron Data Set

#### *5.1.2.1. Plain Text Entropies*

Table 6 shows a sample output of the entropy values for plain text messages.

| EMAIL ID | LOWER BOUND ENTROPY | HUFFMAN ENTROPY | ORIGINAL MESSAGE SIZE | COMPRESSED SIZE | COMPRESSION RATE |
|---|---|---|---|---|---|
| 1134 | 4.85 | 4.90 | 784 | 480 | 0.612 |
| 1272 | 4.81 | 4.84 | 22824 | 13818 | 0.605 |
| 1668 | 4.76 | 4.80 | 95536 | 57275 | 0.600 |
| 1719 | 4.92 | 4.96 | 351360 | 217738 | 0.620 |
| 1871 | 2.81 | 2.86 | 56 | 20 | 0.357 |
| 2116 | 4.47 | 4.50 | 13664 | 7688 | 0.563 |
| 2397 | 4.32 | 4.34 | 13296 | 7219 | 0.543 |
| 2900 | 5.16 | 5.19 | 4944 | 3206 | 0.648 |

**Table 6 - Sample plain text entropy values**

The achievable entropy values (referred to as Huffman entropy for simplicity) resulting from the Huffman algorithm are a little higher but still very close to the lower entropy bounds which shows that the Huffman algorithm is highly efficient in encoding the messages based on its selected probability distribution (frequency of tokens).

We can also observe that the entropy value is independent from the message size. For example, the second message in the table (Email ID: 1272) is almost 30 times larger than the first message (Email ID: 1134). However, their entropy values are almost identical. One has to make a distinction between the *entropy* value and the *total amount of information*.

Here, entropy represents the average amount of information per symbol. Therefore, even though the entropy values are similar, the total informational content of the larger message is higher. In other words, if the total amount of information is held constant, someone with a higher average entropy value would be able to convey the same amount of information in a shorter message than someone with a low entropy value. Table 7 shows the summary statistics of entropy measurements for plain text messages.

| | Number of observations | Mean | Standard Deviation | Min | Max |
|---|---|---|---|---|---|
| Lower bound entropy | 1000 | 4.691 | 0.4712 | 0 | 5.663 |
| Huffman entropy | 1000 | 4.655 | 0.4892 | 0 | 5.695 |
| Original size | 1000 | 16420.2 | 49170.8 | 0 | 524088 |
| Compressed size | 1000 | 9297.7 | 26339.3 | 0 | 309214 |
| Compression rate | 1000 | 0.5863 | 0.0589 | 0 | 0.7119 |

**Table 7- Summary statistics of entropy measurements of plain text messages, Enron emails**

In making important decisions or when dealing with large amounts of information, the amount of data that a person has to process usually becomes a major bottleneck. Most knowledge workers have to go through the process of extracting the meaningful and useful pieces of information from the piles of data delivered to them.

Many times, an important piece of information can easily be neglected if the size of the data is too much for the individual to efficiently process - they may not have the time or the capacity to decipher the whole message (or set of messages). The individuals capable of expressing their ideas more concisely and more efficiently, or those who receive concise efficient messages from others, may have an advantage in the effectiveness of their communication with others and with being more productive in workplace. Although it is beyond the scope of this research, it seems that entropy values may be a good a proxy for this characteristic of individuals' communication.

Figure 8 shows the achievable and optimal entropy values for all 1000 randomly selected messages. As described earlier the Huffman entropy value stays very close to the optimal bound. The plot is very similar in shape to Figure 2, although the average entropy value is lower for hashed text messages.

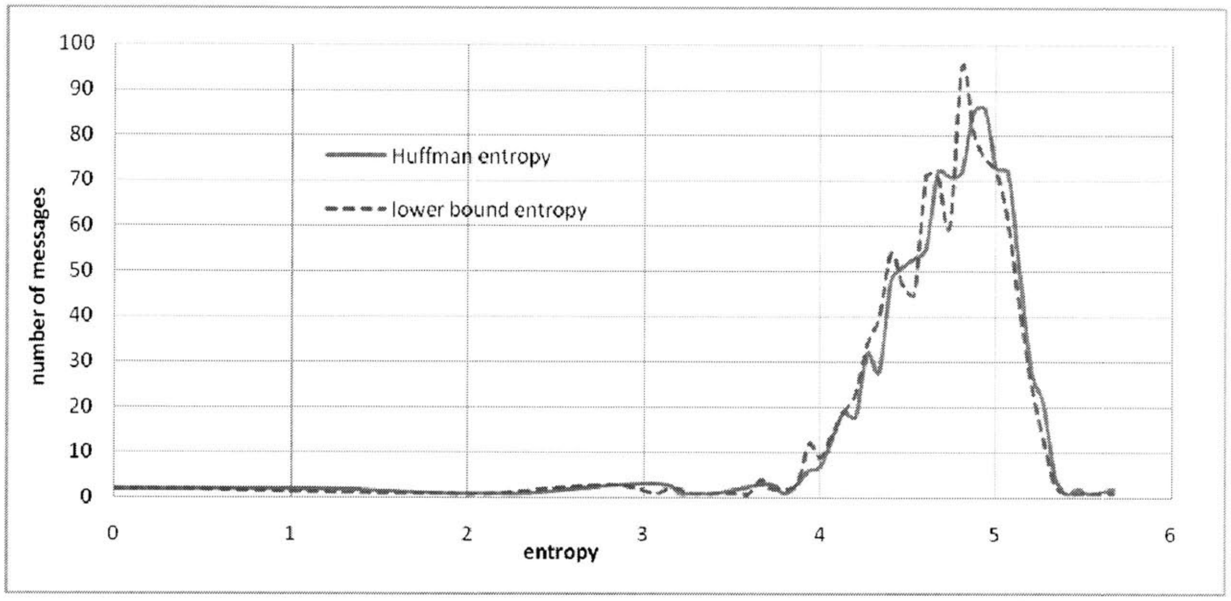

**Figure 8 - Plain text messages, achievable and optimal entropies**

The steps in the graph can be attributed to the nature of our data, and also the rounding effect of entropy values for the selected intervals each one containing the cumulative number of messages in their range. Each step for lower bound entropy values near the middle represents a difference of about 10 messages or one percent of the total data.

Figure 9 shows the distribution of compression rate against the logarithm of original messages size similar to what we had before for the hashed executive recruiting firm messages. Performing a regression analysis shows that unlike the relationship found in the previous section between the hashed messages however, the plain text compression rate seem to be independent of the original messages size. The analysis, shown in the figure, displays no significant dependence between the variables, yielding an $R^2$ value of only 0.194 compared to the value of 0.899 we obtained for the hashed messages in previous section.

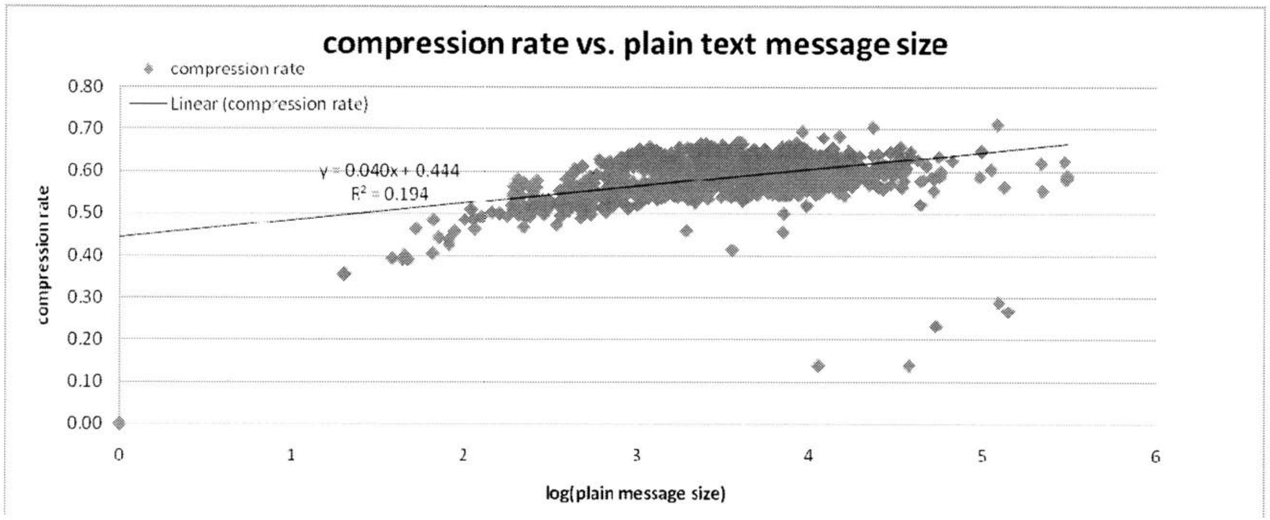

**Figure 9 - Compression rate vs. raw message size, plain text messages**

This result is not very surprising. As discussed earlier, the size of a message in general is independent from its entropy. This however was not the case after the EmailNet algorithm hashed the messages. Our observation that a relationship in fact exists between the compression rate (and therefore the entropy) and the hashed message size (which is the output of the EmailNet algorithm) suggests that the hashing transformation applied by the EmailNet algorithm is somewhat similar to a compression algorithm, where the size of the compressed message (the output of the compression algorithm) is directly related to the compression rate (and therefore the entropy).

In a compression algorithm, the output, which is the compressed message, is not anymore compressible, and if we tried to recompress the message, the compression rate would be equal to one (same output as the input). The output of the EmailNet algorithm - a hashed message - however, has only been compressed to a certain extent. That is why unlike the output of a compression algorithm, we can still compress a hashed message. The resulting compression rate although not constant, still maintains a relationship with the message size, which suggests that the two methods (compression algorithm and the EmailNet) are somewhat similar in nature.

In fact the stemming process in EmailNet is very similar to the frequency analysis performed by the Huffman algorithm. What the Huffman algorithm does additionally is producing optimal varying length code sequences for each token while the EmailNet algorithm on the other hand produces fixed length code sequences. As the message size increases and with greater number of tokens, the compression applied by the EmailNet algorithm becomes less and less efficient and has to pay more penalties for its fixed-length code sequences in comparison with the Huffman algorithm. So even though EmailNet is not a real compression algorithm and its output is not optimally compressed, it seems that it still maintains the entropy value of the original message.

We still need to know the exact relationship between the hashed message and the plain text entropies. In the next section, we will analyze this relationship by hashing the plain text messages, measuring their entropies and comparing them with the plain text entropies.

#### *5.1.2.2. Hashed Emails Entropies*

Table 8 shows the summary statistics of entropy measurements for the hashed version of the same 1000 randomly selected plain text messages used earlier. We hashed the plain text messages using the EmailNet algorithm and measured the optimal and the Huffman entropy values. All the variables in the section are similar to those used for plain text messages and the executive recruiting firm email data. As can be seen in the table, the standard deviation of hashed messages is much greater than the plain text messages. This could be attributed to the fact that in the messages hashed by the EmailNet algorithm, the compression rates and therefore the entropy values depend on the original message size, therefore as the message size changes, EmailNet produces messages with a wider range of entropies while in a real compression algorithm, entropy only depends on the content of the message.

| | Number of messages | Mean | Standard Deviation | Min | Max |
|---|---|---|---|---|---|
| Lower bound entropy | 1000 | 5.4673 | 1.5214 | 0 | 10.1204 |
| Huffman entropy | 1000 | 5.5154 | 1.5182 | 0 | 10.1497 |
| Original size | 1000 | 1461.8550 | 5250.3187 | 0 | 67861 |
| Compressed size | 1000 | 1603.9040 | 4815.0161 | 0 | 64544 |
| Compression rate | 1000 | 0.6914 | 0.1878 | 0 | 1.2687 |

**Table 8- Summary statistics of entropy measurements of hashed text messages, Enron emails**

Performing a T-test on the Enron and the executive recruiting firm messages yields a very low significance level as can be seen in Table 9. The T-test shows that the mean of these variables are in fact very close to each other. Besides the similarity of data, this could also be attributed to the fact that the same transformations were applied by the EmailNet algorithm on both set of data.

| | ESTIMATE | STANDARD ERROR | P-VALUE |
|---|---|---|---|
| LOWER BOUND ENTROPY | 3.5138 | 0.0487 | 0.0079 |
| HUFFMAN ENTROPY | 1.5573 | 0.0486 | 0.0084 |
| ORIGINAL MESSAGE SIZE | -3.3286 | 152.52 | 6.7E-16 |
| COMPRESSED MESSAGE SIZE | -3.1844 | 166.30 | 3.9E-15 |
| COMPRESSION RATE | 8.9771 | 0.0059 | 4E-29 |

**Table 9- T-test results for hashed message metrics**

Table 10 shows a sample output of the entropy values for a few messages.

| EMAIL ID | LOWER BOUND ENTROPY | HUFFMAN ENTROPY | ORIGINAL MESSAGE SIZE | COMPRESSED SIZE | COMPRESSION RATE |
|---|---|---|---|---|---|
| 1134 | 3.222 | 3.170 | 72 | 9 | 0.403 |
| 1272 | 7.064 | 7.039 | 2264 | 283 | 0.883 |
| 1668 | 7.998 | 7.976 | 8920 | 8918 | 1.000 |
| 1719 | 5.063 | 5.014 | 49544 | 31354 | 0.633 |
| 1871 | 0.000 | 0.000 | 8 | 0 | 0.000 |
| 2116 | 7.093 | 7.014 | 1296 | 1149 | 0.887 |
| 2397 | 6.765 | 6.704 | 1224 | 1035 | 0.846 |
| 2900 | 5.630 | 5.562 | 648 | 456 | 0.704 |

**Table 10 - Sample hashed text entropy values**

Figure 10 shows the achievable and optimal entropy values for all 1000 randomly selected hashed messages. Again, the plot is very similar in shape to Figure 2 although the smaller number of sample points (1000 vs. 45217) results in a rougher outline. The number of empty messages (those with a size of zero) is significantly higher in the executive recruiting firm dataset (4% of total messages as opposed to 0.8% for the Enron data).

Since the plain text messages from the executive recruiting firm data was not available, it is difficult to see what may have caused this unusually high number of empty messages. One may attribute this to the fact that the Enron email data set was selected from messages with important and valuable content to the investigators and thereby eliminating any empty messages in the process. Also, if only just one of the employees within the executive recruiting firm mainly uses email for sending attachments or as a backup mechanism for their local files and therefore producing empty messages, the data will be heavily biased. Since there are only 73 individuals within the firm, one or two individuals using their emails in the described manner would be enough to explain the 4% ratio of empty messages.

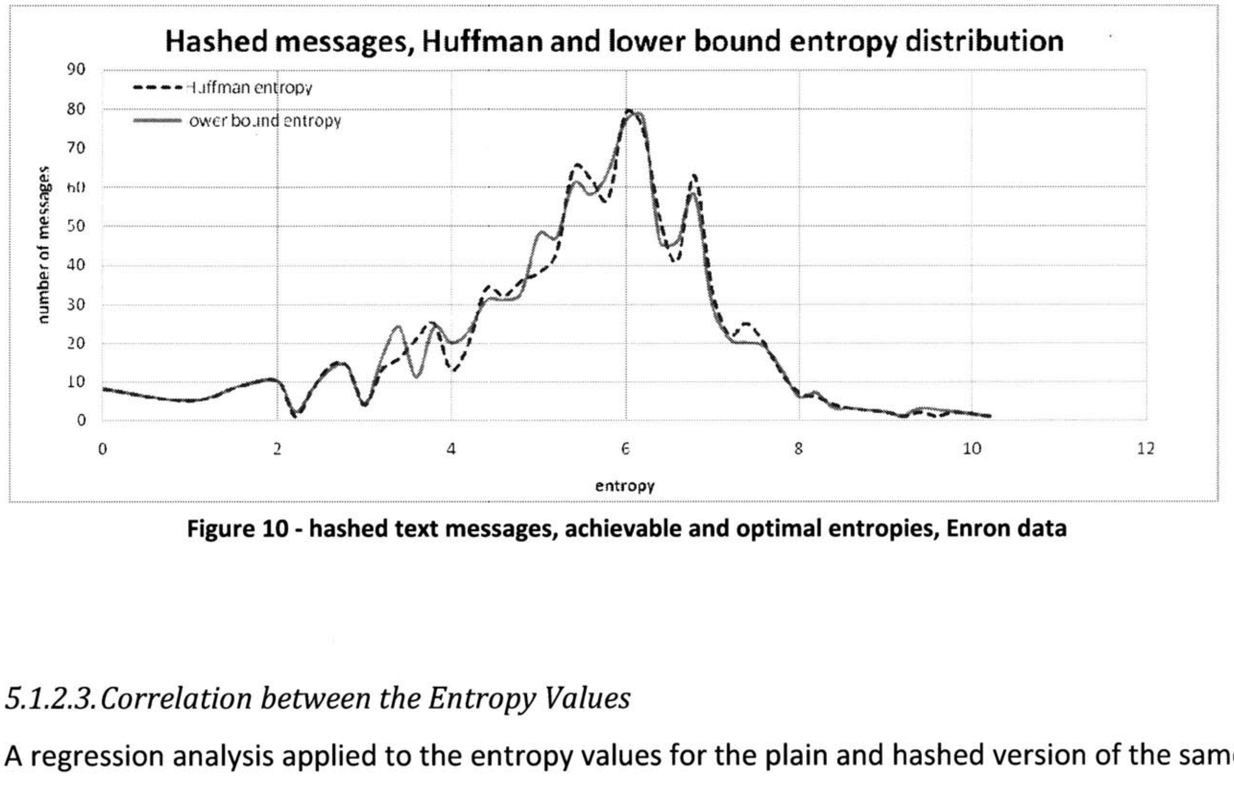

**Figure 10 - hashed text messages, achievable and optimal entropies, Enron data**

### *5.1.2.3. Correlation between the Entropy Values*

A regression analysis applied to the entropy values for the plain and hashed version of the same messages shows a correlation between the entropy values and raw message size. If we define the following variables:

$$\Delta H_a = H_{hashed-text}(achievable) - H_{plain-text}(achievable)$$

$$\Delta H_b = H_{hashed-text}(optimal) - H_{plain-text}(optimal)$$

$$x = \log_{10}(message - size)$$

Then with a linear regression analysis, as shown in Figure 11 and Figure 12, we find the following relation between $\Delta H_a$ ,$\Delta H_b$ and $x$.

$$confidence - levels = \ 99.0\%$$

$$\Delta H_a = 2.047x - 6.851, \quad R^2 = 0.872, \qquad p - value = 0, \qquad st.error = \ 0.4840 \qquad (13)$$

$$\Delta H_b = 2.059x - 6.909, \quad R^2 = 0.874, \qquad p - value = 0\,, \qquad st.error = 0.4840 \qquad (14)$$

The $R^2$ and $p$ values show that there is a good fit between the proposed model and actual data considering the high confidence level of 99%. Furthermore, the theoretical optimal bounds and information entropy measures achieved by the Huffman algorithm show very similar behaviors and the two coefficients are very close to each other. These results show that using the EmailNet algorithm to code plain text messages does not affect the informational content of the original text in any significant way, but also that the relationship between the theoretical and achievable entropy values depends on the original size of the plain text message.

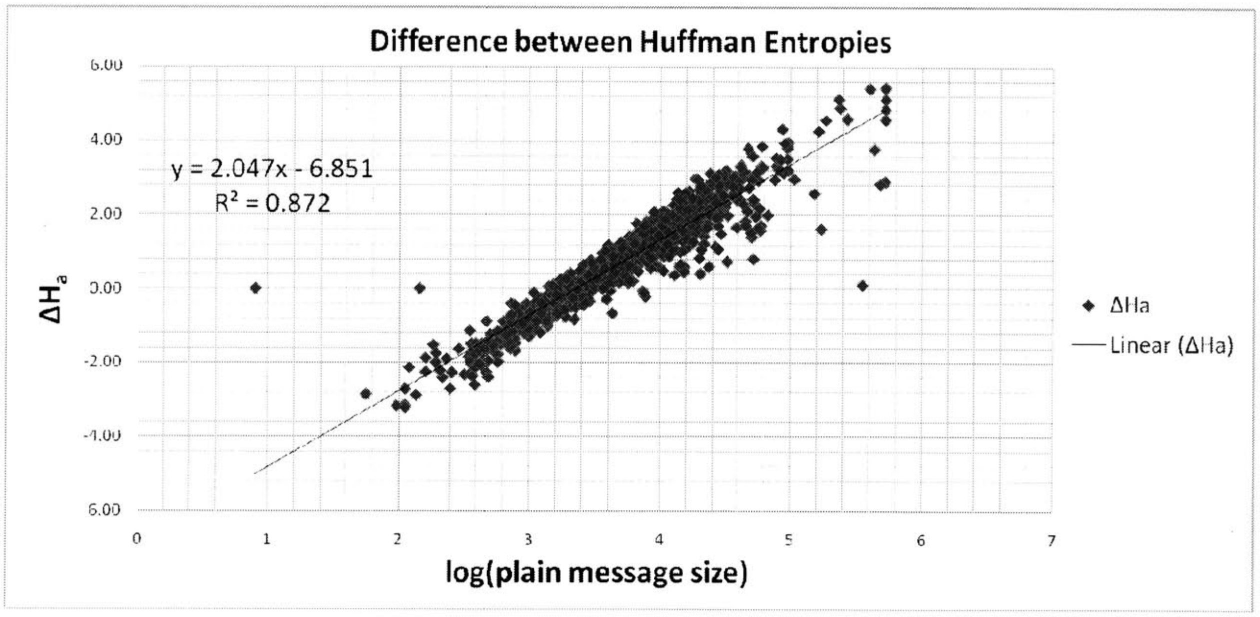

**Figure 11 - Correlation between messages size and achievable entropy values computed by Huffman algorithm**

The first coefficient in Equation 13 (i.e. 2.047) indicates that an order of magnitude increase of in original message size corresponds to multiplying the difference of the hashed message and the original message entropies by 2.047. In other words, as messages get bigger, the plain text probability distribution model (character frequencies) outperforms its hashed text counterpart (hashed vector or stemmed word frequencies) by a greater margin.

In messages of smaller size, the hashed messages have lower entropies which are represented on the graph by the points with a negative value of difference between two entropies. When message size is small, the hashed tokens are limited in number; the alphabet size (total number of different hashed words) is small and the message is far shorter in number of distinct tokens and their total frequencies than the original.

This is due to the fact that EmailNet discards many characters as it removes any special symbols and common words within the text or when it stems the words. For instance, in a message originally 200 characters and 40 words long, we might have 40 distinct characters (including symbols, numbers, punctuation marks, etc...) but only 20 distinct hashed tokens. Assuming a uniform probability distribution, more tokens usually lead to higher entropy values.

As the message size increase, the alphabet size for characters does not change significantly (a maximum of 256 in ASCII standard), while the number of distinct words grows substantially. Compared with the limited number of characters, we observed more than 91000 distinct hashed tokens in the 45217 email messages of the executive recruiting firm data. Therefore, as the message size grows, the character frequency model achieves better compression rates and lower entropies, resulting in an increase in the difference between the two entropies.

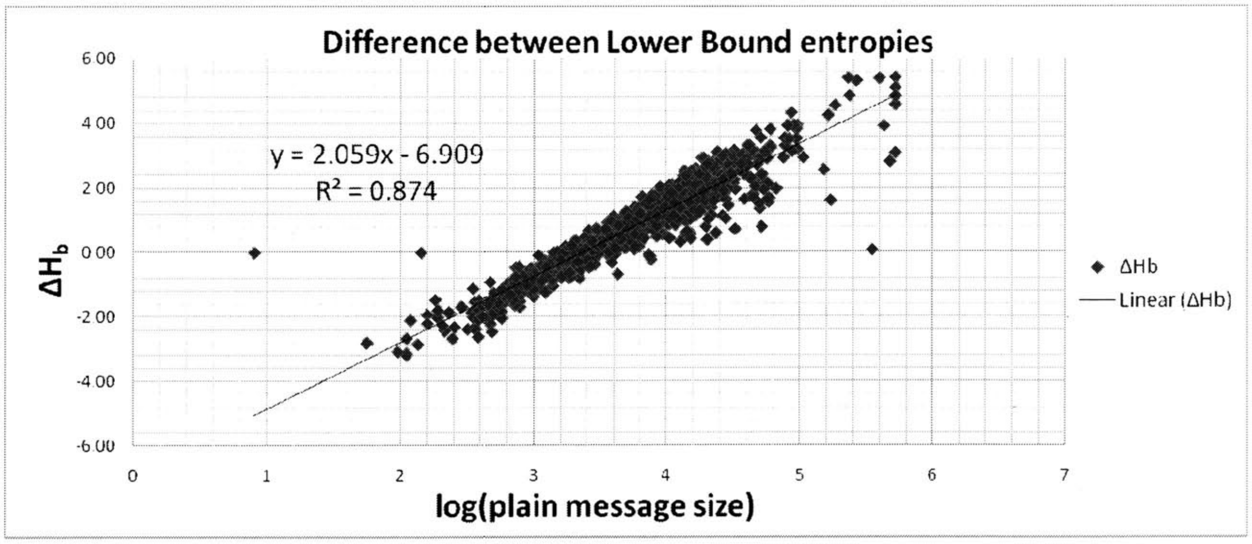

**Figure 12 - Correlation between messages size and optimal lower bound entropy**

## *5.2. Calculating the Biased Corrected Plain Text Entropy*

As we showed in the previous section, the hashed message entropy is a function of two variables, the raw message entropy and the raw message size[2]. Since our data consists only of the hashed messages and we do not have access to the plain text emails, we need to find a way to estimate the plain text characteristics, its entropy $H_r$ and message size $S_r$ from the metrics available to us, the hashed message entropy $H_h$ and the hashed message size $S_h$.

Equations (13) and (14) show that the general form of the relationship between the difference of entropies and the original message size is of the form,

$$(H_h - H_r) = c_1 \log(S_r) + c_2 \qquad (15)$$

Where $c_1$ and $c_2$ are some constants estimated in out regression from Equations 13 and 14. The use of parameter estimates from either of the equations depends on whether we are looking for the optimal or the Huffman entropy values. From the definition of entropy we also know that

$$compression\,rate = \frac{S_h}{S_r} = \frac{H_r}{8} \qquad (16)$$

This is simply due to the fact that entropy represents the average number of bits needed for representing each token in the compressed message. In the plain text message however, the number of bits required for each character is fixed and equal to 8. Therefore, the compression rate is equal to entropy divided by 8. From Equation (16) we have,

$$S_r = \frac{8 * S_h}{H_r} \qquad (17)$$

$$\rightarrow \log(S_r) = \log(8) + \log(S_h) - \log(H_r)$$

[2] Equations (13) and (14).

Substituting back into (15):

$$(H_h - H_r) = c_1\{\log(8) + \log(S_h) - \log(H_r)\} + c_2 \quad \rightarrow$$

$$H_r - c_1 \log(H_r) = H_h - c_1[\log(8) + \log(S_h)] - c_2 \qquad (18)$$

Since all the values are known, we can substitute the right hand side of the Equation (18) with a new constant $c_3$,

$$c_3 = H_h - c_1[\log(8) + \log(S_h)] - c_2 \qquad (19)$$

$$H_r - c_1 \log_{10}(H_r) = c_3 \qquad (20)$$

Equation (20) can be solved analytically by using the Lambert $W$ function, also called the Omega function or product log. The Omega function denoted by $W$ is the inverse function of $f(z) = z * \exp(z)$ where $\exp(z)$ is the natural exponential function and $z$ is any complex number. It can be used to solve various equations involving exponential terms[3]. Following its definition, for every complex number $z$,

$$z = W(z)\,\mathrm{e}^{\mathrm{W(z)}} \qquad (21)$$

To get the results in a cleaner way it is better to use the natural logarithm in Equation (20) instead of the based ten logarithm.

$$H_r - c_1 \frac{\ln(H_r)}{\ln(10)} = c_3$$

And after defining a new constant $c_4$

$$c_4 = \frac{c_1}{\ln(10)} \qquad (22)$$

We will have

[3] For a more comprehensive description of the Lambert W function please look at (Corless, Gonnet, Hare, Jeffrey, & Knuth, 1996).

$$H_r - c_4 \ln(H_r) = c_3 \qquad (23)$$

By applying the $W$ function we can find the estimated value of the original raw text entropy $H_r$,

$$H_r = -c_4 W\left(\frac{-\exp\left(-\frac{c3}{c4}\right)}{c_4}\right) \qquad (24)$$

Or after substituting back the constants $c_3$ and $c_4$ we get the value of $H_r$ in terms of the hashed text entropy $H_h$ and the hashed text message size $S_h$

$$H_r = -\frac{c_1}{\ln(10)} * W\left(\frac{-\exp\left(-\frac{H_h - c_1[log(8) + log(S_h)] - c_2}{c_1/\ln(10)}\right)}{c_1/\ln(10)}\right) \qquad (25)$$

Equation (25) allows us to calculate the value of raw text entropy based on the hashed text entropy and the hashed text message size. The raw message size $S_r$ then can be simply calculated by plugging in the values of $H_r$ and $S_h$ into Equation (26).

$$S_r = \frac{8 * S_h}{H_r} \qquad (26)$$

Equation (25) can be used to measure both the optimal lower bound and the achievable Huffman entropies simply by plugging in the corresponding constants. For the optimal lower bound we had $c_1 = 2.059$ and $c_2 = -6.851$ while for the Huffman entropies, $c_1 = 2.047$and $c_2 = -6.909$.

It must be noted that the current results in this paper do not include the calculated raw text entropies resulting from the equation (25), instead we used the actual hashed text entropies computed by applying the Huffman algorithm. By using these equations however, future studies will be able to recalculate the other informational measures of data such as novel and mutual information and compare and contrast them with the hashed text measures.

It is worth mentioning that if one has to decide between only one of two different entropy measures for studying the relationships between the informational content and productivity of the knowledge workers, the optimal lower bound value seems to be a more reasonable choice. The optimal lower bound provides us with a *pure* measure of entropy free of any limitations imposed by the compression algorithms and only dependent on the underlying probability distribution. Since we are more interested in true informational content of the data rather than compressing, transmitting or reconstructing the compressed messages, the optimal entropy bounds appear to provide us with a more realistic measure. The practical bounds resulting from compression algorithms such as Huffman on the other hand include some unnecessary bias due to the restrictions and the inefficiencies of the algorithm.

## *5.3. Information Rate Measurements*

Figure 13 shows an example of total entropy associated with an individual's outbox increasing over the period of 41 weeks. The entropy value initially shows a sharp rise and after a certain threshold, the increase in total entropy is minimal and it merges to a final value of 10.237 in this case.

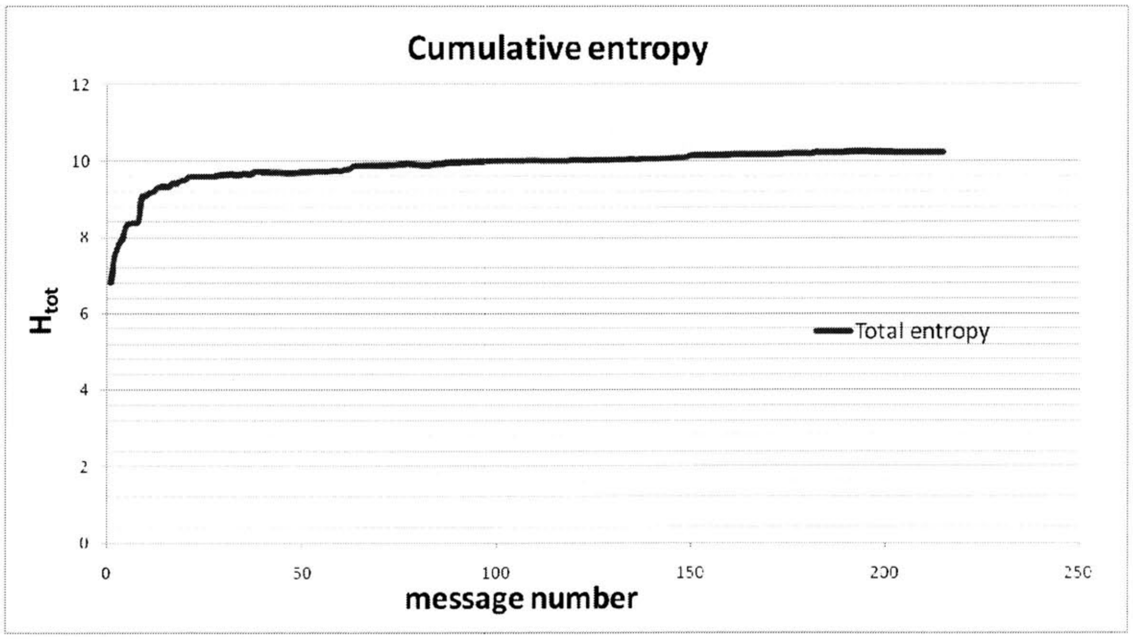

**Figure 13 - Change in total information content of an individual's outbox over time**

Figure 14 on the other hand shows the change in the amount of entropy over time or the information rate. The function follows a natural pattern of decay, approaching zero as time goes by. This valuable piece of information could provide insight into the pattern of access to new information for each individual in the firm and could be analyzed for its relation with entropy.

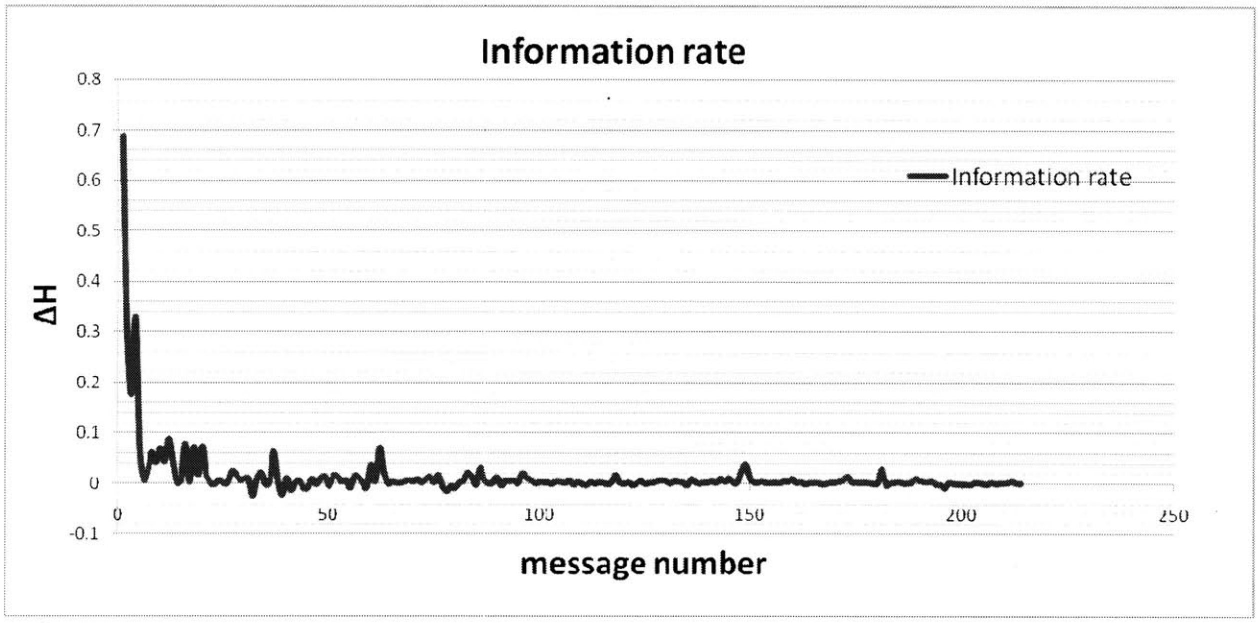

**Figure 14 – Change in entropy (novel information) for the outbox of an individual.**

As described in Section 4.3, amount of novel information in a message is equal to the difference between its entropy and the shared information between that message and all the previous messages. To account for increasing size of messages in this case, we multiply the amount of novel information calculated from this method by the total number of messages seen so far. Finally, we calculate the average of this value for each person. Effectively, this provides us with the amount of novel information per message. Figure 15 shows an example of the average novel information of ten individuals based on the contents of their different folders.

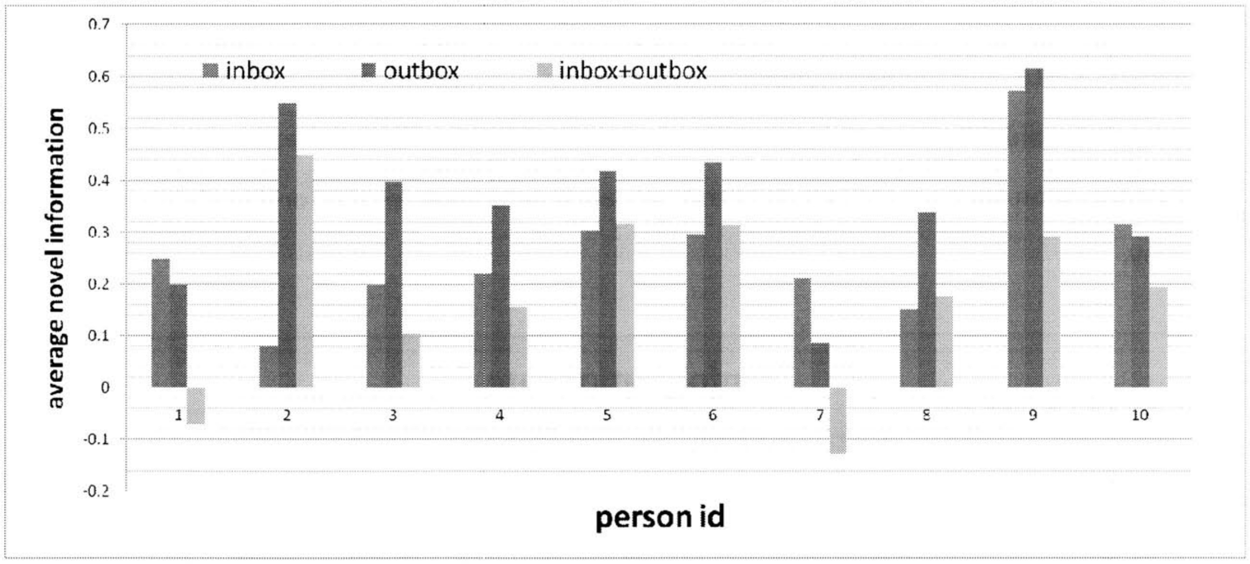

**Figure 15 - Average amount of novel information for different individuals**

In terms of average novel information per message, each individual exhibits different behaviors. In our example, two employees (number 1 and 7) have negative novel information in their collective boxes although the values of novel information for each one of their separate folders are positive. Employees number 2 and 9 on the other hand exhibit higher values of novel information for their outbox folder. Although it is beyond the scope of this research, one may speculate about a direct relation between higher amounts of novel information in a person's outbox and their productivity. Another interesting question is whether the novel information values are related to the concept of diffusion of information within the network.

In all of the graphs, there are a few data points with negative values of novel information for one of both folders. A negative value for novel information is a result of a decrease in the average total entropy of messages over time. This could happen if the new messages someone receives (or sends in case of outbox) are very similar to the past messages. Similar patterns result in higher compression rate and lower entropy values. It must be noted that the total amount of information is a monotonically increasing function in all cases. However, the amount of information overlap between the new message and the set of old messages in this case is very high.

Figure 16 shows the distribution of novel information for all individuals with a folder size of at least 50. At the beginning of the graph on the left side, a few individuals show a high value of novel information for the combination of their folders, although none of the values are quite as high as those corresponding to the inbox or the outbox folders. This could happen if the information contained in the inbox are very different from those in the outbox, resulting in the combination of two boxes having a greater entropy value than any of the individual folders.

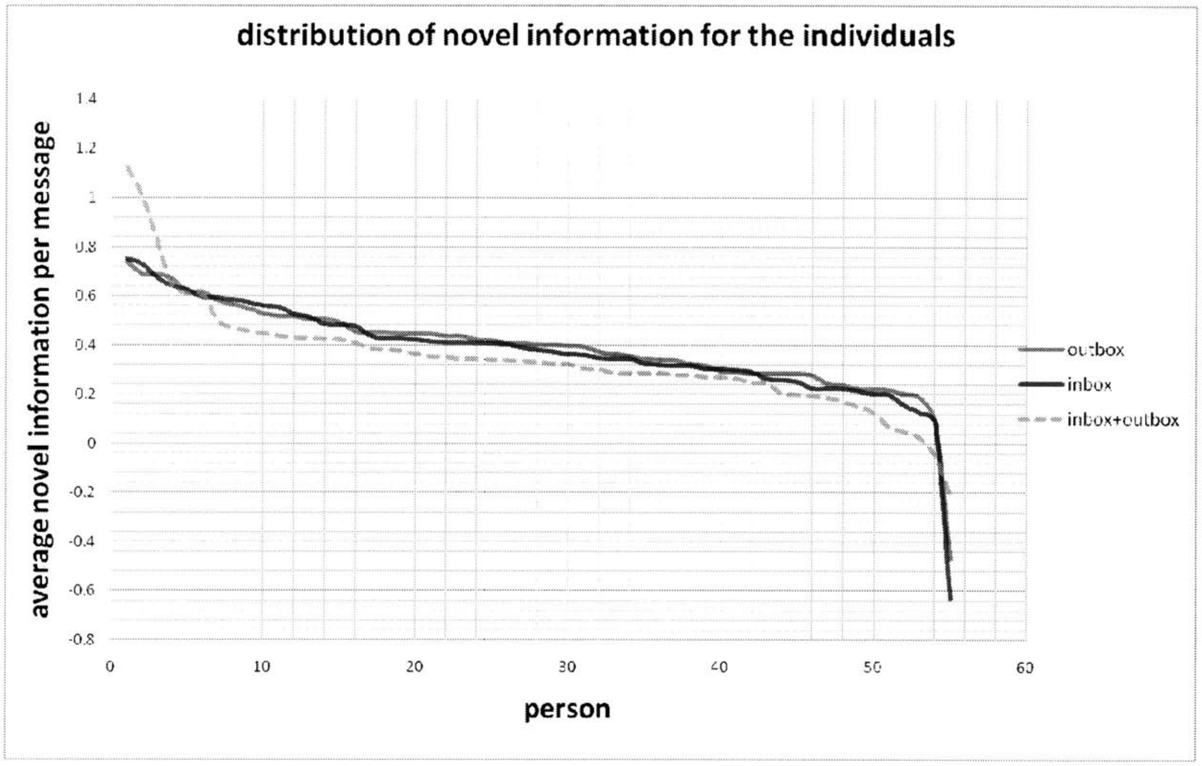

**Figure 16 - Distribution of novel information for all individuals**

Table 11 shows the summary statistics for the average novel information per message in different folders and across individuals. Since some individuals only had a few messages in their folders and based on the mean number of messages in the folders, the folders containing less than 50 messages were not included in the calculations. Otherwise, the final measures such as mean or range values would have been biased by the effect of these few individuals that hardly have any messages in their folders and considering the fact that there are only 73 individuals in the firm.

| FOLDER | NUMBER OF INDIVIDUALS | MEAN | STANDARD DEVIATION | MIN | MAX |
|---|---|---|---|---|---|
| INBOX | 63 | 0.3764 | 0.1929 | -0.6305 | 0.7502 |
| OUTBOX | 56 | 0.3913 | 0.1818 | -0.466 | 0.737 |
| INBOX+OUTBOX | 64 | 0.3526 | 0.216 | -0.2104 | 1.1221 |

**Table 11- Summary statistics for information rate measures across individuals**

The standard deviation for the novel information values are large enough to speculate that various patterns of communication exist across individuals, resulting in varying amounts of novel information. Further research will be able to use these results in conjunction with the productivity, information diversity and diffusion metrics to draw more accurate conclusions about how this may be related to information worker productivity.

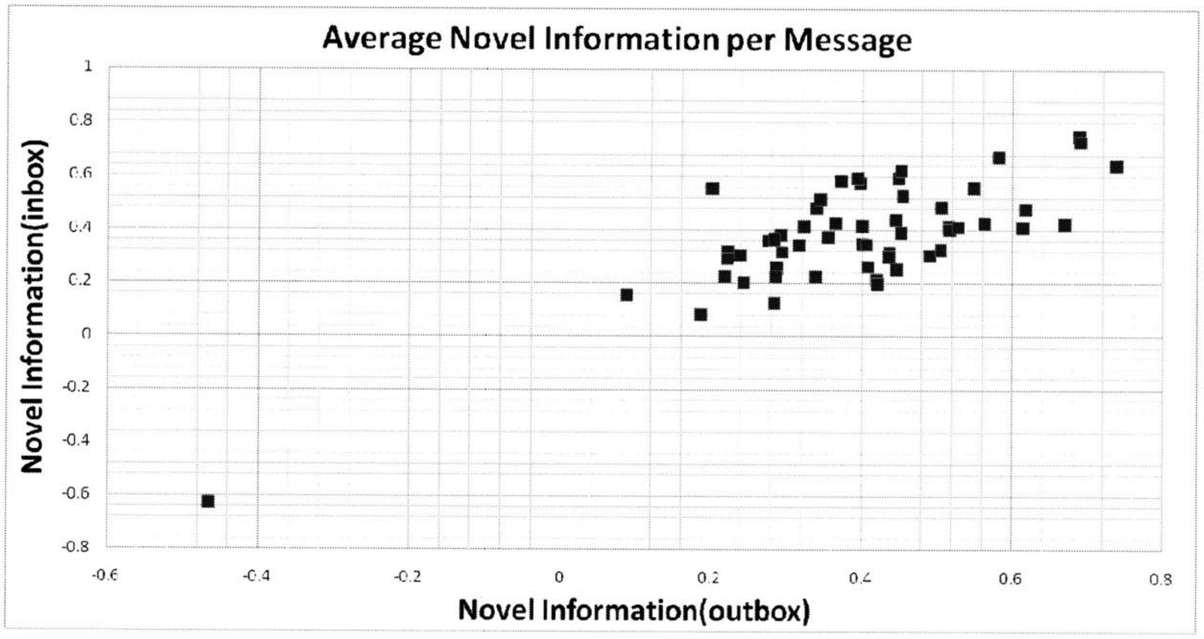

**Figure 17 - Average novel information per message, inbox and outbox folders**

Figure 17 displays the distribution of novel information values for the inbox and outbox folders across all the employees while Figure 18 displays the combination of inbox + outbox vs. the outbox folders. In these figures we considered all the employees with folders containing at least 50 messages.

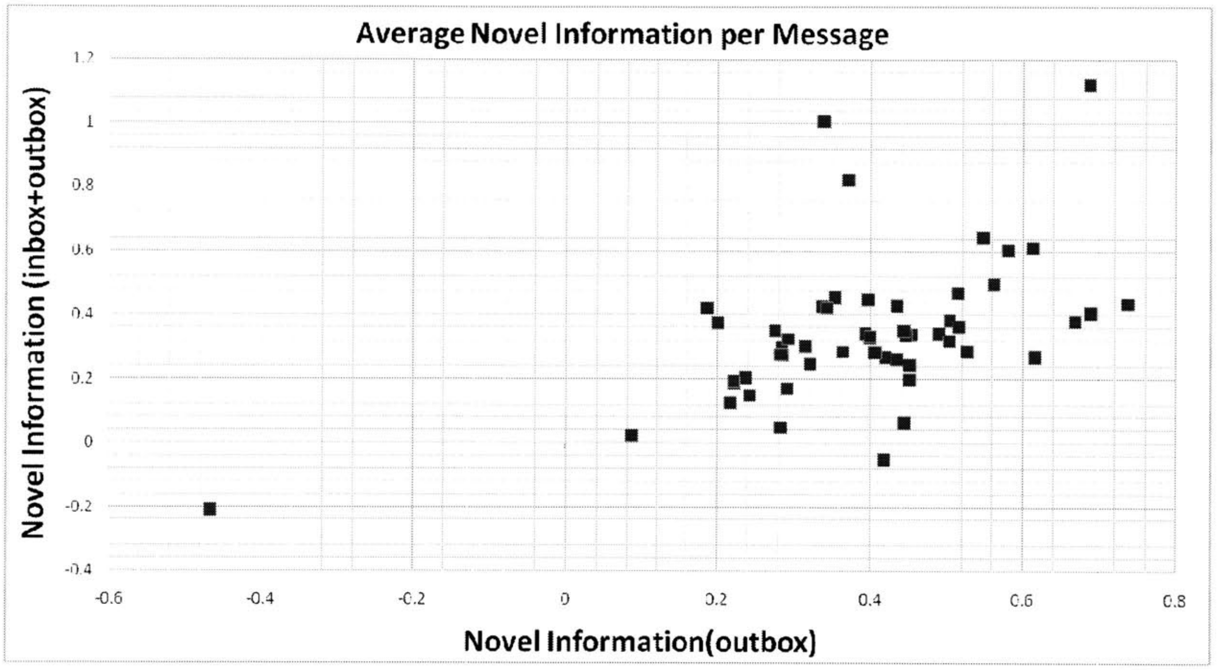

**Figure 18 - Average novel information per message, inbox+outbox vs. the outbox folder**

## *5.4. Mutual Information Measurements*

We measured the amount of information overlap between all the individuals in the firm. Each set of values for information overlap between each pair of individuals consists of three different values, the amount of mutual information between their inboxes, the amount of mutual information between their outbox folders and finally the amount of information overlap between their collective inbox and outboxes.

| ID NUMBER | OVERLAP (INBOX) | OVERLAP (OUTBOX) | OVERLAP (INBOX +OUTBOX) |
|---|---|---|---|
| 1 | 8.3222 | 5.7982 | 8.3347 |
| 2 | 10.010 | 7.9983 | 10.049 |
| 3 | 8.1740 | 0 | 8.1740 |
| 4 | 9.2657 | 7.8284 | 9.5169 |
| 5 | 9.8335 | 8.1183 | 9.9804 |
| 6 | 9.5023 | 7.9379 | 9.7593 |
| 7 | 9.9380 | 8.0214 | 10.0114 |
| 8 | 9.8697 | 8.0840 | 9.9240 |
| 9 | 9.3869 | 7.7714 | 9.3632 |
| 10 | 9.5783 | 7.9430 | 9.7832 |
| 11 | 8.9676 | 7.4094 | 9.0711 |
| 12 | 8.0647 | 0 | 8.0951 |

**Table 12 – Sample mutual information values between one person and a few other employees**

Table 12 shows a sample of mutual information values for one of the employees. The id number represents 12 other employees and each column represents the value of mutual information between our individual and the employee number 1 through 12.

As can be seen from the sample points, some individuals such as employee number 2 have a high average value of mutual information for their inbox folders, but a relatively lower value for their outboxes. This individual probably observes the same type of information that others also see, but creates messages that are more unique and specific in contents. Working on a fewer number of projects than the average, working on very specific and less common projects or not working on a team could be some of the possible reasons for observing this behavior. Further studies however, need to verify such hypotheses. Table 13 shows the summary statistics for the mutual information values across individuals and based on their respective folders.

| FOLDER | NUMBER OF INDIVIDUALS | MEAN | STANDARD DEVIATION | MIN | MAX |
|---|---|---|---|---|---|
| INBOX | 72 | 7.9629 | 2.2337 | 0 | 8.9648 |
| OUTBOX | 72 | 6.4048 | 2.7606 | 0 | 7.9341 |
| INBOX+OUTBOX | 72 | 8.2812 | 2.0560 | 0 | 9.1277 |

**Table 13- summary statistics for the average mutual information across individuals**

As expected, the mean value of mutual information of the inbox folder is relatively higher than outbox and less than the combination of both folders. This is simply due to the fact that a person usually receives much more messages than they send and therefore the probability of having higher values of information overlap increases with the number of messages.

| FOLDER | OBSERVATIONS | MEAN | STANDARD DEVIATION | MIN | MAX |
|---|---|---|---|---|---|
| INBOX | 2628 | 7.9629 | 3.2136 | 0 | 10.4501 |
| OUTBOX | 2628 | 6.4048 | 4.0717 | 0 | 10.2603 |
| INBOX+OUTBOX | 2628 | 8.2812 | 2.9464 | 0 | 10.4892 |

**Table 14 - Summary statistics for the dyadic mutual information measures**

A more thorough examination of data reveals that in general the data points are either very close to the average value or very close to zero. This happens mainly because some individuals have very few messages in one or both of their folders. Consequently the amount of mutual information will be based on only those messages. Table 14 contains the summary statistics for all the $(\frac{73*72}{2}) = 2628$ overlap points and Figure 19 displays the distribution of the pair-wise mutual information values for each one the three folders.

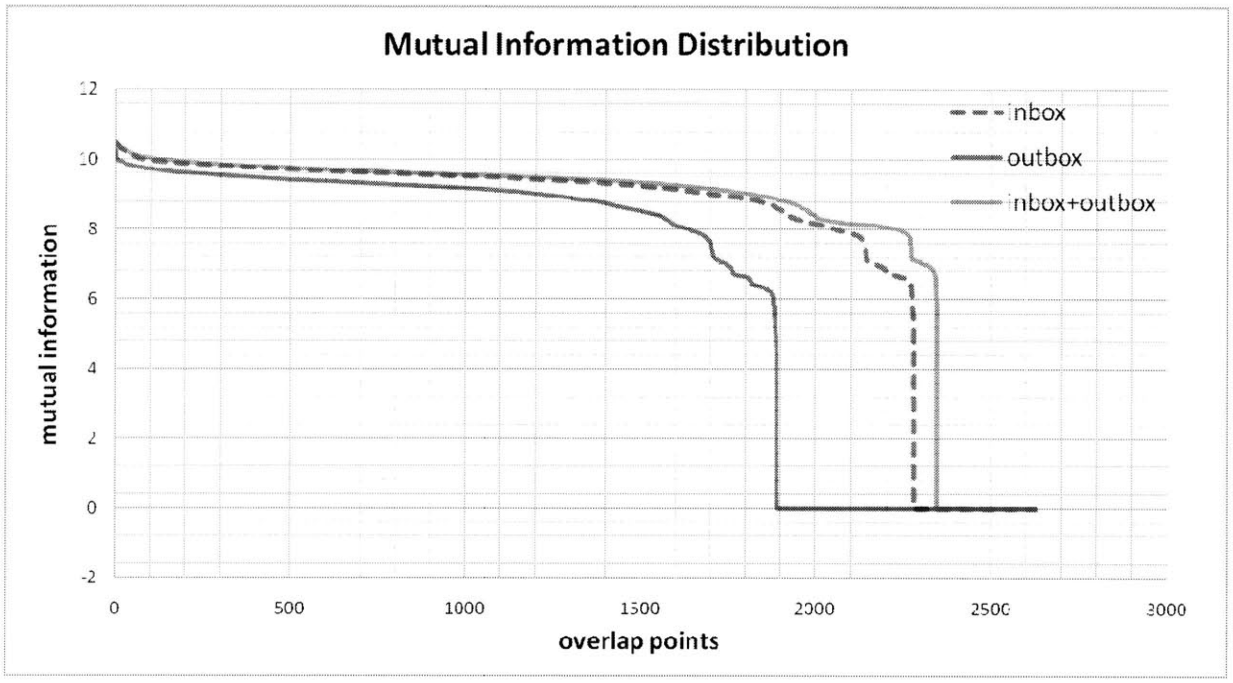

**Figure 19 - Mutual information distribution for overlap points**

The amount of overlap between the outbox folders is the lowest value from the three folders and a large portion of the overlap points lie either very close to maximum or the zero value of information overlap (portions with extremely flat slopes). The values close to zero are mainly a result of having folders containing very few, zero or only empty messages.

Figure 20 shows the distribution of mutual information values for all the employees across different folders. As expected, the mutual information value for the combination of inbox and outbox is greater than the mutual information values for the inbox or outbox of each pair. There are a total of 15768 different overlap points for all the 73 employees in the firm.

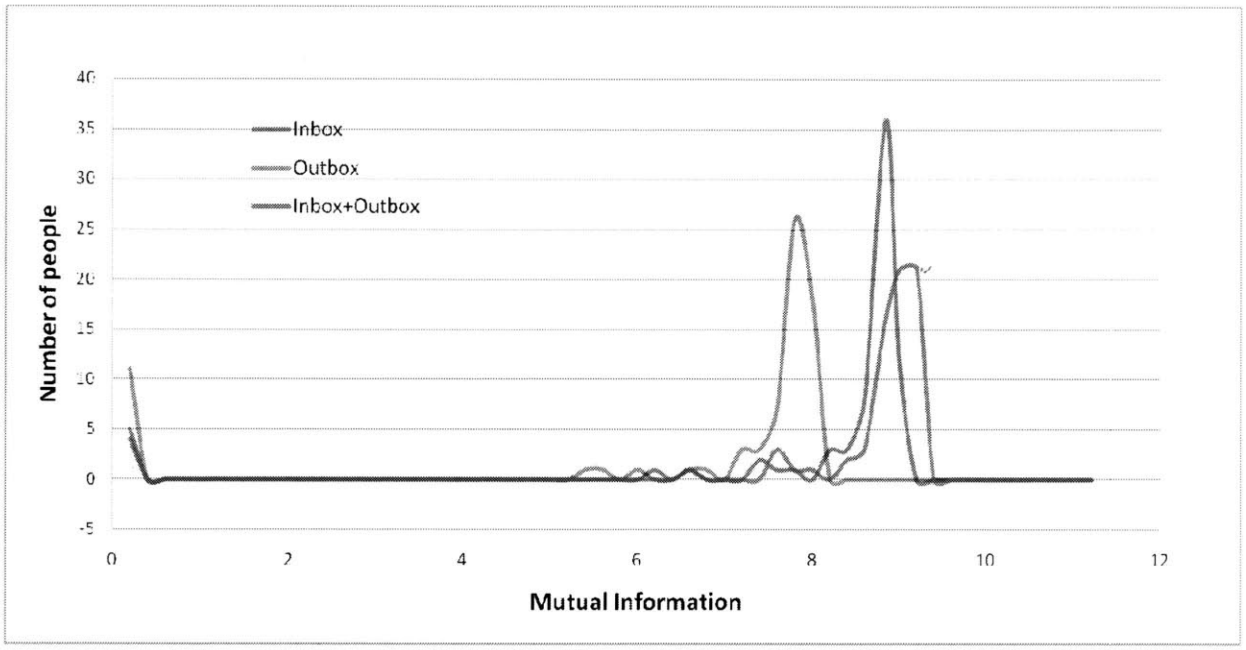

**Figure 20 - Mutual information distributions for different folders**

Interestingly, the shape of the curve corresponding to the inbox is much sharper than any of the other two folders. This could be caused by the fact that many individuals receive copies of the same messages, for instance those who work on the same team or belong to the same group. The outbox on the other hand only contains the messages that an individual sends to others which are fewer in number and could possibly be more specific in content. Therefore, the inbox messages in general seem to be more homogenous in nature than the outbox messages. If one is to focus on an individual's behavior in terms of productivity then, looking at the shared information between the outbox folders might provide a more specific indicator for that person's productive behavior rather than the contents of other folders.

## 6. Discussion and Conclusion

The objective of this study was to study, compare, analyze and implement different methods for measuring several fundamental metrics that represent the informational content of email messages. Also, we wanted to verify whether the value of quantities such as information entropy are preserved in the context of hashed messages and if so to find the exact relationship of these values to the original values extracted from plain text sources. Such verification could indicate that privacy preserving, hashed email data collection can be used in studies of the informational content of email communication. Our results advance research in these areas by demonstrating that the interests of privacy and research can be simultaneously addressed. The entropy, information rate and mutual information measurements derived from our email data set will be used in future productivity studies along with the results from social network analysis.

By using the publicly available Enron email dataset and randomly choosing a subset of those messages, we observed that the fundamental information entropy of messages is slightly changed after using the EmailNet algorithm, but that the true value could be found and that the proposed relationship between the variables have a good fit with the actual data.

Moreover, we applied a modified version of the Huffman compression algorithm to compute the entropy values for all the email messages. To verify the validity of our measurements, the actual achieved entropy values computed by the Huffman algorithm were tested against the theoretical optimal entropy bounds and shown to stay above the optimal bounds at all times.

Furthermore, by sorting the messages of each individual in chronological order, the novel information of the messages were calculated. The outcome matched our expectations in the sense that the value of the information rate over time tends to follow a natural decay rate and converges to a final value for each individual. Finally, the study has provided several sets of mutual information values between all the employees in firm based on the contents of their various email folders.

## 7. Limitations and Future Work

A fundamental question when analyzing the information content of large sets of data by using the Huffman compression algorithm is the selection of an appropriate probability model. In almost all studies on plain text, the algorithm pre-analyzes the message for determining probability. As noted by Shannon however, the information content of a word is the amount of uncertainty resolved by that word and is directly related to the expected value of observing that word.

In the context of emails however, the reader is never aware of the contents of a message beforehand. Therefore, it seems unreasonable to assume the expected value for seeing a word is equal to its probability distribution within that specific message. A more realistic assumption seems to be to calculate the probability of seeing a token in all email messages and to use it as the basis for the underlying probability model. In the context of a user's specific social network which is obviously much broader and more general than a single email message, these new sets of probabilities should better represent a recipient's real expectation of seeing a new word.

The difficulty of adopting this approach however is in the lack of standard benchmarks for validating the entropy values computed using this innovative model. For this study all the entropy values have been calculated by using the original version of the Huffman algorithm and its underlying probability model. But in the future, implementing different probability models and comparing the outcomes of the two methods may prove to provide useful insight into the true meaning of information content of email messages.

## Appendix A: Recruiting Firm Email Data Set

The email data set consists of 603,871 emails that are sent and received by the participating employees of the firm. Due to the difference between the current data set and the data set used at the time of the study, there is a clustering result for only 118,185 emails. The study was conducted before the contents of the emails are hashed to preserve the privacy of the firm and the employees. Table 15 confirms the existence of duplicated emails in the email data set. However, the duplication can be removed by eliminating additional emails with the same sender, recipients, and timestamps. Moreover, we also eliminate emails that share the same sender and timestamps, while their recipient list is a subset of the recipient list of others existing emails.

| YEAR | JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2002 | | | | | | | | 557 | 3481 | 4649 | 6020 | 5307 |
| 2003 | 6037 | 3592 | 647 | 636 | 977 | 1072 | 1501 | 1817 | 3428 | 3639 | 4149 | 4538 |
| 2004 | 5205 | 2011 | | | | | | | | | | |

**Table 15 – The number of non-duplicated internal emails by month**

This extra measure eliminates some additional duplicated emails with the special circumstance. The elimination of duplicated emails reduces the number emails in the data set to 521,316 non-duplicated emails, and the number of non-duplicated emails with bucket information is 110,979 emails. As suggested in section 3.2.1, we separate internal emails and external emails to study the effect of the inclusion of external emails. Our criterion for an internal email is that it is sent by an employee and is received by at least one employee. This way the information is circulated within the firm, and the content of the internal email is likely to be related to the work of the firm. In the email data set, there are 59,294 non-duplicated internal emails.

Table 15 shows the number of emails in each month from August 2002 to February 2004. The period between March 2003 and September 2003 contains significantly fewer emails than the other months. Both internal and external emails share the same trend, so it is not likely to be an effect of a decrease in external emails. This inconsistency is assumed to be caused by a failure of the capturing software on the corporate email server. In order not to let this inconsistency affect our analysis, the period of low email activities is excluded from the analysis. Specifically, our analysis includes emails during the periods from 1 October 2002 to 3 March 2003 and from 1 October 2003 to 10 February 2004. During the time period, there are 452,500 non-duplicated emails and 45,217 non-duplicated internal emails.

# Appendix B: Executive Recruiting Firm Database Structure

Originally, one table in the database contained the information for all the 603,871 email messages. The structure of this table is shown in Table 16. Not only the correct data types for different parameters were not used, but multiple values of data were also saved under just one field. For example, the data type used for "dateStr" variable in original schema is text (varchar) instead of the correct "datetime" format used by most databases. Each one of the "froms", "tos" and "ccs" fields may also contain multiple names instead of just one.

| Field | Type | Null |
|---|---|---|
| emailID | varchar(255) | NO |
| dateStr | varchar(100) | YES |
| subject | varchar(255) | YES |
| froms | varchar(100) | YES |
| tos | varchar(255) | YES |
| ccs | varchar(255) | YES |
| body | text | YES |
| size | int(11) | YES |
| attachNum | int(11) | YES |
| attachType | varchar(255) | YES |

**Table 16 - Original table structure**

In addition to these problems, in the 'froms', 'tos' and 'ccs' fields, several different aliases may refer to the same person. For example "John Smith" could be identified as "John.Smith" in one message, "Mr. Smith" in a second message, "john.smith@somefirm.com" in a third message, "John S." in a fourth message and so on. Although all these aliases refer to the same person, it is very hard to distinguish the different aliases of the same individual.

In the new database structure, the correct data types are used for all the variables. Additional tables are created for separating the multiple values previously stored in one field. The aliases were removed so each sender or recipient is identified by a unique global name. New tables were also added for the data contents gathered about the individuals from the previous studies. The new table names and a summary of their contents is shown in Table 17.

| TABLE NAME | DESCRIPTION |
|---|---|
| DICTNEW | GLOBAL FREQUENCIES OF HASHED VECTORS |
| EMAILS2 | ORIGINAL TABLE CONTAINING BAD DATA TYPES AND ALIASES |
| MSGS | NEW EMAILS TABLE WITH CORRECT DATE STRING BUT STILL BAD FROMS AND TOS |
| ENTROPY | ENTROPY VALUES FOR ALL 45217 NON-DUPLICATED EMAIL MESSAGES USING THE GLOBAL DICTIONARY |
| HUFFMAN | ENTROPY VALUES FOR ALL EMAIL MESSAGES USING FREQUENCY ANALYSIS |
| INFO_RATE | INFORMATION RATE FOR ALL THE INDIVIDUALS AND ALL THE EMAIL MESSAGES |
| MUTUAL_INFO | MUTUAL INFORMATION VALUES CONTAINING 72*73 ROWS AND 4 COLUMNS |
| PROJECT_MEMBERS | PROJECT IDS AND THE INITIAL OF PEOPLE WORKING ON THAT PROJECT |
| PROJECTS | VARIOUS SPEED AND DIFFUSION VALUES AND SUMMARY STATISTICS FOR PROJECTS |
| RECIPIENTS | NORMALIZED, EACH ROW CONTAINS EMAIL ID, ONLY ONE SENDER AND ONE RECIPIENT IDENTIFIED BY THEIR INITIALS FROM THE USER_NAMES TABLE |
| SPEED | DIFFERENT SPEED VALUES FOR INDIVIDUALS (IDENTIFIED BY INTIIALS) AND SUMMARY STAT |
| UIDS | NORMALIZED, SIMILAR TO RECIPIENTS, BUT CONTAINS THE SENT DATE AS WELL |
| USER_NAMES | CONTAINS ALL THE ALIASES FOR AN INDIVIDUAL IN ADDITION TO THEIR UNIQUE NAME AND INITIALS |
| VECTORS | CONTAINS VECTOR SCORES FOR ALL PAIRS OF EMPLOYEES |
| WEEKS | DATA FOR THE 41 DIFFERENT WEEKS, START AND END DATE AND A UNIQUE ID |

**Table 17 – Description of all the tables in the executive recruiting firm's database**

## Appendix C: The Implementation of the Huffman Algorithm

Implementing the Huffman compression algorithm can be summarized by the following steps: We first have to create binary tree of nodes. These nodes are stored in a set, the size of which depends on the number of symbols in the original message. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the symbol itself, the weight (frequency of appearance) of the symbol and links to their parent and children node which makes it easy to read the code (in reverse) starting from a leaf node. Internal nodes contain symbol weight, links to two child nodes and also link to a parent node. As a common convention, bit '0' represents following the left child and bit '1' represents following the right child. A finished tree has N leaf nodes and N−1 internal nodes (Wikipedia, Huffman coding, 2007).

A linear-time method to create a Huffman tree is used by creating two queues, the first one containing the initial weights (along with pointers to the associated leaves), and combined weights (along with pointers to the trees) being put in the back of the second queue. This assures that the lowest weight is always kept at the front of one of the two queues. The required steps for generating the Huffman Tree are as follows:

- Start with as many leaves as there are symbols.
- Enqueue all leaf nodes into the first queue
- Sort the first queue based on the probability of the tokens (lowest to highest)
    - While there is more than one node in the queues
        - Dequeue the two nodes with the lowest weight.
        - Create a new internal node, with the two just-removed nodes as children (either node can be either child) and the sum of their weights as the new weight.
        - Enqueue the new node into the rear of the second queue.
- The last remaining node with the highest probability value (sum of all frequencies, usually equal to one) is the root node; the tree has now been generated.

It is generally beneficial to minimize the variance of codeword length. For example, a communication buffer receiving Huffman-encoded data may need to be larger to deal with especially long symbols if the tree is especially unbalanced. To minimize variance, we simply break ties between queues by choosing the item in the first queue. This modification will retain the mathematical optimality of the Huffman coding while both minimizing variance and minimizing the length of the longest character code (Wikipedia, Huffman coding, 2007).
Four main classes were created for this implementation, namely

- HuffNode: the basic data structure for a Huffman Node, each Huffman node initially contains a keyword(if it is a leaf node), the frequency(weight) of the node, flag indicating whether it is a lead node or not and pointers to its parent and children nodes. After the tree is populated and the codewords are assigned, each node carries extra pieces of information, namely the codeword and its length.
- HuffTree: the fundamental data structure for a Huffman Tree, a Huffman Tree contains a set of Huffman Nodes, it calculates the optimum codewords for each one of the tokens within the tree. Additionally, it contains methods for calculating the lower bound and achievable entropies.
- HuffPlainTextGenerator: executable file that creates a set of HuffNodes by using character frequencies.
- HuffHashedTextGenerator: executable file that parses the hashed body messages and uses the hashed vector frequencies for the weights.

Listing 1 contains the sample Java code for the classes mentioned earlier in addition to other dependent classes containing methods for accessing the database, parsing the messages and measuring the frequencies.

```
public class HuffTree {
        private TreeSet<HuffNode> nodes;
        private HuffNode root;
        private ArrayList<HuffNode> output; // to represent the coded strings
        private double messageLength = 0;
        private double messageHuffEntropy = 0;
        private double messageTheoreticalEntropy = 0;
```

```
	private double totalFreq = 0; // to use in probabilty and entropy measurements


	public HuffTree(TreeSet<HuffNode> input){
		nodes = input;
		root = null;
		output = new ArrayList<HuffNode>();
	}

	// returns the root element
	public HuffNode buildTree(){
		if (nodes.size()<1)
			return null;
		while (nodes.size() >1){ // loop ends when there's only one element left
			Iterator<HuffNode> it = nodes.iterator();
			HuffNode first = it.next();
			it.remove();
			HuffNode second = it.next();
			it.remove();
			HuffNode combinedNode = new HuffNode(first, second);
			nodes.add(combinedNode);
		}
		assert nodes.size() == 1;
		root = nodes.first();
		return nodes.first(); // last node is the last element, only one node should be left in the tree at this point
	}

	public void generateCodes(){
		if (root == null){
			return;
		}
		root.code ="";
		generateCodesHelper(root);

	}
	private void  generateCodesHelper(HuffNode node){
		node.codeLength = node.code.length();
		if (node.codeLength > 0)
			assert node.codeLength == node.parent.codeLength + 1;

		if (node.isLeaf()){
			output.add(node); // only add it to output if it's a leaf node
			totalFreq = totalFreq + node.getCount();
			return;
		}
		node.left.code = (node.code + "0");
		node.right.code = (node.code+"1");
		generateCodesHelper(node.right);
		generateCodesHelper(node.left);
	}
	public ArrayList<HuffNode> getLeaves(){
		return this.output;
	}
	public void calcEntropy() throws IOException{
		if (root == null){
			messageHuffEntropy = 0;
			messageLength = 0;
			return;
		}
		Iterator<HuffNode> it = getLeaves().iterator();
		while (it.hasNext()){
			HuffNode current = it.next();
			messageLength = messageLength + (current.codeLength * current.getCount());
			double probabilityThisToken = ((double) current.getCount())/ ((double) totalFreq);
			messageHuffEntropy = messageHuffEntropy +
			((double)(current.codeLength) * probabilityThisToken);
```

```
				messageTheoreticalEntropy = messageTheoreticalEntropy +
				(-1 * probabilityThisToken * (Math.log10(probabilityThisToken)/Math.log10(2)));
			}

		}
		public double getMsgLength(){
			return messageLength;
		}
		public double getMsgHuffmanEntropy(){
			return messageHuffEntropy;
		}
		public double getMsgTheoreticalEntropy(){
			return this.messageTheoreticalEntropy;
		}
		public double getTotalFreq(){
			return totalFreq;
		}
}
/***************************************************************************************************/
public class HuffNode implements Comparable<HuffNode>{
		private String keyword;
		private boolean isLeaf;
		public HuffNode left;
		public HuffNode right;
		public HuffNode parent;
		public String code;
		public int codeLength; // number of bits in the code
		private int count;
		public HuffNode(String content, int instances){
			keyword = content;
			count = instances;
			left = null;
			right = null;
			parent = null;
			isLeaf = true;
		}
		public HuffNode(HuffNode child1, HuffNode child2){
			keyword = null;
			isLeaf = false;
			left = child1;
			right = child2;
			count = left.getCount() + right.getCount();
			left.parent = this;
			right.parent = this;
		}
		public int getCount(){
			return this.count;
		}
		public boolean isLeaf(){
			return this.isLeaf;
		}
		public String getKeyword(){
			return this.keyword;
		}
		public int compareTo(HuffNode otherNode) {
			if (this.getCount() == otherNode.getCount()){
				return -1;
			} else {
				return this.getCount() - otherNode.getCount();
			}
		}
}
/***************************************************************************************************/
public class Hasher {

		public static TreeMap<Character, Integer> getFreqs(String input){
			StringReader reader = new StringReader(input);
			TreeMap<Character,Integer> ans = new TreeMap<Character, Integer>();

			try {
				int thisChar= reader.read();
				while (thisChar >-1){
					Character c = new Character((char) thisChar);
					if (!ans.containsKey(c)){
						ans.put(c,new Integer(1));
					} else {
						int oldValue = (new Integer (ans.get(c)).intValue());
						ans.put(c, new Integer(oldValue+1));
					}
					thisChar = reader.read();

				}
			} catch (IOException e) {
				e.printStackTrace();
			}
			return ans;
		}
		public static void processMsg(String body, EmailConnector ec) throws SQLException{
			StringTokenizer st = new StringTokenizer(body, ";");
			while (st.hasMoreTokens()){
				String word = st.nextToken();
				int firstBracket = word.indexOf("<");
				int lastBracket = word.indexOf(">");
				if (firstBracket <1 || lastBracket <1)
					break;
				String hashed= word.substring(0, firstBracket).trim();
				String instances = word.substring(firstBracket+1, lastBracket);
				int insts = Integer.parseInt(instances);
				updateWord(hashed, insts, ec);
			}
		}
		private static boolean updateWord(String hashedWord, int numInstances, EmailConnector ec) throws SQLException{
			// let's see first if the word is already in the table;
			ResultSet rs = ec.executeQuery("SELECT * FROM "+dictTableName+ " WHERE hash ='"+hashedWord+"'");
			// if it exists get the values and update them, otherwise just INSERT it into the table
			boolean exists = rs.next();
			if (!exists){
				String insCmd = "INSERT INTO "+dictTableName+" (hash, total, totaldsc) VALUES
(\""+hashedWord+"\", "+numInstances+", 1);";
				ec.executeCommand(insCmd);
				rs.close();
				rs = null;
				return true;
			} else { //if the word exists in the table
				rs.close();
				rs = null;
				String updateCmd = "UPDATE "+dictTableName+" SET total=total+"+numInstances+", totaldsc
=totaldsc+1 WHERE hash=\""+hashedWord+"\";";
				ec.executeCommand(updateCmd);
				return true;
			}
		}

}
/*********************************************************************************************/
public class HuffmanHashedTextGenerator {
		public static void main(String[] args) {
			EmailConnector chicago = new EmailConnector(dbURL, dbName);
				boolean mustCreateHuffTbl = true;
				if (mustCreateHuffTbl){
					String createHuffcommand = "CREATE TABLE "+huffTblName+" (emailID varchar(255),
h_huffman double, h_theory double, compressed_size int, orig_size int, total_freq int,"
```

```
                                        + " CONSTRAINT pk_huffman PRIMARY KEY (emailID),"
                                        + " CONSTRAINT fk_emailID FOREIGN KEY(emailID) references msgs(emailID));";
                                        if (chicago.tableExists(dbName, huffTblName)) {
                                                System.out.println("huffman table exists. dropping it...");
                                                chicago.dropTable(huffTblName);
                                        }
                                        chicago.executeCommand(createHuffcommand);
                                }
                                int i = 0;
                                ResultSet hashedMails = chicago.executeQuery("SELECT emailID, body FROM msgs ORDER BY
sentOn");

                                while (hashedMails.next()){// iterating over all mails
                                        i++;
                                        if (i%50 == 0) System.out.println("processing email number "+i);
                                        double messageSize = 0;
                                        TreeSet<HuffNode> words = new TreeSet<HuffNode>();
                                        String body = hashedMails.getString("body");
                                        String emailID = hashedMails.getString("emailID");
                                        // get the frequencies of characters in this message (one email)
                                        StringTokenizer st = new StringTokenizer(body, ";");
                                        // looping over all the tokens in the message body for each email
                                        while (st.hasMoreTokens()){
                                                String word = st.nextToken();
                                                int firstBracket = word.indexOf("<");
                                                int lastBracket = word.indexOf(">");
                                                if (firstBracket <1 || lastBracket <1)
                                                        break;
                                                String hashed= word.substring(0, firstBracket).trim();
                                                String instances = word.substring(firstBracket+1, lastBracket);
                                                int insts = Integer.parseInt(instances);
                                                HuffNode hf = new HuffNode(hashed, insts);
                                                words.add(hf);
                                        //      System.out.println("words size = "+words.size()+ " the current word = "+
hashed + " # freqs = " +insts );
                                        }
                                        messageSize = words.size();

                                        // now build the huffman tree
                                        HuffTree hTree = new HuffTree(words);
                                        hTree.buildTree(); // build the huffman binary tree
                                        hTree.generateCodes(); // generate the codes (bits) and number of bits for each node,
also adds all the leaves to the output list in hTree
                                        hTree.calcEntropy();
                                        // now update the table;
                                        String insertHuff = "insert into "+ huffTblName + " (emailID, h_huffman, h_theory,
compressed_size, orig_size, total_freq) values (\""+
                                                emailID+"\", "+ hTree.getMsgHuffmanEntropy() + ", "+
hTree.getMsgTheoreticalEntropy()+ ", "+ hTree.getMsgLength() + ", "+ messageSize +
                                                ", "+hTree.getTotalFreq()+");";
                                        chicago.executeCommand(insertHuff);
                                }
                }
}
```

**Listing 1 - Sample code for the implementation of the Huffman compression algorithm**

## Appendix D: Enron Email Data Set

In 2003, as part of an investigation into Enron's business dealings in California, the Federal Energy Regulatory Commission made public a database containing more than 500,000 emails sent by 151 Enron employees. Subjects ranged from corporate decisions to jokes to personal matters. While the subject matter makes for intriguing reading, the entire database also proved an interesting subject for a number of researchers around the country (Enron Email Database Easy Pikcings for FastBit, 2006).

The raw Enron corpus contains 619,446 messages belonging to 158 users. Data contained large numbers of duplicate emails, which were already present in the users' other folders. In our cleaned Enron corpus, there are a total of 252,759 messages belonging to 151 users with an average of 757 messages per user. Table 18 shows the summary statistics of the messages for the Enron Email data set. The compression is performed by the Huffman algorithm using character frequencies on raw text and hashed token frequencies for hashed messages.

| | | Mean | Standard Deviation | Min | Max |
|---|---|---|---|---|---|
| Plain Text Messages | Lower bound entropy | 4.691 | 0.4712 | 0 | 5.663 |
| | Huffman entropy | 4.655 | 0.4892 | 0 | 5.695 |
| | Original size | 16420.2 | 49170.8 | 0 | 524088 |
| | Compressed size | 9297.7 | 26339.3 | 0 | 309214 |
| | Compression rate | 0.5863 | 0.0589 | 0 | 0.7119 |
| Hashed Messages | Lower bound entropy | 5.4672 | 1.5206 | 0 | 10.12 |
| | Huffman entropy | 5.5154 | 1.5174 | 0 | 10.15 |
| | Original size | 1603.904 | 4812.6 | 0 | 64688 |
| | Compressed size | 1461.855 | 5247.7 | 0 | 67681 |
| | Compression rate | 0.6908 | 0.1873 | 0 | 1.2687 |

**Table 18 – Summary statistics for entropy measurements of Enron data**

## Appendix E: Recruiting Firm Emails Entropy Measures

Table 19 contains the entropy measures for the outbox folder of all the employees in our firm. For privacy reason, the names of the individuals are omitted from the data.

| id | Huffman entropy | Lower bound Entropy | Original message size (bits) | Compressed message size (bits) | Compression rate |
|---|---|---|---|---|---|
| 1 | 6.5372 | 6.5010 | 2146.0000 | 1966.7500 | 0.8172 |
| 2 | 5.3557 | 5.3083 | 779.8905 | 622.3054 | 0.7184 |
| 4 | 5.9284 | 5.8808 | 1278.6548 | 1121.4102 | 0.7664 |
| 5 | 5.9680 | 5.9220 | 1427.3867 | 1264.0833 | 0.7825 |
| 6 | 5.4955 | 5.4469 | 851.4639 | 684.8968 | 0.6966 |
| 7 | 5.6075 | 5.5564 | 888.3200 | 719.7018 | 0.7247 |
| 8 | 5.0067 | 4.9565 | 696.0000 | 539.6667 | 0.6258 |
| 9 | 6.0533 | 6.0069 | 1431.4661 | 1245.2048 | 0.7819 |
| 10 | 5.8590 | 5.8109 | 1120.6237 | 969.0161 | 0.7696 |
| 11 | 5.8721 | 5.8232 | 1018.6780 | 849.8480 | 0.7532 |
| 12 | 6.0632 | 6.0126 | 1246.1929 | 1076.2538 | 0.7598 |
| 13 | 7.2365 | 7.1876 | 2245.3333 | 2101.6667 | 0.9046 |
| 14 | 4.6758 | 4.6072 | 461.2179 | 347.7478 | 0.5947 |
| 15 | 5.2444 | 5.2048 | 2628.9183 | 2014.8833 | 0.6962 |
| 16 | 6.5356 | 6.4880 | 1527.0605 | 1350.2724 | 0.8215 |
| 17 | 5.1585 | 5.1186 | 1126.1138 | 817.7988 | 0.7019 |
| 18 | 6.4403 | 6.3943 | 1653.1940 | 1435.4521 | 0.8066 |
| 19 | 5.6523 | 5.6102 | 1267.9111 | 1116.9238 | 0.7728 |
| 20 | 5.4912 | 5.4443 | 876.2137 | 715.8140 | 0.7220 |
| 21 | 6.1982 | 6.1495 | 1314.2274 | 1149.2575 | 0.7778 |
| 22 | 5.4649 | 5.4178 | 1144.2595 | 994.1063 | 0.7047 |
| 23 | 6.2151 | 6.1683 | 1448.1797 | 1296.8514 | 0.7871 |
| 24 | 6.0547 | 6.0097 | 1812.9762 | 1625.4985 | 0.7885 |
| 25 | 6.3221 | 6.2732 | 1280.9844 | 1104.7638 | 0.7936 |
| 26 | 6.5279 | 6.4801 | 1647.4595 | 1467.2162 | 0.8160 |
| 27 | 6.1182 | 6.0723 | 1196.8642 | 1006.5648 | 0.7671 |
| 28 | 5.3889 | 5.3398 | 874.8725 | 725.5571 | 0.6787 |
| 30 | 6.0018 | 5.9249 | 820.0000 | 658.5000 | 0.7502 |
| 31 | 5.0267 | 4.9821 | 815.8412 | 678.1191 | 0.7153 |
| 32 | 5.5867 | 5.5434 | 1157.1903 | 975.8485 | 0.7672 |
| 33 | 6.0902 | 6.0451 | 975.1287 | 806.1584 | 0.7613 |
| 34 | 5.1419 | 5.0867 | 524.0000 | 404.0217 | 0.6427 |
| 37 | 6.1381 | 6.0899 | 1204.1271 | 1025.7863 | 0.7799 |
| 38 | 5.9770 | 5.9331 | 1360.2977 | 1186.4465 | 0.7760 |

| | | | | | |
|---|---|---|---|---|---|
| 39 | 5.3672 | 5.3179 | 835.6326 | 679.9317 | 0.7174 |
| 40 | 5.4365 | 5.3902 | 1008.2902 | 846.8662 | 0.7418 |
| 41 | 5.6727 | 5.6243 | 1051.9407 | 887.9426 | 0.7449 |
| 42 | 4.8098 | 4.7628 | 711.3741 | 582.2489 | 0.6290 |
| 43 | 5.5259 | 5.4784 | 873.7518 | 708.3406 | 0.6958 |
| 44 | 4.5515 | 4.5052 | 686.4059 | 539.3173 | 0.6675 |
| 45 | 5.2010 | 5.1584 | 864.7705 | 719.0793 | 0.7427 |
| 46 | 5.5675 | 5.5176 | 799.3702 | 655.0021 | 0.7019 |
| 47 | 6.2801 | 6.2306 | 1305.8923 | 1128.6218 | 0.7878 |
| 48 | 5.8625 | 5.8133 | 1105.6375 | 927.2073 | 0.7474 |
| 49 | 5.5891 | 5.5428 | 1080.0324 | 907.8787 | 0.7205 |
| 50 | 4.6397 | 4.5985 | 838.6164 | 717.5854 | 0.7127 |
| 56 | 6.3737 | 6.3325 | 1365.8667 | 1160.7167 | 0.8102 |
| 57 | 6.0990 | 6.0521 | 1301.6552 | 1100.6379 | 0.7758 |
| 58 | 5.2838 | 5.2380 | 833.1659 | 690.6515 | 0.7073 |
| 59 | 5.2930 | 5.2478 | 998.0000 | 832.9369 | 0.6898 |
| 60 | 5.1990 | 5.1528 | 718.5536 | 572.5625 | 0.7118 |
| 61 | 5.9903 | 5.9422 | 1170.4685 | 1008.6194 | 0.7878 |
| 62 | 5.6909 | 5.6452 | 1062.4591 | 882.4833 | 0.7540 |
| 63 | 5.1985 | 5.1299 | 737.6000 | 585.0000 | 0.6498 |
| 66 | 5.8823 | 5.8409 | 1816.5455 | 1747.6970 | 0.7870 |
| 67 | 5.8409 | 5.7980 | 1997.4390 | 1890.2756 | 0.7544 |
| 68 | 6.1017 | 6.0541 | 1508.6777 | 1316.9452 | 0.7889 |
| 69 | 6.1905 | 6.1423 | 1363.1673 | 1183.6996 | 0.7784 |
| 70 | 4.8240 | 4.7751 | 723.7813 | 594.6414 | 0.6354 |
| 71 | 6.0941 | 6.0462 | 1367.0588 | 1183.9559 | 0.8062 |
| 72 | 6.2797 | 6.2339 | 1415.2395 | 1253.0076 | 0.8080 |
| 73 | 4.7074 | 4.6633 | 726.2136 | 577.8252 | 0.6966 |

**Table 19- entropy values for individuals, outbox folder**

Table 20 contains the entropy measures for the inbox folder of all the employees in our firm.

| id | Huffman entropy | Lower bound entropy | Original message size | Compressed message size | Compression rate |
|---|---|---|---|---|---|
| 1 | 5.9711 | 5.9235 | 1361.6180 | 1186.1040 | 0.7546 |
| 2 | 5.7163 | 5.6669 | 941.7584 | 765.6178 | 0.7226 |
| 3 | 5.8264 | 5.7662 | 800.7273 | 661.5455 | 0.7283 |
| 4 | 6.0817 | 6.0332 | 1299.6730 | 1132.9620 | 0.7724 |
| 5 | 6.0872 | 6.0386 | 1557.5370 | 1454.2030 | 0.7609 |
| 6 | 5.5215 | 5.4733 | 882.7905 | 706.8515 | 0.6934 |
| 7 | 5.8182 | 5.7703 | 876.6022 | 702.2634 | 0.7312 |
| 8 | 4.7349 | 4.6762 | 513.0000 | 417.6250 | 0.5919 |

| | | | | | |
|---|---|---|---|---|---|
| 9 | 6.1827 | 6.1366 | 1592.8140 | 1428.0780 | 0.7794 |
| 10 | 5.5998 | 5.5514 | 937.9896 | 789.4611 | 0.7342 |
| 11 | 6.1556 | 6.1061 | 1185.4720 | 1002.5610 | 0.7717 |
| 12 | 6.0109 | 5.9624 | 1243.2590 | 1078.1220 | 0.7514 |
| 13 | 4.3735 | 4.3086 | 400.6667 | 283.5833 | 0.5964 |
| 14 | 5.4174 | 5.3691 | 953.2033 | 801.4893 | 0.6956 |
| 15 | 5.4800 | 5.4366 | 2342.7070 | 1787.6900 | 0.6896 |
| 16 | 6.4117 | 6.3644 | 1628.8350 | 1468.3940 | 0.8026 |
| 17 | 5.3976 | 5.3533 | 1149.0670 | 827.5978 | 0.6885 |
| 18 | 6.1075 | 6.0593 | 1406.7160 | 1211.4080 | 0.7674 |
| 19 | 5.9013 | 5.8528 | 1163.5250 | 1013.1580 | 0.7416 |
| 20 | 5.7121 | 5.6642 | 1005.9680 | 844.3457 | 0.7376 |
| 21 | 6.2320 | 6.1849 | 1482.5270 | 1335.9100 | 0.7847 |
| 22 | 3.6717 | 3.6373 | 788.3990 | 699.1722 | 0.6864 |
| 23 | 6.2995 | 6.2522 | 1510.7020 | 1364.5570 | 0.7950 |
| 24 | 4.5848 | 4.5425 | 938.1997 | 810.9186 | 0.7230 |
| 25 | 6.1987 | 6.1489 | 1396.5850 | 1233.6290 | 0.7800 |
| 26 | 5.8942 | 5.8491 | 1183.2730 | 1020.5760 | 0.7598 |
| 27 | 6.0295 | 5.9833 | 1101.9740 | 921.6992 | 0.7557 |
| 28 | 5.5959 | 5.5467 | 958.1051 | 788.9678 | 0.7065 |
| 30 | 5.9633 | 5.8943 | 652.0000 | 493.2500 | 0.7454 |
| 31 | 5.0997 | 5.0557 | 879.3623 | 736.0163 | 0.7167 |
| 32 | 5.2557 | 5.1963 | 856.9159 | 715.6778 | 0.6776 |
| 33 | 5.6713 | 5.6216 | 880.2376 | 716.4356 | 0.7089 |
| 34 | 6.1379 | 6.0874 | 1186.0350 | 1033.7110 | 0.7695 |
| 37 | 6.0478 | 5.9990 | 1216.3530 | 1059.1510 | 0.7615 |
| 38 | 5.8052 | 5.7562 | 1101.3810 | 931.0468 | 0.7283 |
| 39 | 4.3583 | 4.3178 | 673.3155 | 530.8291 | 0.6667 |
| 40 | 5.5993 | 5.5521 | 963.8875 | 800.4061 | 0.7235 |
| 42 | 5.6318 | 5.5831 | 1136.9790 | 988.8347 | 0.7146 |
| 43 | 5.8397 | 5.7904 | 1039.4040 | 867.3784 | 0.7389 |
| 44 | 3.5349 | 3.5002 | 519.6101 | 407.0097 | 0.6589 |
| 45 | 5.8131 | 5.7620 | 991.0487 | 833.6140 | 0.7323 |
| 46 | 6.2390 | 6.1909 | 1448.7880 | 1298.1640 | 0.7823 |
| 47 | 6.3510 | 6.3026 | 1656.5790 | 1503.9260 | 0.7964 |
| 48 | 4.6656 | 4.6268 | 907.1106 | 758.2975 | 0.7387 |
| 49 | 5.5244 | 5.4792 | 1097.1890 | 931.0755 | 0.7488 |
| 50 | 5.9794 | 5.9302 | 1176.9130 | 1009.6810 | 0.7558 |
| 51 | 6.2497 | 6.2022 | 1458.1590 | 1286.4710 | 0.7823 |
| 52 | 6.1863 | 6.1404 | 1429.2990 | 1264.5100 | 0.7794 |
| 54 | 5.8128 | 5.7635 | 977.9782 | 815.9373 | 0.7319 |

| | | | | | |
|---|---|---|---|---|---|
| 55 | 5.0867 | 5.0390 | 854.8649 | 708.2297 | 0.6819 |
| 56 | 6.0572 | 6.0078 | 1347.4420 | 1198.2440 | 0.7661 |
| 57 | 6.2442 | 6.1951 | 1446.4340 | 1301.5220 | 0.7814 |
| 58 | 5.5598 | 5.5063 | 823.9014 | 663.0884 | 0.7079 |
| 59 | 5.3956 | 5.3485 | 1047.5580 | 868.1350 | 0.7025 |
| 60 | 5.9530 | 5.9053 | 1251.7710 | 1105.2360 | 0.7563 |
| 61 | 5.9092 | 5.8610 | 1191.0280 | 1032.8420 | 0.7508 |
| 62 | 5.8760 | 5.8256 | 1087.6000 | 910.2983 | 0.7407 |
| 63 | 6.6017 | 6.5576 | 1972.5430 | 1746.1850 | 0.8252 |
| 64 | 6.2044 | 6.1543 | 1450.6060 | 1267.8260 | 0.7797 |
| 65 | 6.4310 | 6.3870 | 1066.2860 | 915.5714 | 0.8039 |
| 66 | 5.9080 | 5.8593 | 1197.8540 | 1011.0700 | 0.7484 |
| 67 | 5.6039 | 5.5552 | 1168.8500 | 1032.7600 | 0.7059 |
| 68 | 5.4096 | 5.3616 | 1060.2910 | 931.0242 | 0.6905 |
| 69 | 5.8441 | 5.7980 | 1288.1770 | 1116.3180 | 0.7483 |
| 70 | 5.2824 | 5.2344 | 958.6681 | 835.8857 | 0.6640 |
| 71 | 5.6996 | 5.6526 | 1005.0080 | 856.9504 | 0.7235 |
| 72 | 5.8600 | 5.8118 | 1113.4550 | 954.7289 | 0.7409 |
| 73 | 5.1449 | 5.0951 | 686.5257 | 535.6179 | 0.6484 |

**Table 20 – entropy values for individuals, inbox folder**

Finally, Table 21 shows the entropy values calculated by combining the contents of both outbox and the inbox folders for each individual.

| id | Huffman entropy | Lower bound entropy | Original message size | Compressed message size | Compression rate |
|---|---|---|---|---|---|
| 1 | 5.9711 | 5.9235 | 1361.6180 | 1186.1040 | 0.7546 |
| 2 | 5.5285 | 5.4801 | 860.4158 | 693.5477 | 0.7201 |
| 3 | 5.8264 | 5.7662 | 800.7273 | 661.5455 | 0.7283 |
| 4 | 6.0210 | 5.9729 | 1292.0610 | 1129.1790 | 0.7699 |
| 5 | 6.0291 | 5.9816 | 1501.9600 | 1372.0500 | 0.7703 |
| 6 | 5.5130 | 5.4646 | 869.6047 | 697.7628 | 0.6950 |
| 7 | 5.6871 | 5.6372 | 882.4615 | 711.2264 | 0.7269 |
| 8 | 4.8090 | 4.7526 | 562.9091 | 450.9091 | 0.6011 |
| 9 | 6.1629 | 6.1166 | 1540.6050 | 1366.2550 | 0.7817 |
| 10 | 5.7816 | 5.7335 | 1037.4470 | 885.2132 | 0.7538 |
| 11 | 6.0213 | 5.9720 | 1106.6350 | 930.3896 | 0.7630 |
| 12 | 6.0472 | 5.9978 | 1254.6110 | 1087.4750 | 0.7569 |
| 13 | 4.9461 | 4.8844 | 769.6000 | 647.2000 | 0.6624 |
| 14 | 4.9010 | 4.8385 | 612.5549 | 487.3463 | 0.6252 |
| 15 | 5.5011 | 5.4582 | 2177.9130 | 1762.3060 | 0.7156 |
| 16 | 6.4751 | 6.4276 | 1580.3850 | 1411.8100 | 0.8122 |

| | | | | | |
|---|---|---|---|---|---|
| 17 | 5.3330 | 5.2907 | 1031.6170 | 777.2003 | 0.7095 |
| 18 | 6.3042 | 6.2573 | 1551.2070 | 1342.7290 | 0.7903 |
| 19 | 5.7802 | 5.7349 | 1201.8990 | 1051.3930 | 0.7540 |
| 20 | 5.6033 | 5.5559 | 937.5756 | 776.2237 | 0.7294 |
| 21 | 6.2294 | 6.1816 | 1400.3570 | 1240.8050 | 0.7830 |
| 22 | 4.4021 | 4.3635 | 878.2611 | 754.0801 | 0.7011 |
| 23 | 6.2864 | 6.2391 | 1468.8290 | 1316.5100 | 0.7917 |
| 24 | 5.3474 | 5.3035 | 1390.6400 | 1233.6400 | 0.7599 |
| 25 | 6.2434 | 6.1939 | 1359.4480 | 1192.0800 | 0.7851 |
| 26 | 6.1652 | 6.1190 | 1381.8270 | 1211.6240 | 0.7843 |
| 27 | 6.0701 | 6.0240 | 1146.4730 | 961.4972 | 0.7609 |
| 28 | 5.5072 | 5.4580 | 918.2584 | 758.5959 | 0.6932 |
| 30 | 5.9761 | 5.9045 | 708.0000 | 548.3333 | 0.7470 |
| 31 | 5.1200 | 5.0753 | 860.6660 | 718.2259 | 0.7168 |
| 32 | 5.3803 | 5.3278 | 973.2088 | 815.1484 | 0.7130 |
| 33 | 5.8766 | 5.8291 | 917.7889 | 752.9497 | 0.7346 |
| 34 | 6.0210 | 5.9699 | 1108.3470 | 959.8189 | 0.7545 |
| 37 | 6.0764 | 6.0281 | 1211.6100 | 1042.6870 | 0.7686 |
| 38 | 5.9555 | 5.9094 | 1254.9910 | 1083.1190 | 0.7598 |
| 39 | 4.8151 | 4.7709 | 750.6887 | 601.6794 | 0.6925 |
| 40 | 5.5438 | 5.4971 | 978.8860 | 816.1042 | 0.7293 |
| 41 | 5.6727 | 5.6243 | 1051.9410 | 887.9426 | 0.7449 |
| 42 | 5.1870 | 5.1392 | 900.5714 | 762.3509 | 0.6675 |
| 43 | 5.6653 | 5.6170 | 947.6836 | 779.3151 | 0.7149 |
| 44 | 3.8846 | 3.8464 | 585.2484 | 459.8049 | 0.6656 |
| 45 | 5.5625 | 5.5150 | 947.7069 | 794.7825 | 0.7371 |
| 46 | 5.9529 | 5.9041 | 1174.5660 | 1026.8640 | 0.7482 |
| 47 | 6.3175 | 6.2686 | 1490.9340 | 1326.6530 | 0.7923 |
| 48 | 5.2140 | 5.1705 | 999.8051 | 837.1468 | 0.7437 |
| 49 | 5.5814 | 5.5353 | 1091.3300 | 921.3171 | 0.7330 |
| 50 | 5.3715 | 5.3260 | 1023.4210 | 877.1509 | 0.7383 |
| 51 | 6.2497 | 6.2022 | 1458.1590 | 1286.4710 | 0.7823 |
| 52 | 6.1863 | 6.1404 | 1429.2990 | 1264.5100 | 0.7794 |
| 54 | 5.8128 | 5.7635 | 977.9782 | 815.9373 | 0.7319 |
| 55 | 5.0867 | 5.0390 | 854.8649 | 708.2297 | 0.6819 |
| 56 | 6.1932 | 6.1468 | 1368.2820 | 1197.3520 | 0.7852 |
| 57 | 6.2332 | 6.1843 | 1436.2520 | 1287.8960 | 0.7807 |
| 58 | 5.4296 | 5.3798 | 828.8880 | 676.8269 | 0.7075 |
| 59 | 5.3474 | 5.3012 | 994.8201 | 825.6054 | 0.6954 |
| 60 | 5.6953 | 5.6479 | 1050.1420 | 902.7090 | 0.7404 |
| 61 | 5.9486 | 5.9002 | 1194.3490 | 1034.5130 | 0.7690 |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 62 | 5.7747 | 5.7270 | 1075.8580 | 896.8342 | 0.7490 |
| 63 | 6.5201 | 6.4746 | 1900.7440 | 1678.6740 | 0.8150 |
| 64 | 6.2044 | 6.1543 | 1450.6060 | 1267.8260 | 0.7797 |
| 65 | 6.4310 | 6.3870 | 1066.2860 | 915.5714 | 0.8039 |
| 66 | 5.9442 | 5.8994 | 1562.9490 | 1442.0990 | 0.7722 |
| 67 | 5.7528 | 5.7076 | 1664.7920 | 1545.6050 | 0.7350 |
| 68 | 5.7872 | 5.7393 | 1294.5290 | 1132.7090 | 0.7404 |
| 69 | 6.0353 | 5.9880 | 1332.0530 | 1155.8430 | 0.7651 |
| 70 | 5.0490 | 5.0005 | 839.1819 | 713.1660 | 0.6497 |
| 71 | 5.9005 | 5.8531 | 1189.4230 | 1023.5150 | 0.7648 |
| 72 | 6.1269 | 6.0801 | 1298.3250 | 1136.7140 | 0.7826 |
| 73 | 4.9903 | 4.9425 | 691.1182 | 542.1323 | 0.6636 |

**Table 21 – entropy values for individuals, inbox and outbox folders together**

## Appendix F: Information Rate Measurements

Table 22 provides the novel information measures of the individuals in the firm. These data points only contain individuals who had at least 50 messages in both of their main folders (inbox and outbox).

| id | Novel info(Outbox) | Novel info(Inbox) | Novel info(Inbox+outbox) |
|---|---|---|---|
| 1 | 0.2857 | 0.2568 | 0.2765 |
| 2 | 0.4355 | 0.3148 | 0.4282 |
| 3 | 0.5045 | 0.4830 | 0.3842 |
| 4 | 0.4450 | 0.2504 | 0.0645 |
| 5 | 0.2008 | 0.5527 | 0.3760 |
| 6 | 0.1859 | 0.0811 | 0.4214 |
| 7 | 0.5481 | 0.5575 | 0.6434 |
| 8 | 0.2176 | 0.2209 | 0.1250 |
| 9 | 0.3540 | 0.3718 | 0.4559 |
| 10 | 0.3374 | 0.2225 | 0.4263 |
| 11 | -0.4660 | -0.6305 | -0.2104 |
| 12 | 0.2373 | 0.3030 | 0.2037 |
| 13 | 0.4182 | 0.2122 | -0.0515 |
| 14 | 0.0872 | 0.1536 | 0.0221 |
| 15 | 0.3387 | 0.4790 | 1.0053 |
| 16 | 0.2221 | 0.3174 | 0.1871 |
| 17 | 0.2915 | 0.3784 | 0.1702 |
| 18 | 0.7370 | 0.6400 | 0.4358 |
| 19 | 0.2760 | 0.3575 | 0.3514 |
| 20 | 0.3709 | 0.5820 | 0.8219 |
| 21 | 0.3215 | 0.4095 | 0.2483 |
| 22 | 0.6872 | 0.7502 | 1.1221 |
| 23 | 0.4535 | 0.5248 | 0.3387 |
| 24 | 0.4511 | 0.3888 | 0.1976 |
| 25 | 0.5146 | 0.4103 | 0.4688 |
| 26 | 0.3990 | 0.4130 | 0.3303 |
| 27 | 0.6891 | 0.7281 | 0.4076 |
| 28 | 0.2425 | 0.2003 | 0.1488 |
| 29 | 0.3968 | 0.5731 | 0.4480 |
| 30 | 0.6166 | 0.4755 | 0.2700 |
| 31 | 0.5159 | 0.4008 | 0.3638 |
| 32 | 0.4349 | 0.2974 | 0.2627 |
| 33 | 0.4070 | 0.2614 | 0.2833 |

| | | | |
|---|---|---|---|
| 34 | 0.2829 | 0.1261 | 0.0489 |
| 35 | 0.4901 | 0.3025 | 0.3433 |
| 36 | 0.4471 | 0.5930 | 0.3364 |
| 37 | 0.2214 | 0.2906 | 0.1931 |
| 38 | 0.4437 | 0.4352 | 0.3518 |
| 39 | 0.2846 | 0.2227 | 0.3000 |
| 40 | 0.6680 | 0.4193 | 0.3801 |
| 41 | 0.3927 | 0.5919 | 0.3431 |
| 42 | 0.5273 | 0.4081 | 0.2861 |
| 43 | 0.2832 | 0.3626 | 0.2775 |
| 44 | 0.3639 | 0.4243 | 0.2857 |
| 45 | 0.4514 | 0.6222 | 0.2444 |
| 46 | 0.2930 | 0.3153 | 0.3253 |
| 47 | 0.3997 | 0.3469 | 0.3340 |
| 48 | 0.4198 | 0.1955 | 0.2698 |
| 49 | 0.6135 | 0.4066 | 0.6105 |
| 50 | 0.3431 | 0.5121 | 0.4223 |
| 51 | 0.3157 | 0.3416 | 0.3029 |
| 52 | 0.5807 | 0.6720 | 0.6042 |
| 53 | 0.4055 | 0.3440 | 0.2845 |
| 54 | 0.5623 | 0.4235 | 0.4979 |
| 55 | 0.5039 | 0.3238 | 0.3204 |

**Table 22 – The novel information measurements for individuals with at least 50 messages in each folder**

## Appendix G: Mutual Information Measurements

Table 23 shows the average value of mutual information for a single individual and across their different folders.

| Id | Average mutual information of (inbox) | Average mutual information of (outbox) | Average mutual information of (inbox+outbox) |
|---|---|---|---|
| 1 | 8.7969 | 6.5175 | 9.0309 |
| 2 | 8.7203 | 7.7056 | 8.8818 |
| 3 | 7.3904 | 0 | 7.5458 |
| 4 | 8.8604 | 7.8893 | 9.0552 |
| 5 | 8.7896 | 7.7628 | 8.9570 |
| 6 | 8.5593 | 7.6045 | 8.7561 |
| 7 | 8.1754 | 7.5142 | 8.5200 |
| 8 | 7.2202 | 5.5225 | 7.5534 |
| 9 | 8.8303 | 7.7936 | 8.9857 |
| 10 | 8.6528 | 7.7201 | 8.8367 |
| 11 | 8.6975 | 7.7259 | 8.8751 |
| 12 | 8.7446 | 7.8072 | 8.9265 |
| 13 | 6.1055 | 6.6457 | 7.5422 |
| 14 | 8.5761 | 7.5963 | 8.7684 |
| 15 | 8.6247 | 7.5405 | 8.7792 |
| 16 | 8.9410 | 7.8634 | 9.1044 |
| 17 | 8.3228 | 7.3658 | 8.4730 |
| 18 | 8.7637 | 7.8636 | 8.9875 |
| 19 | 8.8693 | 7.9292 | 9.0571 |
| 20 | 8.7547 | 7.8061 | 8.9476 |
| 21 | 8.8577 | 7.8089 | 9.0054 |
| 22 | 8.5979 | 7.5306 | 8.7173 |
| 23 | 8.9197 | 7.9047 | 9.0942 |
| 24 | 8.8170 | 7.9341 | 9.0645 |
| 25 | 8.8549 | 7.7279 | 9.0162 |
| 26 | 8.4084 | 7.5202 | 8.6360 |
| 27 | 8.6013 | 7.6102 | 8.7409 |
| 28 | 8.8355 | 7.8739 | 9.0523 |
| 29 | 0 | 0 | 0 |
| 30 | 6.4165 | 5.2722 | 6.5791 |
| 31 | 8.7657 | 7.7313 | 8.9369 |
| 32 | 8.6811 | 7.6366 | 8.8149 |
| 33 | 8.3990 | 7.5768 | 8.6710 |
| 34 | 8.8021 | 7.2684 | 9.0319 |

| | | | |
|---|---|---|---|
| 35 | 0 | 0 | 0 |
| 36 | 0 | 0 | 0 |
| 37 | 8.8266 | 7.8368 | 9.0068 |
| 38 | 8.5849 | 7.7835 | 8.8161 |
| 39 | 8.6402 | 7.6369 | 8.8241 |
| 40 | 8.7801 | 7.6723 | 8.9361 |
| 41 | 0 | 7.7430 | 8.8523 |
| 42 | 8.9648 | 7.8521 | 9.1277 |
| 43 | 8.6194 | 7.6798 | 8.8295 |
| 44 | 8.1122 | 7.0671 | 8.2573 |
| 45 | 8.6895 | 7.6646 | 8.8605 |
| 46 | 8.8801 | 7.7530 | 9.0545 |
| 47 | 8.9012 | 7.7612 | 9.0505 |
| 48 | 8.5665 | 7.6776 | 8.7760 |
| 49 | 8.7921 | 7.8051 | 8.9616 |
| 50 | 8.7414 | 7.7037 | 8.9003 |
| 51 | 8.7811 | 0 | 8.9717 |
| 52 | 8.8184 | 0 | 9.0122 |
| 53 | 0 | 0 | 0 |
| 54 | 8.7832 | 0 | 8.9563 |
| 55 | 8.6728 | 0 | 8.8949 |
| 56 | 8.1679 | 7.2411 | 8.3379 |
| 57 | 8.9057 | 7.0154 | 9.1011 |
| 58 | 8.7989 | 7.8486 | 9.0128 |
| 59 | 8.6466 | 7.8146 | 8.9065 |
| 60 | 8.8709 | 7.5915 | 9.0154 |
| 61 | 8.7526 | 7.7569 | 8.9133 |
| 62 | 8.5545 | 7.6197 | 8.7323 |
| 63 | 7.7494 | 5.8471 | 7.9578 |
| 64 | 8.8439 | 0 | 9.0500 |
| 65 | 7.5163 | 0 | 7.6682 |
| 66 | 8.4102 | 7.6586 | 8.7392 |
| 67 | 8.7834 | 7.8949 | 9.0656 |
| 68 | 8.6658 | 7.8656 | 8.9793 |
| 69 | 8.8210 | 7.8547 | 9.0094 |
| 70 | 8.8209 | 7.7839 | 8.9883 |
| 71 | 8.6021 | 7.6339 | 8.7529 |
| 72 | 8.6331 | 7.7538 | 8.8513 |
| 73 | 8.2394 | 7.1633 | 8.4159 |

**Table 23 – Mutual Information measures for individuals**

## Works Cited

Aral, S., & Van Alstyne, M. (2007). Network Structure & Information Advantage: Structural Determinants of Access to Novel Information and Their Performance Implications. *Academy of Management Conference.* Philadelphia, PA.

Aral, S., & Weill, P. (2006). IT Assets, Organizational Capabilities and Firm Performance: How Resource Allocations and Organizational Differences Explain Performance Variation. *MIT Sloan Research Paper* .

Aral, S., Brynjolfson, E., & Van Alstyne, M. (2007). Productivity Effects of Information Diffusion in Networks. *International Conference on Network Sciene.* New York, NY.

Aral, S., Brynjolfsson, E., & Van Alstyne, M. (2006). Information, Technology & Information Worker Productivity: Task Level Evidence. *International Conference on Information Systems.* Milwaukee, WI.

Chen, X., Francia, B., Li, M., McKinnon, B., & Seker, A. (2004). Shared Information and Program Plagirism Detection. *Information Theory, IEEE Transactions* , 1545-1551.

Choe, T. (2006). Identifying Word Categories for Diffusion Studies in an Email Social Network. *Master of Engineering Thesis* . Cambridge, MA: Massachusetts Institute of Technology.

Corless, R. M., Gonnet, G. H., Hare, D. G., Jeffrey, D. J., & Knuth, D. E. (1996). On the Lambert W Function. *Advances in Computational Mathematics* , 329-359.

*Enron Email Database Easy Pikcings for FastBit.* (2006, April 4). Retrieved May 15, 2007, from Ernest Orlando Lawrence Berkeley National Laboratory: http://www.lbl.gov/cs/Archive/news040406.html

Huffman, D. A. (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE* , 1098-1101.

Manoharn, P. (2006, August). Diversity Measurement for the Email Content of Information Workers. *Master of Engineering Thesis* . Cambridge, MA: Massachusetts Institute of Technology.

Shannon, C. (1949). *The mathematical theory of communication.* Urbana: University of Illinois Press.

Shetty, J., & Adibi, J. (2005). Discovering important nodes through graph entropy the case of Enron email database. *3rd international Workshop on Link Discovery* (pp. 74-81). (Chicago, Il.: ACM Press.

Thomas, A. J., & Cover, T. M. (1991). *Elements of information theory.* New York : Wiley.

Van Alstyne, M., & Zhang, J. (2003). *EmailNet.* Retrieved May 5, 2007, from Executive Recruiting Online Survey: http://ipresearch.net/emailnet/

Wallace, C. S. (1999, VOL 42). Minimum message length and Kolmogorov complexity. *Computer Journal* , pp. 270-283.

*Wikipedia, Huffman coding.* (2007). Retrieved May 15, 2007, from Wikipedia: http://en.wikipedia.org/wiki/Huffman_coding

Ziv, J., & Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on* , 530-536.