The copyright of this thesis rests with the University of Cape Town. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.
A Lossy, Dictionary-based Method for Short Message Service (SMS) Text Compression

Mini-dissertation, M.Sc. Information Technology
by Wickus Martin
© 2009 All rights reserved.

Supervised by Professor Gary Marsden
This paper is accompanied by a DVD with source code, build scripts, binaries and the LD-based dictionary in text format. See readme.txt for details.
I know the meaning of plagiarism and declare that all of the work in the document, save for that which is properly acknowledged, is my own.
Abstract

Short message service (SMS) message compression allows either more content to be fitted into a single message or fewer individual messages to be sent as part of a concatenated (or long) message. While essentially only dealing with plain text, many of the more popular compression methods do not bring about a massive reduction in size for short messages. The Global System for Mobile communications (GSM) specification suggests that untrained Huffman encoding is the only required compression scheme for SMS messaging, yet support for SMS compression is still not widely available on current handsets. This research shows that Huffman encoding might actually increase the size of very short messages and only modestly reduce the size of longer messages. While Huffman encoding yields better results for larger text sizes, handset users do not usually write very large messages consisting of thousands of characters. Instead, an alternative compression method called lossy dictionary-based (LD-based) compression is proposed here. In terms of this method, the coder uses a dictionary tuned to the most frequently used English words and economically encodes white space. The encoding is lossy in that the original case is not preserved; instead, the resulting output is all lower case, a loss that might be acceptable to most users. The LD-based method has been shown to outperform Huffman encoding for the text sizes typically used when writing SMS messages, reducing the size of even very short messages and even, for instance, cutting a long message down from five to two parts.

Keywords: SMS, text compression, lossy compression, dictionary compression
Preface

This mini-dissertation concludes my studies towards the Masters in Information Technology at the University of Cape Town. The topic of SMS compression was not the original one I investigated. At first, I wrote code to send binary attachments such as PDF and MP3 files by SMS. The experiment worked, but then I had to admit how impractical such a pursuit would be; it takes a very large number of SMS messages to transfer even a tiny PDF document. I then started thinking whether to compress the attachments prior to transmission. When compressing the attachments failed to sufficiently reduce the size to make sending SMS attachments viable, I abandoned the idea completely. It was then that I started wondering about the merits of compressing plain-text SMS messages. The idea seemed worth exploring, but I still had to convince the department to let me undertake the research as part of my Masters project. Professor Gary Marsden from the department is actively involved in the field of mobile interaction. I decided to pitch my idea in the hope I might convince him to take on the role of supervisor. Professor Marsden was immediately supportive of the idea and I want to thank him for allowing me the freedom to pursue a topic of my own interest. I also want to thank my girlfriend, Allison Johnson, for the the support she gave me throughout the time I was working on the dissertation. Thank you also for proofreading my writing!

Wickus Martin
November 2009
# Table of Contents

Chapter 1 - Introduction ............................................................................................................................ 1  
  1.1 Motivation ................................................................................................................................ 1  
  1.2 Problem statement..................................................................................................................... 2  
  1.3 Dissertation outline.................................................................................................................... 2  

Chapter 2 - Background.......................................................................................................................... 3  
  2.1 SMS........................................................................................................................................... 3  
  2.2 Compression and decompression............................................................................................... 6  

Chapter 3 - Related work .................................................................................................................... 13  

Chapter 4 - Design and implementation............................................................................................. 16  
  4.1 Algorithm................................................................................................................................ 16  
  4.2 Dictionary................................................................................................................................ 17  
  4.3 Keyword substitution............................................................................................................... 22  
  4.4 The prototype........................................................................................................................... 23  

Chapter 5 - Results and discussion..................................................................................................... 26  

Chapter 6 - Conclusion....................................................................................................................... 47  

Chapter 7 - Future work....................................................................................................................... 49  

Chapter 8 - References....................................................................................................................... 50  

Appendix A - Printout Ref# 0012 ....................................................................................................... 53  
Appendix B - Printout Ref# 0043 ....................................................................................................... 55  
Appendix C - Printout Ref# 0064 ....................................................................................................... 57  
Appendix D - Printout Ref# 0179 ....................................................................................................... 59  
Appendix E - Printout Ref# 0205 ....................................................................................................... 62  
Appendix F - Printout Ref# 0353 ....................................................................................................... 65  
Appendix G - Printout Ref# 0600 ....................................................................................................... 69  
Appendix H - Printout Ref# 0723 ....................................................................................................... 75
Chapter 1 - Introduction

1.1 Motivation

Mobile SMS is an incredibly successful technology. In December 2008, there were 4 billion mobile phone subscribers [1], and a 2007 report forecasts that, by 2012, the number of SMS messages sent will total 3.7 trillion globally [2]. In addition, in 2008, more than 29 billion messages were sent in Germany alone. This is 15% more than in 2007, and the number expected in 2009 is even greater owing to the popularity of technologies such as Twitter [3]. In the UK messages totalled 41.8 billion, 56.9 billion and 78.9 billion in 2006, 2007 and 2008 respectively [4]. These figures also amount to huge profits and, in 2006 and 2008, the SMS industry globally was worth $81 and $151 billion respectively. In 2013 that figure is expected to rise to $212 billion globally [5].

The market appeal of SMS messaging is proportional to the simplicity of the technology, and it is in developing countries that SMS messaging is seeing its largest growth. Despite its success, the technology has an inconvenient limitation, namely, a 160-character-per-SMS message limit. Initially, users would type their message up to the number of allowed characters and, if the message was not yet finished, send another one. Handsets eventually automated this process, allowing users to type a long message in its entirety and send the message. Without prompting the user, the handset breaks the message into parts, labels the sequence of each part, and then transmits the parts in sequence. As the parts are received, the receiving handset processes the sequence numbers and integrates the message parts in the correct order for the recipient to view as a whole. This feature is called concatenated (or long) SMS, and while it offers usability convenience, it actually further restricts the numbers of characters that can be typed per SMS message. That is, concatenated SMS requires a special header to be sent with every message, leaving only 153 characters for the user. Users are also billed the price of a normal SMS message for every part comprising the whole. A 307-character SMS message is thus billed as three messages (153 + 153 + 1), whereas only two messages would be billed in the absence of automatic concatenation (160 + 147).

The profit margin of SMS messages is near 90%, and so the consumer is the big loser here. This forces us to consider why concatenated messages are not billed as single messages, as the way in which SMS messages are charged makes them one of the most expensive forms of data transfer in existence. In the UK, it costs 374.49 GBP to transmit just 1 MB of data via SMS at 2008 rates. This makes it 42 times more expensive than downloading the same amount of data from the Hubble space telescope for which NASA charges 8.85 GBP per megabyte [6].
1.2 Problem statement

The goal of this study is to design, implement and evaluate a dictionary-based method for compressing English SMS text messages. We would have achieved our objective if, by applying the developed compression scheme, an SMS can be transmitted and received in fewer parts than which would be required without any form of compression – the standard case for SMS transmission.

It is easy for a design to fall short in practice and to avoid this mistake, an actual, working prototype is built and tested against. Additionally, we will look at whether the method designed is a better fit for SMS compression than other known compression methods such as Huffman encoding. The basis for comparison is simple – whichever method requires the fewest parts to transmit a specific message is the more suitable candidate for SMS compression. Among the compression methods, special attention is given to Huffman encoding as it is the only method officially prescribed by the GSM specification – albeit compression in general is not supported, as far as this author could determine, on any of the leading handsets.

1.3 Dissertation outline

Chapter 2 provides background information on those aspects of SMS technology and compression relevant to this study. Chapter 3 discusses related work from literature. Chapter 4 covers the design and implementation of the compression method, the scientific approach taken to evaluate the outcome and the metrics involved. Chapter 5 presents and discusses the findings and results. Chapter 6 presents the conclusion, and Chapter 7 suggests areas and ideas for future work.
Chapter 2 - Background

2.1 SMS

Short message service (SMS) is a communication protocol for transmitting text messages over the GSM network. The technology was introduced by the European Telecommunication Standards Institute in 1991 [7]. SMS works quite simply: when a message is punched into a handset, it is sent to a short message service centre. It is the job of the SMS centre to deliver the message to the recipient or store the message for a later retry if the recipient cannot be reached, for example, when a phone is switched off. There is no guarantee that an SMS message will be delivered, but operators usually make their best attempt and only discard the message after a few days if they have no success.

The official terminology for a device that sends or receives an SMS message is a short message service entity (SMSE); this term is used to describe, for example, a personal computer, mobile phone, GSM modem or any other device that can send and/or receive SMS messages directly over the GSM network. Although SMS is a standard, required part of the GSM network, more recently the technology has been carried over to developed wireless networks, such as 3G. Messages are limited to 160 characters, however, as a remnant of the GSM implementation in which messages were transmitted using the Mobile Application Part of the SS7 signalling protocol [8], which limits payloads to 140 octets (8-bit bytes) or 1120 bits. In order to represent text characters as binary digits, a mapping set is required. This is the same method used by personal computers for storing text. Every character in the language is represented by a unique binary codeword. US-ASCII is a widely used binary alphabet (or character set) in which every Latin character is assigned a unique 7-bit pattern. US-ASCII caters mainly for English, so in order to support symbols from other languages, a much larger character set is required. There are many different implementations, but Unicode UTF-8 and UTF-16 are some of the more popular ones. The implementers of the GSM alphabet have had to deal with the same issues; it comes as no surprise then that the default alphabet for English and western European languages, called ETSI GSM 03.38, closely resembles US-ASCII and also uses 7-bit codewords [9]. Messages written in languages such as Chinese and Russian, on the other hand, are encoded using 16-bit Unicode USC-2. Depending on the language, the maximum number of characters that can be fitted into a single SMS message will be either 160 (1120 bits = 7 bits per character * 160 characters) or 70 (1120 bits = 16 bits per character * 70 characters). It is obvious then that the reason for the 160-character limit is twofold, owing to both the choice of the SS7 MAP signalling protocol and the implementation of the default character set [10]. There are advantages and disadvantages to any character set implementation.
making those easier to send. The drawback is that the longest character '0' takes 19 times longer to transmit than the shortest character 'E'. When the telegraph was mechanized, the focus switched to constant length codes whereby each character takes the same time to transmit. The smallest number that can encode 26 characters using a binary signalling system is 32 or $2^5$ (two to the power 5) which means each character can be uniquely encoded using 5 bits. The French telegraph engineer Emile Baudot invented this system in 1870. Early teleprinters commonly used the Baudot system and if the designers of SMS had considered Baudot, then 224 characters per message $(1120 / 5)$, an increase of 40%, would instead have been possible.

SMS handsets and modems can operate in either text or PDU mode. Text mode is an encoding of the underlying bit stream using GSM alphabets. PDU stands for protocol data unit and represents the way digital information is coded and structured when it is transmitted. If complete control of the data is required, then the text mode can be bypassed and the raw binary data can be directly manipulated in terms of ones and zeroes (or, more specifically, as hexadecimal strings) in PDU mode. Since binary data are verbose, PDU data are usually specified in terms of hexadecimal bytes. A normal 8-bit byte can be represented as 1111 1111, which can be written as FF in hexadecimal (1111 in binary $= 15$ in decimal $= F$ in hexadecimal).

The transmission of a message can be broken down into two steps, namely, mobile-originated (MO) and mobile-terminated (MT) services. A mobile-originated service deals with transmitting a message from the sending phone to the SMS centre, while the mobile-terminated service is concerned with transporting the message from the SMS centre to the destination phone. The SMS-SUBMIT PDU relates to the mobile-originated service, while the SMS-DELIVER PDU relates to the mobile-terminated service. When control over raw data is required, both types of PDU must be addressed. To send a message, an SMS-SUBMIT PDU must be created, and to receive an SMS message, the SMS-DELIVER PDU must be interpreted. The main point is that PDU mode allows for the manipulation of SMS in virtually any way desired. The catch, of course, is that such a message is meaningless to a receiving handset unless it can process the manipulation; otherwise, the handset simply has a message that it does not understand and cannot display to the user in a meaningful way. In order to transmit a message this way, a custom-protocol stack layered on top of the normal SMS stack that is available to both the sender and the receiver is needed. This could be done by either inserting the code into the handset firmware or by encoding the message before sending it to the GSM modem and, on receipt, instructing the GSM modem not to interpret the message but rather to hand it over to the custom stack. Ideally, the stack would be adopted in the GSM specification, which would mean handset manufacturers would support the feature natively. To first demonstrate the protocol practically, however, a custom stack must be introduced and wrapped around the GSM modem interface.
This can be done with a PC running the stack and interfacing with the GSM modem over a physical serial link. The SMS specification supports this set-up through the Hayes set of AT commands [11].

Initially, when users had to type messages exceeding the 160-character limit, they would type one message, send it, then continue the conversation by typing the next message and send that. It was not long, however, before some of the handset manufacturers started to automate the process by allowing the user to type a long message in its entirety, press send and let the handset break the message into 160-character parts to send each individually. At this stage, there was no support for reassembling the message on the receiving side. The recipient received the parts, often out of order, and opened each of the parts individually to read it. Eventually, the idea of concatenated (or long) SMS was taken up officially in the GSM specification, which meant that handset manufacturers could implement functionality in a standard way. The concatenated SMS specification stated that the handset should label each of the parts with a sequence number, that is, the equivalent of “part 2 of 3”, to allow the receiving handset to recombine the parts in the correct order in order to display the message to the user as a whole.

The solution for concatenated SMS builds on the core PDU structure and embeds the concatenated meta-information into bits that would otherwise be used for the text message payload. The concatenated meta-information (i.e. “part 2 or 3”) is incorporated as a special 6-byte header called the concatenated SMS user datagram header (UDH). The header is present in every part of a concatenated SMS message, which reduces the storage space for the text message from 1120 bits to 1072 bits (1120 – 6 bytes * 8 bits per byte). Therefore, only 153 characters (1072 / 7 bits per character) can be fitted into each part of a concatenated SMS message. The binary data making up the PDU packets are transferred in byte-high, bit-low order. In other words, the higher-order bytes are transferred first, followed by the lower-order bytes. The bytes themselves are transmitted one bit at a time, from the lowest-order bit first to the highest-order bit. For example, the hexadecimal string 02AE would be transferred as 1010111000000010 as shown in Table 2.1.

<table>
<thead>
<tr>
<th>HEX</th>
<th>BINARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE1</td>
<td>02 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>BYTE0</td>
<td>AE 1 0 1 0 1 1 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bit7</th>
<th>bit6</th>
<th>bit5</th>
<th>bit4</th>
<th>bit3</th>
<th>bit2</th>
<th>bit1</th>
<th>bit0</th>
</tr>
</thead>
</table>

Table 2.1: PDU packets are transferred in byte-high bit-low order.
2.2 Compression and decompression

Text compression is the process of making a body of printed or written work more compact in order to minimise the space required for storage or transmission. The bit or binary digit is the basic unit of information storage and transmission in digital computing. Therefore, in computer science, text compression involves encoding the content using fewer bits than the number required to represent the message in its original state [12]. However, compressed data is not immediately useful and must first be restored by reversing the changes in a process called decompression. Compression involves two main steps, namely, modelling and coding. Modelling represents derived knowledge about the subject data and typically incorporates a redundancy study [13]. The resulting model and data are then fed into a coder, which encodes the data into a more compact form by applying the model. The coding part involves feeding the model and data into an algorithm for encoding. The algorithm reads the model and uses the contained domain context to calculate the best way of compacting the data. To decompress the data, these steps are reversed in that the model and compacted data are fed into an algorithm so that they can be decoded to produce the prototype source.

When air is compressed, the degree of compression is a result of the pressure applied. To compress air into a smaller volume, more pressure must be applied; similarly there is a trade-off that must be considered when compressing text. The design and selection of a particular scheme involves considering the achievable compression ratio against various factors in terms of acceptable distortion, processing power, memory requirements and time of execution. Text is usually compressed in such a way that the original message can be reconstructed exactly. This is called lossless compression. There are situations in which it becomes perfectly acceptable to lose some of the original representation in order to achieve a higher compression ratio. The popular MP3 audio format uses a lossy compression scheme to vastly reduce the bits required to store music tracks while still offering near-CD-quality listening. This reduction is achieved by sacrificing the precision of certain sounds occurring in the range outside what most people can discern [14]. Lossless compression allows the original information to be recreated exactly, without any loss of quality or precision. The disadvantage is that the resulting compression ratio will not be as high as that which can be achieved using lossy compression, through which some of the data can be discarded. Lossless compression typically involves some form of entropy encoding, which exploits statistical redundancy to make the data more concise while preserving the original precision. The idea is to replace repeating units of the data with shorter codewords. The units that appear most frequently are replaced with the shortest codewords, and the ones that appear least often are replaced with the longest codewords. A codeword is nothing more than a sequence of bits, the length of which equals the number of bits in a sequence. The major difference between the different schemes is the way in which they produce the statistical model from
which frequencies or probabilities are determined. The schemes all employ either a static or an adaptive model. Static models are those produced entirely before the encoding stage; the encoder receives the model and does not modify it in any way. Adaptive systems work differently, as the model changes during encoding in response to analyses of the growing dataset. In an extreme case, an adaptive system might start with a perfectly trivial or empty model, yield poor results and then adapt to compact the data dynamically better by performing a single pass or multiple iterations of introspection and compression.

Lossy compression followed by decompression does not restore the original data to exactly the same state. When the data are compressed, there is a net loss of quality or precision that cannot be recovered during decompression. While lossless compression is typically used for text and computer data files, it is very common to use lossy compression for rich media such as pictures (e.g. JPEG), music (e.g. MP3) and films (e.g. MPEG-4). The main advantage of lossy over lossless encoding is that much smaller encodings can be created while still remaining useful. Sometimes the quality difference between the original and the restored data is almost imperceptible. A naïve but illustrative example of lossy compression would involve storing the number 9.99999999 as 10, which is a reduction in character count of 80%. The original precision is lost, but the approximation is close. There are two main categories of lossy compression method, namely, transform and predictive methods. Transform codes take samples from a music track, for example, break it into parts, transform these in terms of a new basis coordinate and reduce the discrete values in a process called quantisation before entropy encodes the result. In predictive lossy compression, the next target sample is predicted on the basis of a previous result. The differential between the actual and predicted result is then quantised and encoded.

The branch of information theory called rate-distortion science deals with determining the amount of precision that may be sacrificed during lossy compression without rendering the result unusable. Dictionary-based compression is among the most popular forms of lossless data compression [15]. The data are broken up into non-overlapping units called phrases, and these are then mapped to shorter bit strings called codewords. The dictionary is the tool that provides the mapping between phrases and codewords. When compressing text, the simplest implementation of this scheme would involve replacing each word with a codeword representing the index of the word in a dictionary such as the Oxford English Dictionary. The representation of verbal information as single numbers can optimize main storage, peripheral storage, and data transmission [16]. In a large dictionary however, the index for a word can consist of more digits than the word it represents has characters. However, this could be offset somewhat by the advantage that words at the start of the dictionary have indices shorter than their corresponding words. The other drawback of this method is the sheer size of the
dictionary. If the dictionary were to be encoded along with the message so that the recipient would have it to decompress the text, then the compressed message might be significantly larger than the original input. Considering that the Oxford English Dictionary contains 59 million words and requires 540 megabytes of storage space, it becomes obvious why this approach is impractical. However, since the dictionary is static and not based on the content of the message, another option is to omit it from the encoded form if an agreement can be reached on the recipient having the dictionary on hand. However, there are methods that are more sophisticated than using words as the phrases to replace. The problem with words involves the sheer number of possibilities; encoding characters, for instance, reduces the size of the set significantly, and some dictionary-based methods are predicated on this approach. In fact, this is the single method used to represent text on computers. Computers only understand binary digits and cannot, for example, make sense of the concept “a” or “b” natively. The US ASCII Character Set, for example, uses a dictionary of 7-bit patterns to represent every character in the English alphabet. Larger sets such as Unicode UTF-8 use the same dictionary design and incorporate symbols not only from English but from all modern written languages by allowing each symbol to be represented by anything from 8 to 32 bits. This is a form of encoding, however, and not compression. Compression of a character would require representing that character with fewer bits than those required using the bit pattern from its character set. Dictionary size and the speed at which lookups can be made are the two trade-offs between cost and latency.

Huffman coding is a form of lossless, dictionary-based data compression using entropy theory based on the assumption that the input data consist of some symbols, be they characters or bit sequences, which occur more frequently than others [17]. Data that satisfy this assumption can be compressed quite well, while those that do not are better suited to other compression means. The assumption does, however, hold true for most text files and raw images. The algorithm first scans the data input, then identifies repeating symbols, and finally organises these into a frequency table sorted from most to least frequent. The entries from the table are then organised into a binary tree, the purpose of which is to derive a unique bit sequence of variable length for each of the symbols. The bottom-up layout of the tree ensures that the most frequently used symbols are assigned the shortest bit sequences, while the less frequent ones have longer sequences. The algorithm then makes a second pass over the original data, replacing each of the symbols with the bit sequence derived from the tree before also storing the frequency table in a compression header. During decompression, the frequency table is read from the header, and then, just as in compression, a tree is created to identify the mapping between bit sequences and symbols. The compressed data is then scanned, and each of the bit sequences is replaced with the corresponding symbol from the tree so as to arrive at the original data exactly. The most obvious question regarding this method of compression is why it is necessary to re-create the tree from the frequency table stored in the compression header rather than
just storing the tree instead of the table. The answer is that the bit sequences assigned to the symbols would take up more real estate in terms of storage than the frequency of each symbol. The main concept on which Huffman encoding is based is very simple and not even unique; it represents symbols with codewords that are shorter than their original encoding allows based on the frequency of the symbols. The cleverness of the Huffman method rests in the use of a bottom-up tree to find shorter codewords. It is for this reason that, even though the frequency table is stored in the compression header, it is known by a rather generic name, while the tree is crowned with the inventor's name, that is, the Huffman tree. A simple example can be used to illustrate Huffman encoding. Table 2.2 shows the frequency table for the text "aaaaaabbbbcc".

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>6</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
</tr>
</tbody>
</table>

*Table 2.2: Huffman frequency table for "aaaaaabbbbcc".*

The next step is to build the Huffman tree. There are different ways to explain the algorithm: two instances being mathematical and logical. The logical method described next is easier to understand. Initially, there are no parents, only disconnected leaf nodes. The tree is built by repeatedly iterating over the existing nodes and finding the two nodes that have the lowest frequency number and are not yet a parent. The two nodes, once found, are then given a common parent node that is assigned a frequency equal to the sum of that of the children. This process is repeated over and over until a single parent can be placed at the top of the tree, unifying all the branches. This so-called “ultimate” parent node is called the root. Figure 2.1 shows the Huffman tree in line with this example.

\[
a = \begin{array}{c}
[6]\hline
\end{array}
\]

\[
a = \begin{array}{c}
[6]\hline
\end{array} \quad \begin{array}{c}
| \\
| \\
|\hline\end{array} \quad \begin{array}{c}
[12]\hline
\end{array}
\]

\[
b = \begin{array}{c}
[4]\hline
\end{array} \quad \begin{array}{c}
| \\
| \\
|\hline\end{array} \quad \begin{array}{c}
[6]\hline
\end{array}
\]

\[
c = \begin{array}{c}
[2]\hline
\end{array}
\]

*Figure 2.1: Huffman tree for "aaaaaabbbbcc" showing frequency numbering.*
The next step is to traverse the tree from the root to each of the leaves, assigning a 1 to the path along every left branch and a 0 to the branch on the right as depicted in Figure 2.2.

\[
\begin{align*}
  a &= [6] \to (0) \to (0) \to [12] \\
  b &= [4] \to [6] \to (1) \\
  c &= [2] \to [6] \to (1)
\end{align*}
\]

*Figure 2.2: Huffman tree for "aaaaaabbbccc" showing numbered paths.*

The bit sequence assigned to every symbol is created by traversing the direct path from the root to the leaf assigned to the symbol and appending all of the 1s or 0s encountered along the way. Table 2.3 shows the codebook or dictionary with the substitution codes that will be used to represent each of the characters in the message. Figure 2.3 shows how the message is encoded by representing each of the characters by its corresponding codeword from the dictionary.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Bit sequence = tree path from root</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
</tr>
<tr>
<td>c</td>
<td>11</td>
</tr>
</tbody>
</table>

*Table 2.3: Huffman codebook for "aaaaaabbbccc".*

*Figure 2.3: Message encoding for "aaaaaabbbccc".*
The length of the code is inversely proportional to the estimated frequency of occurrence of the character in question; in other words, more common characters are represented by shorter strings of bits in comparison to less common characters, which employ longer bit strings. All of this is calculated using a binary tree, more commonly known as a “Huffman tree”. Thus, to take an extreme example involving an SMS text message written in English, we can expect the letter “a” to appear far more regularly than the letter “z”; likewise, an “e” will appear more frequently than an “n” but potentially not as often as a space. The corresponding bit patterns will reflect this. The estimated frequency of the occurrence of characters can be determined in one of three ways. There can be a pre-agreed and therefore fixed character distribution frequency. The advantage of this is that streams of data (in our case, characters in a text message) that comply with the fixed distribution frequency improve the overall compression rate. The disadvantage lies in the possibility that an input stream of data may emerge that deviates significantly from the pre-agreed character frequency distribution. To avoid this, there is an alternative “dynamic” Huffman encoding method. In this scenario, a coder monitors and adapts the frequency distribution based on the appearance of previous characters in the bit stream during input processing. The decoder then generates the Huffman tree, which reflects the information processed in the bit stream. Gallager first showed that a Huffman tree with a distinguished node could be converted to another Huffman tree by swapping subtrees of equal weight [18, 19]. A distinguished node is one from which every other node on the tree can be reached without a loop. The conversion can be performed in time logarithmic to the number of nodes on the tree and without requiring structural change if the count on the distinguished node were to be increased. Knuth built on this idea by producing an algorithm that maintains a Huffman tree when leaf weights are decremented or incremented [20]. Vitter improved on the algorithm further by introducing a new system for numbering nodes corresponding to their level ordering [21].

The compressed size of a message encoded with a dynamic Huffman algorithm will usually be less than that obtainable using the original static algorithm, since the coding can be different or tailored to different places in the input stream. With small amounts of data, however, such as we may expect to find in an SMS text message, this comes at a price as a result of the byte space occupied by the compressor header, which is used to generate the character frequency distribution. As a third alternative, both methods can be combined so that there is a fixed frequency distribution that can then be amended by the coder should any significant deviations arise in the data stream. This is even less suited to short messages, however, unless one reduces the number of characters within the character frequency distribution list. Those characters that are less common are assigned a “special character” status when they do appear, and the coder adds them to the frequency distribution when appropriate. McIntyre and Pechura [22] showed that, for short messages, the dynamic and semi-dynamic methods are less effective than the static method, which uses a fixed coding tree for all messages. There are
methods such as arithmetic coding [23, 24, 25, 26] which are superior [27] in most respects to Huffman coding. However, the GSM 03.42 compression algorithm for text messaging services [28] requires the implementation of raw untrained dynamic Huffman coding only. This paper proposes an alternative method to that outlined in GSM 03.42 and therefore discussion and comparison will be limited to Huffman coding.
Chapter 3 - Related work

Textspeak is an abbreviated form of slang in which proper spelling, grammar and punctuation are ignored in favour of brevity, thereby allowing users to save on keystrokes while also reducing the size of the message. According to Döring [29], the challenges of a small screen, restricted keypad and 160-character limit has encouraged the evolution of textspeak as an even more abbreviated language than that which emerged prior to that in chatrooms and virtual worlds. The associated keystroke benefits are less important now that predictive texting is widely available, but textspeak does reduce message size and, as such, can be viewed as a form of user-performed compression. For example, vowels are often removed from existing words to create abbreviations such as "txt" for "text" and "pls" for "please." Common phrases may be reduced to acronyms, so that "laugh out loud" and "be right back" become "lol" and "brb," respectively. Words are also often replaced by similar-sounding spellings. For example, "see" becomes "c" and "you" becomes "u" so that "see you" becomes "cu". The same can be done with syllables so that "great" becomes "gr8". Textspeak can greatly reduce the size of a message and speed up typing; however, although SMS provides an informal environment where mistakes are acceptable, it may not appeal to everyone or be appropriate for all situations.

However, there has clearly been a need to shorten messages and, in 1997, Vodafone undertook a study to investigate the way handsets might compress SMS messages. Vodafone's approach focused on using Huffman encoding to compress the SMS text prior to transmission and required the compression stack to be installed on the sending and receiving handsets. This method used a variation of the standard Huffman algorithm called dynamic or adaptive Huffman encoding, which compresses data as it is transmitted. The difference between the two methods is that standard Huffman encoding requires two passes over the data, whereas adaptive Huffman uses a single pass. Adaptive Huffman is faster to perform, but it is not as optimal as standard Huffman encoding in terms of the compression ratio that can be achieved. Using this method, Vodafone discovered that they could increase the number of characters per SMS message to 200. Additionally, they extended the algorithm further to incorporate the use of a static dictionary for business English. The dictionary codewords consisted of a list of keywords of up to 255 characters that could be replaced in the original text to further increase compression efficiency. With all of the options enabled, the number of characters per message was extended to 240, an increase of 50% [30]. The dictionary option supported English text only.

In 1998, the European Telecommunications Standards Institute extended the GSM specification with official support for SMS compression based on the work pioneered by Vodafone but enhanced with
further functionality. Unlike the Vodafone implementation, GSM 03.42 not only supports English but also western European languages such as German, Italian, French, Spanish, Dutch, Swedish, Danish and so on. At the time of writing, the specification has only been implemented for English and German, while support for the remaining western European languages is pending completion. The specification required only that untrained, adaptive Huffman compression be included in any implementation purporting to support compression. Character grouping is an optional feature whereby characters that are expected to appear together are grouped such that transitions are signalled not between individual characters, but between groups of characters. The net result is a smaller Huffman tree together with an increased compression ratio. The tree for the text “abcABC” would thus not contain characters for both lower-case and upper-case characters but instead for the lower-case characters only and then a special non-printable character to signal the transition to upper-case. The stream would thus be encoded as “abc[UpperCaseTransitionSignal]abc”. The savings here are not in the payload but in the tree that would be freed from storing nodes for “A and B and C”. The algorithm also optionally supports keyword substitution through the use of language-specific dictionaries. The dictionary for each language supports 128 static entries for common words. The presence of a keyword substitution is signalled with a special keyword preceding the substitution. A keyword is encoded using 10 fixed bits. If, for example, the dictionary contained only the words “this” and “sunday”, then the text “i’ll see you this sunday” would be encoded as “i’ll see you [KeyWordSignal][10 bits = this][KeyWordSignal][10 bits = sunday]”. The final option supported is punctuation processing, which, if used, distorts the text so that the message does not resemble the original exactly. Punctuation processing removes leading and trailing white spaces from the input stream and reduces otherwise redundant spaces to a single space. On encoding, the first character of every sentence is converted to lower case, and the last character of the stream is dropped if it is a full stop. On decoding, the cases are restored except for the final full stop. In other words, when punctuation processing is used, GSM 03.42 constitutes a form of lossy compression.

CleverTexting [31] is a commercial, patent-pending SMS compression scheme from an Indian company by the same name. The compression scheme is implemented as a Java midlet that is installed on sending and receiving handsets. CleverTexting was released in 2009 and can achieve a 30 to 40% increase in text length, extending the number of characters in an SMS to 224. The company website does not fully describe the compression scheme, but does explain a custom implementation is created for each of the supported languages and that the punctuation such as spaces, commas and full stops are not removed.

The Technical University of Berlin and Aalborg University in Denmark have researched compressing short messages using prediction by partial matching (PPM). PPM is categorized as a form of adaptive
statistical compression that builds a context model of the input stream to predict future symbols. The probability of each symbol is passed through an arithmetic coder to compute the compressed sequence of bits. Arithmetic coding does not replace each symbol with a code, but instead encodes the entire message into a single number which will be a fraction n where \(0.0 \leq n < 1.0\). The processor is novel in that it does not use the dictionary to store the single character arrays and their probability. Instead, it uses a single data array where each element consists of two bytes: The first byte contains a symbol count and the second a parity check byte. The single elements are accessed by a specific hash function that assigns each character array an element of the data array. Collisions are detected by a parity check of the hash function input and the parity byte of the mapped element in the data array. This data model, together with functions to compute the complete statistics of the requested symbol, form the context model. This technique increases the length of an SMS by 50 to 55%, allowing the number of characters in an SMS message to be increased from 160 characters to 320–340 characters [32,33]. The researchers have applied for a patent on the work with the help of the Patent and Contract Unit at Aalborg University and this is currently pending. The team consists of Stephan Rein, Clemens Guehmann and Frank Fitzek. Frank Fitzek co-founded Acticom GmbH in Berlin, which is a leading supplier of protocol stack software for companies such as LG Electronics, Novell and VTT and offers SMS Zipper as a commercial product based on the published work: SMS Zipper is installed as a Java application on the sending and receiving handsets, which must support J2ME. SMS Zipper uses an external model to support specific languages, which allows the model to be tailored to a specific language in order to yield the best possible results. It also allows the stack to be easily expanded to support additional languages. At the time of writing, SMS Zipper supports English, German, Danish and Italian. In [32], Rein, Guehmann and Fitzek state that, to their knowledge, their paper is the first to combine lossless short message compression with a low-complexity context modelling scheme.

The area of text compression has been well investigated, but little research has been done on its application to the domain of mobile text messaging. In addition to compressing short messages, there is the challenge of encoding the data in a way that is compliant with the GSM specification for SMS PDUs. Nakayama [34] proposes a method whereby, unlike conventional paradigms that send messages in the form of character sequences, key code sequences are used that reflect the user’s typing history to author the message. The key-code representation can be as efficient as 4 bits per key code. Experiments using the Canterbury corpus and the optimal dictionary have shown that key-code representation requires 2.95 fewer bits per character compared to the conventional GSM 03.38 representation. Using this method, each character can be encoded in 4.05 bits. After evaluating LD-based compression, we will revisit the methods discussed here to compare the resulting compression ratios.
Chapter 4 - Design and implementation

4.1 Algorithm

We propose to implement a software algorithm for compressing SMS messages. The core method of coding involves the use of a bespoke dictionary containing short keywords to represent English words. Instead of sending an SMS message as normal text, the message is processed with the English words being replaced with shorter keywords from the dictionary. In other words, if we wish to send the message "Hello World" and using the dictionary from Table 4.1, we could encode the message as "1 2".

<table>
<thead>
<tr>
<th>English Word</th>
<th>Replacement Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello</td>
<td>1</td>
</tr>
<tr>
<td>World</td>
<td>2</td>
</tr>
</tbody>
</table>

*Table 4.1: Example dictionary showing how shorter codes can be used to substitute for longer sections of text (in this case words).*

This is, of course, a very naive implementation, but it does demonstrate the fundamental concept, namely, the replacement of English words with much shorter indicators. In the above example, the number of characters was reduced from 11 (including spaces) to 3, which is a drastic reduction in size. To accomplish this type of encoding, the dictionary should contain as many entries as possible. However, there will be cases in which certain words are not found in the dictionary. In such cases, the word will not be substituted but rather kept as is. If we consider the message "Hello Again World" using the same example dictionary as before, then we lack a dictionary entry for "Again". In other words, we do not have a so-called “hit” in the dictionary, and the resulting encoding will look like "1 Again 2". The reduction is not as drastic this time, but it is still significant. This means that we can encode any text message and not merely messages in which all the words are found in the dictionary. This is the fundamental basis of the coding mechanism proposed in this study. Of course, the characters making up the word "Again" would still need to be encoded according to a character alphabet. To avoid fracturing the implementation to accommodate both a word dictionary and a character alphabet, the two structures are combined so that the lossy dictionary-based dictionary contains entries for characters and words. There are also special signalling entries to indicate conditions such as "Yes, this message was encoded using our algorithm". Further discussion of the signalling entries is deferred until a later section. Additionally, the keywords are not actually textual as described above, but are rather unique binary sequences. On the receiving side, the message must be decoded
This decoder considers the input stream, parses out the symbols and then looks up each symbol against an exact copy of the dictionary that was used on the sending side. The binary symbols are replaced with their corresponding textual entries. Some of the symbols are replaced by word entries, while others are replaced by character entries. Table 4.2 and 4.3 show the steps involved to respectively compress and decompress an SMS message using the LD-based method.

<table>
<thead>
<tr>
<th>Step</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Preprocessing</td>
<td>Eliminate redundant whitespace. Eliminate non-printable characters. Convert all text to lowercase.</td>
</tr>
<tr>
<td>2</td>
<td>Parsing</td>
<td>Perform lexical analysis of input stream and parse text into words and punctuation characters.</td>
</tr>
<tr>
<td>3</td>
<td>Add compression header</td>
<td>A special compression header is constructed using a signalling entry from the dictionary.</td>
</tr>
<tr>
<td>4</td>
<td>Dictionary replacement</td>
<td>Words from the message are replaced with keywords from the dictionary. Words not found in the dictionary are encoded one character at a time using binary keywords from the dictionary.</td>
</tr>
<tr>
<td>5</td>
<td>PDU structuring</td>
<td>The binary stream of keywords is packaged according to the rules for a SMS SUBMIT PDU.</td>
</tr>
</tbody>
</table>

Table 4.2: Algorithm outline for compressing an SMS message using LD-based encoding.

<table>
<thead>
<tr>
<th>Step</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PDU extraction</td>
<td>Extract the data stream representing the message from the PDU DELIVER PDU.</td>
</tr>
<tr>
<td>2</td>
<td>Recognition</td>
<td>Inspect the first byte of the data stream to determine whether it was encoded using LD-based compression. If not, then abort further processing and treat as a normal SMS.</td>
</tr>
<tr>
<td>3</td>
<td>Dictionary replacement</td>
<td>Replace binary keywords in data stream with textual entries from dictionary. Some of the keywords will map to word entries, others to character entries.</td>
</tr>
</tbody>
</table>

Table 4.3: Algorithm outline for decompressing an LD-based SMS message.

4.2 Dictionary

In designing the dictionary, care should be taken to ensure that, during encoding, the hit ratio for matching words in the SMS message to entries in the dictionary is as high as possible. The more words there are that can be replaced with short codewords from the dictionary, the better the compression result will be. The dictionary must thus contain those words that will be used most frequently in SMS messages. One way to accomplish this would be to analyse thousands and
thousands of SMS messages and build up a statistical model indicating the frequency at which words occur. The dictionary would then be created by adding words from the statistical model, starting with the most frequently occurring words until all available slots have been filled. The problem with this approach is that it is hard to find such an archive of SMS messages. Another approach might be to take a literary corpus of written work and build the statistical model from that. There are indeed sufficient books available in the public domain to follow this approach. However, the language used in SMS messages tends not to resemble that found in books but rather is generally patterned like the informal flow of spoken speech. According to Lingley [35], written speech is organised and transactional, while spoken speech is typically unplanned, less structured and interactive. The researchers in this study decided to create a statistical model using spoken speech as the training medium. In order to do so they took scripts from television shows and films, which are often freely available on the Internet. In computational linguistics, this type of statistical model is called a frequency list. In simple terms, a frequency list is a sorted list of words and their frequencies, the frequency indicates the number of occurrences in a given corpus, which is, in this case, television and film dialogue.

Table 4.4 show the ten most common words in the English language as rated by the Oxford English Corpus [36].

<table>
<thead>
<tr>
<th>Rank</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>the</td>
</tr>
<tr>
<td>2</td>
<td>be</td>
</tr>
<tr>
<td>3</td>
<td>to</td>
</tr>
<tr>
<td>4</td>
<td>of</td>
</tr>
<tr>
<td>5</td>
<td>and</td>
</tr>
<tr>
<td>6</td>
<td>a</td>
</tr>
<tr>
<td>7</td>
<td>in</td>
</tr>
<tr>
<td>8</td>
<td>that</td>
</tr>
<tr>
<td>9</td>
<td>have</td>
</tr>
<tr>
<td>10</td>
<td>I</td>
</tr>
</tbody>
</table>

Table 4.4: Ten most common words according to the Oxford English Corpus.

A similar study was performed by the open content group Wiktionary, using collections of TV and film scripts and transcripts mainly downloaded from the Internet. Table 4.5 shows the most common words as indicated by Wiktionary [36].
There is very little difference between the lists, and significantly, there are very few differences when comparing the 100 most common words, as the same words appear with only slightly different ranks. For instance, the words "be", "in" and "have" are missing from the Wiktionary top-10 list but appear instead ranked at 25, 13 and 18, respectively. The researchers used the frequency lists from the Wiktionary project as the basis for constructing the dictionary only because the entries are expected to align slightly better with regular spoken dialogue. The Wiktionary project counted 29,213,800 words of transcript dialogue. Hyphenated words were broken down so that, for example, "happy-juice" was counted as "happy" and "juice." Apostrophes were stripped from words, unless they were entirely contained within word characters. For instance, "'cause" would be counted as "cause", but "don't" would be counted as "don't". All of the words were converted to lower case, so "He" and "he" would both be counted as "he". The only exception is "I", which always appeared in upper case. Verbal expressions such as "phew" and "brr" were counted, but they only entered the lower-frequency end of the list.

There are three types of entry in the dictionary, namely, signalling symbols, character symbols and word symbols. Signalling systems indicate special processing conditions. For instance, the decoder algorithm reads the first byte of the input stream to determine if the LD-based compression instruction is set. This instruction is a signalling symbol that indicates to the decoder that the message was indeed encoded using LD-based compression and can be decompressed safely. If the signalling symbol is not set, then the input stream is treated as normal, uncompressed SMS data. The dictionary can accommodate exactly 32,768 entries. This is because the coder reads and writes data in either 8- or 16-bit chunks and uses the highest-order bit of the highest-order byte of every chunk to indicate the size of the chunk. An 8-bit chunk would be stored as 0xxxxxxx, and a 16-bit chunk would be

<table>
<thead>
<tr>
<th>Rank</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>you</td>
</tr>
<tr>
<td>2</td>
<td>I</td>
</tr>
<tr>
<td>3</td>
<td>to</td>
</tr>
<tr>
<td>4</td>
<td>the</td>
</tr>
<tr>
<td>5</td>
<td>a</td>
</tr>
<tr>
<td>6</td>
<td>and</td>
</tr>
<tr>
<td>7</td>
<td>that</td>
</tr>
<tr>
<td>8</td>
<td>it</td>
</tr>
<tr>
<td>9</td>
<td>of</td>
</tr>
<tr>
<td>10</td>
<td>me</td>
</tr>
</tbody>
</table>

Table 4.5: Ten most common words as indicated by Wiktionary.
stored as 1xxxxxxxxxxxxxxxx. If we bear in mind that the value represented in 8 bits by 0xxxxxxxx is equivalent to the 16-bit value represented by 00000000 0xxxxxxx, we can create a continuous range of values from 00000000 00000000 to 11111111 11111111. There is a catch, however, in that the first bit cannot be used, since it must indicate the size of the chunk; this leaves 15 bit positions in the lower order. The binary number 111 1111 1111 1111 can be written as 0111 1111 1111 1111. This means that the range of slots in the dictionary will be numbered from 00000000 to 01111111 11111111, which, in decimal notation, is 0 to 32,767, for a total of 32,768 places.

Before seeding the dictionary, the Wiktionary lists were further processed by removing all words that consisted of a single character, such as "a" and "I", as such words already appear in the dictionary as single character entries instead of word entries. In the end, the dictionary takes the form illustrated in Table 4.6.

<table>
<thead>
<tr>
<th>Number in decimal</th>
<th>Symbol</th>
<th>Symbol type</th>
<th>Bits to encode</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>&lt;LD-based compression indicator&gt;</td>
<td>signal</td>
<td>7</td>
</tr>
<tr>
<td>00001</td>
<td>&lt;sp&gt;</td>
<td>character</td>
<td>7</td>
</tr>
<tr>
<td>00002</td>
<td>!</td>
<td>character</td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>00040</td>
<td>a</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>00041</td>
<td>b</td>
<td>character</td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>00070</td>
<td>you&lt;sp&gt;</td>
<td>word</td>
<td>7</td>
</tr>
<tr>
<td>00071</td>
<td>to&lt;sp&gt;</td>
<td>word</td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>00126</td>
<td>at&lt;sp&gt;</td>
<td>word</td>
<td>7</td>
</tr>
<tr>
<td>00127</td>
<td>how&lt;sp&gt;</td>
<td>word</td>
<td>7</td>
</tr>
<tr>
<td>00128</td>
<td>got&lt;sp&gt;</td>
<td>word</td>
<td>15</td>
</tr>
<tr>
<td>00129</td>
<td>there&lt;sp&gt;</td>
<td>word</td>
<td>15</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>16418</td>
<td>you</td>
<td>word</td>
<td>15</td>
</tr>
<tr>
<td>16419</td>
<td>to</td>
<td>word</td>
<td>15</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>32766</td>
<td>brr</td>
<td>word</td>
<td>15</td>
</tr>
<tr>
<td>32767</td>
<td>grr</td>
<td>word</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 4.6: Partial extract from LD-based dictionary.
The dictionary (partially shown in Table 4.6) contains only lower-case entries. This simplifies the problem of addressing different permutations of words such as "The", "THE" and "the". If all permutations were to be accommodated in the dictionary, then we would quickly run out of space and far fewer unique words could be supported. This means that the input stream has to be converted to lower case prior to encoding and that the original case will not be recovered upon decoding. It is this aspect that makes this encoding "lossy", but this is also the reason that we are able to achieve a relatively high compression rate. Furthermore, it takes 7 bits to encode a space character, as denoted by <sp> above. Since most words are followed by a space, this is a significant waste of space in that the number of spaces will usually be equal to 1 less than the number of words in the message. The dictionary contains an optimisation whereby all of the words are double encoded, once followed by a space and once without. In other words, the word "you" appears in the dictionary as both "you" and "you<sp>". This obviously cuts the number of unique words that can be accommodated in the dictionary in half, but with an sufficiently large dictionary this is not a problem. The LD-based dictionary stores more than 16,000 of the most commonly used English words in 316 Kb. The Oxford English Corpus indicates that English consists of a small number of very common words, a larger number of intermediate ones and an indefinitely long "tail" of rare terms [37]. Nation [38] estimate that a native English speaker would need to know around 5,000 words in order to read a novel written for teenagers. This type of novel was selected because it is deemed to be a good example of an accessible work of literature. In 2006, Nation ISP published the results of a very detailed study on how many words are needed for reading and writing at different skill levels (shown in Table 4.7).
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many words do you need to read a novel?</td>
<td>2,000 words for 87.83% coverage</td>
</tr>
<tr>
<td></td>
<td>4,000 words plus proper nouns for 94.8% coverage</td>
</tr>
<tr>
<td></td>
<td>9,000 words plus proper nouns for 98.24% coverage</td>
</tr>
<tr>
<td></td>
<td>proper nouns account for 1.53%</td>
</tr>
<tr>
<td>How many words do you need to read newspapers?</td>
<td>2,000 words for 83% coverage</td>
</tr>
<tr>
<td></td>
<td>4,000 words plus proper nouns for 95% coverage</td>
</tr>
<tr>
<td></td>
<td>8,000 words plus proper nouns for 98% coverage</td>
</tr>
<tr>
<td></td>
<td>proper nouns account for 4.55% to 6.12%</td>
</tr>
<tr>
<td>How many words do you need to read graded readers?</td>
<td>2,000 words for 91.20% coverage</td>
</tr>
<tr>
<td></td>
<td>2,000 words plus proper nouns for 96.75% coverage</td>
</tr>
<tr>
<td></td>
<td>3,000 words plus proper nouns for 98.86% coverage</td>
</tr>
<tr>
<td></td>
<td>proper nouns account for 5.55%</td>
</tr>
<tr>
<td>How many words do you need to know to be familiar with most words in a</td>
<td>4,000 words plus proper nouns for 96.70% coverage</td>
</tr>
<tr>
<td>children’s movie?</td>
<td>7,000 words plus proper nouns for 98.08% coverage</td>
</tr>
<tr>
<td></td>
<td>proper nouns account for 1.47%</td>
</tr>
<tr>
<td>How many words do you need to cope with an unscripted talk-back interview?</td>
<td>2,000 words for 89.41% coverage</td>
</tr>
<tr>
<td></td>
<td>3,000 words plus proper nouns for 96.52% coverage</td>
</tr>
<tr>
<td></td>
<td>6,000 words plus proper nouns for 98.26% coverage</td>
</tr>
<tr>
<td></td>
<td>7,000 words plus proper nouns for 98.62% coverage</td>
</tr>
<tr>
<td></td>
<td>proper nouns account for 1.29%</td>
</tr>
<tr>
<td>How many words do you need to cope with unscripted conversation?</td>
<td>2,000 words for 89.35% coverage</td>
</tr>
<tr>
<td></td>
<td>3,000 words plus proper nouns for 96.03% coverage</td>
</tr>
<tr>
<td></td>
<td>6,000 words plus proper nouns for 97.67% coverage</td>
</tr>
<tr>
<td></td>
<td>7,000 words plus proper nouns for 97.95% coverage</td>
</tr>
<tr>
<td></td>
<td>proper nouns account for 1.03%</td>
</tr>
</tbody>
</table>

Table 4.7: How large a vocabulary is needed for reading and listening?

The conclusion is that 8,000 to 9,000 words are needed for reading and writing and 6,000 to 7,000 words are needed for speaking and listening in order to understand 98% of content. The dictionary should thus contain at least the 9,000 most common English words in order to approach a 100% hit ratio, but in fact it contains the first 16,000 most common words.

4.3 Keyword substitution

The entries from the dictionary are all keyed against a binary pattern consisting of either 7 or 15 bits. Entries 0–127 are stored using 7 bits, and those from 128–32,767 are stored in 15 bits. During encoding, an extra bit is added just prior to writing the entry to indicate whether the entry itself is stored in 7 or 15 bits. A simplified version of the coder, which ignores signalling entries for the sake
of explanation, would encode the text "I love summer" according to the mapping shown in Table 4.8. During transmission, the mapping will be used to produce the binary stream illustrated in Figure 4.1.

<table>
<thead>
<tr>
<th>i</th>
<th>&lt;sp&gt;</th>
<th>love&lt;sp&gt;</th>
<th>summer</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>1</td>
<td>179</td>
<td>17505</td>
</tr>
</tbody>
</table>

| dictionary index (dec) | 48 | 1 | 179 | 17505 |
| dictionary index (bin) | 0110000 | 0000001 | 000000010110011 | 100010001100001 |

Table 4.8: Encoding “I love summer” with the LD-based algorithm.

0 0110000 0 0000001 1 000000010110011 1 100010001100001

Figure 4.1: Binary stream for “I love summer” rendered by LD-based algorithm.

Table 4.9 shows hows the binary stream would be decoded on the receiving side.

<table>
<thead>
<tr>
<th>Bits</th>
<th>7-bit flag</th>
<th>15-bit flag</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td></td>
<td>i</td>
</tr>
<tr>
<td>0110000</td>
<td></td>
<td></td>
<td>i</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td></td>
<td>&lt;sp&gt;</td>
</tr>
<tr>
<td>0000001</td>
<td></td>
<td></td>
<td>love&lt;sp&gt;</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
<td>summer</td>
</tr>
<tr>
<td>000000010110011</td>
<td></td>
<td></td>
<td>summer</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100010001100001</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.9: Decoding the LD-based binary stream for “I love summer”.

4.4 The prototype

A working proof of concept was created to test the theory behind the code design, as well as to help identify problems in the design. The first step was to generate the dictionary. Generating such a large codebook by hand would have been a slow, error-prone process, and so instead it was done using a combination of scripting and programming. Most of the prototype was implemented in OpenJDK 1.6 Java accompanied by Bash front-end scripts. True to the Unix philosophy, the prototype was written as a combination of various parts or libraries. Libraries, all created in Java, were created to (1) accept a text message from the command line; (2) compress the message using the LD-based routine; (3) encode the compressed message into an SMS PDU; and (4) submit the PDU to the GSM modem. A
Siemens TC35i terminal (shown in Figure 4.2) provided the GSM interface for sending and receiving SMS messages.

Figure 4.2: Workstation showing Siemens TC35i and aerial (circled).

The TC35i can be instructed using serial AT commands and is fully compliant with GSM 07.05 for SMS, which makes it straightforward to implement a driver using only GSM documentation. The terminal connects to a workstation running GNU/Linux kernel version 2.6.28-13 by way of a ch341-uart converter cable mounted on /dev/ttyUSB0. The PDU is transmitted to the terminal via port /dev/ttyUSB0 with the help of a driver that creates the necessary AT commands for pushing the PDU, as well as the open source RXTX library for serial line communication.
Figure 4.3 illustrates, at a high level what the coding and communication stack looks like for the sender and receiver. The code for the prototype is distributed along with this dissertation and includes data (i.e. novels and frequency lists) and scripts for generating the dictionary and building the prototype, as well as scripts for sending and receiving compressed SMS messages for running all the experiments. In this way, these experiments can be repeated objectively in order to generate reports on individual experiments, mine data from various experiment reports and present findings in a tabular form. There are also scripts for parsing the tabular data and generating Gnuplot graphs. Essentially, all the experiments and findings presented in the dissertation can be repeated in order to verify the results, study the source code and verify the logic involved.
Chapter 5 - Results and discussion

Public domain texts (shown in table 5.1) were taken from Project Gutenberg and then analysed against the dictionary to determine how much of the content is covered. By scanning every word in the text and tabulating the number of words that also exist in the dictionary, we are able to describe the coverage in terms of the hit ratio whereby a "hit" constitutes a word from the text that also exists in the dictionary. The public domain texts were analysed as is, in other words, unaltered from the original downloaded copies, which meant they included notes not present in the original printing such as disclaimers and terms of use added later by Project Gutenberg.

<table>
<thead>
<tr>
<th>Text</th>
<th>% unique hit ratio</th>
<th>% coverage hit ratio</th>
<th>Bits per character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dracula</td>
<td>58.16</td>
<td>90.14</td>
<td>4.01</td>
</tr>
<tr>
<td>Emma</td>
<td>59.69</td>
<td>91.67</td>
<td>4.03</td>
</tr>
<tr>
<td>Frankenstein</td>
<td>60.20</td>
<td>88.30</td>
<td>4.11</td>
</tr>
<tr>
<td>Monte Cristo</td>
<td>44.95</td>
<td>90.75</td>
<td>4.19</td>
</tr>
<tr>
<td>Origin of Species</td>
<td>45.75</td>
<td>86.23</td>
<td>4.28</td>
</tr>
<tr>
<td>Paradise Lost</td>
<td>45.81</td>
<td>85.77</td>
<td>4.50</td>
</tr>
<tr>
<td>Price and Prejudice</td>
<td>60.79</td>
<td>93.34</td>
<td>4.01</td>
</tr>
<tr>
<td>War and Peace</td>
<td>43.56</td>
<td>91.71</td>
<td>4.08</td>
</tr>
<tr>
<td>War of the Worlds</td>
<td>60.29</td>
<td>86.59</td>
<td>4.26</td>
</tr>
</tbody>
</table>

Table 5.1: Evaluating the LD-based dictionary for coverage against public domain novels.

There is a subtle distinction between the unique hit ratio and the coverage hit ratio shown in Table 5.1. The unique hit ratio indicates the percentage of unique words that also appear in the dictionary, whereas the coverage hit ratio indicates the total number of words appearing in the dictionary. A strong unique ratio indicates that the dictionary has a large vocabulary sufficient to cover many words, even if some of those words might only appear once in the text. The unique hit ratio is not a good indicator of how well the dictionary will perform, as there is little benefit to including scarcely used, exotic words. The more important metric is the coverage hit ratio, as this indicates how much of the total content of the text can be encoded against the dictionary. While the unique hit ratio fluctuates greatly between the different texts, the coverage hit ratio is more regular and predictable. A text such as Darwin's Origin of the Species is expected to contain many scientific and domain-specific words, which would not constitute the larger part of a vocabulary, and this is reflected in the rather low unique hit ratio obtained. Despite being a scientific text, however, the majority of the words
appear in casual speech, and words such as "the" and "and" appear more often than the scientific ones, resulting in a sufficiently high coverage hit ratio. In other words, while the dictionary contains only 45.75% of the unique words in *Origin of the Species*, it can be used to compress 86.23% of the content. The coverage hit ratio is relatively constant, ranging from a low of 85.77% to a high of 91.71%. The compression ratio achieved ranges from 4.50-4.01 bits per character, or 35.71- 42.71%.

At first, the compression ratio achieved does not sound very impressive, and indeed it is not, since the compression algorithm is not tailored to give the best compression for large texts such as the above. A method such as GZIP, for instance, would provide a much better compression ratio. However, the compression method is more effective than GZIP or some of the other popular algorithms when applied to small texts such as the ones for which the LD-based method is intended.

Table 5.2 shows the results of compressing the full 61,313 characters of text from *Paradise Lost*. The conclusion is that LD-based compression is not well suited for compressing large datasets.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Typical file extension</th>
<th>Kilobytes</th>
<th>% difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain US-ASCII</td>
<td>-</td>
<td>420.1</td>
<td>reference</td>
</tr>
<tr>
<td>Plain UTF-8 text</td>
<td>-</td>
<td>479.0</td>
<td>+14.02</td>
</tr>
<tr>
<td>BZip2</td>
<td>bz2</td>
<td>148.0</td>
<td>-64.77</td>
</tr>
<tr>
<td>Comic Book ZIP</td>
<td>cbz</td>
<td>196.2</td>
<td>-53.30</td>
</tr>
<tr>
<td>Gnu Zip</td>
<td>gzip</td>
<td>196.1</td>
<td>-53.32</td>
</tr>
<tr>
<td>Lempel-Ziv-Markov chain-Algorithm</td>
<td>lzma</td>
<td>167.1</td>
<td>-60.22</td>
</tr>
<tr>
<td>Zip</td>
<td>zip</td>
<td>196.2</td>
<td>-53.30</td>
</tr>
<tr>
<td>Huffman</td>
<td>-</td>
<td>216.5</td>
<td>-48.46</td>
</tr>
<tr>
<td>LD-based</td>
<td>-</td>
<td><strong>263.2</strong></td>
<td>-37.35</td>
</tr>
</tbody>
</table>

*Table 5.2: Compression results for the epic poem Paradise Lost.*

Previously, compression was measured using non-standard files, which raised the possibility that the file selected might unfairly favour the particular algorithm being tested. In order to compare the compression results from two different algorithms, the same baseline should be used to ensure that the comparison is like for like and unbiased. The Canterbury Corpus (Table 5.3) was presented in 1997 as a replacement for the older Calgary Corpus as, after 10 years of use, the latter had started to reveal several shortcomings. The biggest problem with the Calgary Corpus was that it was compiled from a rather arbitrary body of work. The authors of the Canterbury Corpus, on the other hand, took care to ensure that the inclusion of a document in the corpus was justifiable as a means to indicate compression efficiency. The criteria for choosing the Canterbury Corpus included that 1) the
documents be representative of the type of files that would be likely compression targets in real-world situations; 2) the files be of moderate size to make distribution a non-issue; and 3) the corpus be available to all wishing to use it. This last point was addressed by limiting the corpus to documents freely available from the public domain. The Canterbury Corpus also incorporated contemporary formats such as HTML and, moreover, was distilled from a pool of 800 candidates. These candidate documents were divided into 11 predefined categories, and within each category the most representative candidate was selected based on a scatter plot of file size before and after various compression methods were applied. The file that delivered results closest to the regression line was selected from every category to become a de facto part of the corpus.

<table>
<thead>
<tr>
<th>File</th>
<th>Category</th>
<th>Description</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice29.txt</td>
<td>text</td>
<td>English text (Alice in Wonderland)</td>
<td>152,089</td>
</tr>
<tr>
<td>asyoulik.txt</td>
<td>play</td>
<td>Shakespeare (As you like it)</td>
<td>125,179</td>
</tr>
<tr>
<td>cp.html</td>
<td>html</td>
<td>HTML source</td>
<td>24,603</td>
</tr>
<tr>
<td>fields.c</td>
<td>Csre</td>
<td>C source</td>
<td>11,150</td>
</tr>
<tr>
<td>grammar.lsp</td>
<td>list</td>
<td>LISP source</td>
<td>3,721</td>
</tr>
<tr>
<td>kennedy.xls</td>
<td>Excl</td>
<td>Excel Spreadsheet</td>
<td>1,029,744</td>
</tr>
<tr>
<td>lce.t10.txt</td>
<td>tech</td>
<td>Technical writing</td>
<td>426,754</td>
</tr>
<tr>
<td>plrabn12.txt</td>
<td>poem</td>
<td>Poetry</td>
<td>481,861</td>
</tr>
<tr>
<td>ptt5</td>
<td>fax</td>
<td>CCITT test set</td>
<td>513,216</td>
</tr>
<tr>
<td>sum</td>
<td>SPRC</td>
<td>SPARC Executable</td>
<td>3,824</td>
</tr>
<tr>
<td>xargs.1</td>
<td>man</td>
<td>GNU manual page</td>
<td>4,227</td>
</tr>
</tbody>
</table>

Table 5.3: Contents of the Canterbury Corpus.

The LD-based algorithm is designed specifically for compressing English text, and so we will focus exclusively on the compression of alice29.txt.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Bits per character</th>
</tr>
</thead>
<tbody>
<tr>
<td>bred-r3</td>
<td>2.55</td>
</tr>
<tr>
<td>ppmD5</td>
<td>2.20</td>
</tr>
<tr>
<td>szip-b</td>
<td>2.24</td>
</tr>
<tr>
<td>bzip-9</td>
<td>2.25</td>
</tr>
<tr>
<td>bzip-6</td>
<td>2.25</td>
</tr>
<tr>
<td>szip</td>
<td>2.25</td>
</tr>
<tr>
<td>ppmD7</td>
<td>2.26</td>
</tr>
<tr>
<td>bzip2-9</td>
<td>2.27</td>
</tr>
<tr>
<td>bzip2-6</td>
<td>2.27</td>
</tr>
<tr>
<td>ppmC-896</td>
<td>2.30</td>
</tr>
<tr>
<td>ppmD3</td>
<td>2.31</td>
</tr>
<tr>
<td>dmc-50M</td>
<td>2.38</td>
</tr>
<tr>
<td>dmc-5M</td>
<td>2.38</td>
</tr>
<tr>
<td>dmc-16M</td>
<td>2.38</td>
</tr>
<tr>
<td>ppmCnx-896</td>
<td>2.39</td>
</tr>
<tr>
<td>bzip-1</td>
<td>2.40</td>
</tr>
<tr>
<td>bzip2-1</td>
<td>2.42</td>
</tr>
<tr>
<td>gzip-b</td>
<td>2.85</td>
</tr>
<tr>
<td>huffword2</td>
<td>3.09</td>
</tr>
<tr>
<td>yabba-d</td>
<td>3.18</td>
</tr>
<tr>
<td>compress</td>
<td>3.27</td>
</tr>
<tr>
<td>ppmC-56</td>
<td>3.29</td>
</tr>
<tr>
<td>gzip-f</td>
<td>3.43</td>
</tr>
<tr>
<td>ppmCnx-56</td>
<td>3.57</td>
</tr>
<tr>
<td>srank-d</td>
<td>3.66</td>
</tr>
<tr>
<td><strong>LD-based</strong></td>
<td><strong>3.85</strong></td>
</tr>
<tr>
<td>gzip-d</td>
<td>3.86</td>
</tr>
<tr>
<td>char</td>
<td>4.59</td>
</tr>
<tr>
<td>pack</td>
<td>4.62</td>
</tr>
<tr>
<td>lzw1</td>
<td>4.94</td>
</tr>
<tr>
<td>yabba512</td>
<td>5.31</td>
</tr>
<tr>
<td>cat</td>
<td>8.00</td>
</tr>
</tbody>
</table>

Table 5.4: Compression ratio achieved on alice29.txt from the Canterbury Corpus.

Table 5.4 outlines the results from compressing the Canterbury Corpus. The outcome in this instance follows that from compressing *Paradise Lost*; that is, LD-based compression is not very effective for
large-sized documents. However, this coding method was not designed to be effective for such documents, and importantly, the outcome is quite different when we compare the compression results for a small text such as:

Thanks! Hope you have great Xmas too! What you up to these days? Still in London? Can't believe I've been back in NZ for nearly 18 months, but I'm still loving every minute of it! (Ref# 0179)

The above message is 179 characters long and about the length we can expect for a typical SMS message. Table 5.5 shows the results of compressing this message with some of the popular compression methods.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Typical file extension</th>
<th>Bytes</th>
<th>% difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain US-ASCII</td>
<td>-</td>
<td>157</td>
<td>reference</td>
</tr>
<tr>
<td>Plain UTF-8 text</td>
<td>-</td>
<td>179</td>
<td>+14.01</td>
</tr>
<tr>
<td>BZip2</td>
<td>bz2</td>
<td>174</td>
<td>+10.82</td>
</tr>
<tr>
<td>Comic Book ZIP</td>
<td>cbz</td>
<td>306</td>
<td>+94.90</td>
</tr>
<tr>
<td>Gnu Zip</td>
<td>gzip</td>
<td>177</td>
<td>+12.73</td>
</tr>
<tr>
<td>Lempel-Ziv-Markov chain-Algorithm</td>
<td>lzma</td>
<td>169</td>
<td>+7.64</td>
</tr>
<tr>
<td>Zip</td>
<td>zip</td>
<td>306</td>
<td>+94.90</td>
</tr>
<tr>
<td>Huffman</td>
<td>-</td>
<td>148</td>
<td>-5.73</td>
</tr>
<tr>
<td>LD-based</td>
<td>-</td>
<td>75</td>
<td>-52.22</td>
</tr>
</tbody>
</table>

Table 5.5: Result of compressing message Ref# 0179 with different methods.

The methods perform very well with larger datasets, which is the domain for which compression is typically required. That is, it aims to reduce large datasets to small ones. However, these methods do not fare so well with small datasets. The LD-based method was designed to be effective with small texts, and so it is much more effective in this range but less so for larger datasets. It is significant how much better LD-based encoding fares compared to Huffman encoding, especially since Huffman encoding is the only compression method officially supported in the SMS specification. For large datasets, we can expect Gnu Zip to outperform Huffman encoding as is evident from the experiments discussed previously, but in the case of small messages, Huffman encoding was the only standard algorithm that provided any savings at all, suggesting why it was chosen as the basis for GSM 03.42. However, the use of LD-based encoding resulted in a reduction in message size of over 50%. Notice also that, while the compression ratios achieved for methods such as Huffman and GZip encoding
vary greatly depending on the size of the message, LD-based encoding is more consistent. When novels were compressed, the average reduction in size was around 40.52%.

As Huffman compression is the only method supported in the GSM specification, we are especially interested in comparing LD-based and Huffman encoding. To this aim, eight short messages were created and a prototype was employed to send and receive the messages, as well as to report the differences between standard uncompressed SMS, Huffman-encoded SMS and LD-based encoded SMS. Table 5.6 lists the messages used in testing.

<table>
<thead>
<tr>
<th>Message</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World!</td>
<td>0012</td>
</tr>
<tr>
<td>The quick brown fox jumps over the lazy dog</td>
<td>0043</td>
</tr>
<tr>
<td>Now is the time for all good men to come to the aid of the party</td>
<td>0064</td>
</tr>
<tr>
<td>Thanks! Hope you have great Xmas too! What you up to these days? Still in London? Can't believe I've been back in NZ for nearly 18 months, but I'm still loving every minute of it!</td>
<td>0179</td>
</tr>
<tr>
<td>Yes, but to be honest, when I am working on a problem I never think about beauty. I only think about how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong</td>
<td>0205</td>
</tr>
<tr>
<td>Hey mate, how are you? Meant to email you after a trip to cape town to let you know i'd finally seen your beautiful city. What are you doing in london when you can call cape town home? Was a nice change from life in the slums too. Hopefully coming over for a wedding in April next year - will keep you posted. And would love you to come visit Australia!</td>
<td>0353</td>
</tr>
<tr>
<td>And so it was indeed: she was now only ten inches high, and her face brightened up at the thought that she was now the right size for going through the little door into that lovely garden. First, however, she waited for a few minutes to see if she was going to shrink any further: she felt a little nervous about this; 'for it might end, you know,' said Alice to herself, 'in my going out altogether, like a candle. I wonder what I should be like then?' And she tried to fancy what the flame of a candle is like after the candle is blown out, for she could not remember ever having seen such a thing.</td>
<td>0600</td>
</tr>
<tr>
<td>We understand it still that there is no easy road to freedom. We know it well that none of us acting alone can achieve success. We must therefore act together as a united people, for national reconciliation, for nation building, for the birth of a new world. Let there be justice for all. Let there be peace for all. Let there be work, bread, water and salt for all. Let each know that for each the body, the mind and the soul have been freed to fulfill themselves. Never, never and never again shall it be that this beautiful land will again experience the oppression of one by another and suffer the indignity of being the skunk of the world. Let freedom reign. The sun shall never set on so glorious a human achievement!</td>
<td>0723</td>
</tr>
</tbody>
</table>

*Table 5.6: SMS messages used in testing.*
The results for LD-based compression of the messages listed in Table 5.6 were evaluated with the aid of the prototype; the results in each case were compared to that obtainable through Huffman encoding.

<table>
<thead>
<tr>
<th>Ref# 0012</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
</tr>
<tr>
<td>Decompressed message</td>
</tr>
<tr>
<td>Characters</td>
</tr>
<tr>
<td>Hit ratio</td>
</tr>
<tr>
<td>Words not found in dictionary</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>84</td>
<td>160</td>
<td>40</td>
</tr>
<tr>
<td>Bits per input char</td>
<td>7.00</td>
<td>13.33</td>
<td>3.33</td>
</tr>
<tr>
<td>Transmission bits</td>
<td>208</td>
<td>280</td>
<td>168</td>
</tr>
<tr>
<td>Transmission bits per input char</td>
<td>17.33</td>
<td>23.33</td>
<td>14.00</td>
</tr>
<tr>
<td>Number of SMS messages</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Characters per SMS</td>
<td>12.00</td>
<td>12.00</td>
<td>12.00</td>
</tr>
</tbody>
</table>

Table 5.7: Results of encoding and sending SMS message Ref# 0012.

Table 5.7 shows a summary of results from running the test for Ref#0012. The full, line-by-line, log is attached to Appendix A. The GSM terminal was mounted on /dev/ttyUSB0 and the message "Hello World!" sent to 07894555501. Next, the AT driver's log shows the commands issued to send SMS-SUBMIT PDU containing the LD-based payload. The same modem was in turn used to receive the SMS message, which means that the actual phone number for the SIM in the GSM terminal is the same as the number to which the message was sent. This was done to eliminate the expense of buying a second terminal. To allow the message time to be delivered from the GSM network, a 60-second pause was included before issuing the AT commands for checking for unread messages. In fact, the command issued did not actually ask for unread messages, as there is no such command. Instead, all messages were retrieved, and then each was inspected by checking for a bit flag that indicated whether the message had been read or not. To avoid repeatedly downloading messages that had already been read, the researchers introduced a little hack whereby once the most recent unread messages were obtained, the script issued AT commands to delete all messages on the terminal. This
meant that the researchers started with a so-called “clean slate” so that, following the next sent message, only the latest messages were retrieved. Note that these peculiarities do not appear to affect the outcome of this analysis and were added to avoid capturing log output for messages in which we were not interested for the sake of the experiments.

The unread message was then decoded, and the LD-based payload was read from the SMS-DELIVER PDU. The payload was decompressed, and the message was displayed. A quick inspection shows that although the original message was sent as "Hello World!", the message that was actually sent and received was the lower-case equivalent "hello world". Other than the change in case, the message was exactly the same. The summary at the end of the output shows that all of the words in the message were found in the dictionary (i.e. 100% hit ratio); the summary also lists the index rank at which each of the words were found in the dictionary as well as the compressed hex and binary stream for each word. Notice that the hex values for the words are exactly the same as those that appear in the payload issued over the serial line, as printed out by the AT driver. The output size of the string is 84, 160 and 40 bits for the uncompressed, Huffman compressed and LD-based compressed algorithms respectively.

Consistent with our earlier observation, Huffman actually increases the size of a very short string. The increase in size is quite severe, as the Huffman output is almost twice the size of the original message. Alternatively, the LD-based output is almost half the size of the uncompressed message. It is significant to observe that the output produced by the LD-based algorithm is four times as effective as Huffman encoding, which is the official GSM method for compressing SMS messages. The summary shows entries for both "Bits" and "Transmission Bits". The "Bits" entry, which we have already discussed, indicates the size of the compression output. Remember, however, that the payload must be packed into a PDU-SUBMIT PDU. This PDU has a strict structure and requires setting additional header and indicator flags, and it represents the actual size of the SMS message in terms of the number of bits that would be used to carry the SMS message over the GSM network. The difference between the "Bits" and "Transmission Bits" values for each of the three cases is about 10 bits. This difference of 10 bits is constant and there is nothing we can do to optimise it since it forms part of the fixed structure of the PDU. However, since it is a fixed amount that is added to each of the "Bits" values, it should be clear that the case with the highest "Bits" value is also the case with the highest "Transmission Bits" value.
The results for Ref#0043 (shown in Table 5.8) are similar to those of the previous message and the hit ratio is again 100%. The message "The quick brown fox jumps over the lazy dog" is an interesting one in that it uses every letter in the alphabet. Using the entire alphabet in such a short message means that there is little repetition for Huffman encoding to exploit. Remember that Huffman encoding attempts to find those patterns that occur most often in an input stream and then replaces each of these patterns with the shortest possible keyword. The most frequently occurring patterns are assigned the shortest codewords, and the least frequently occurring ones are assigned the longest codewords. Once again, Huffman encoding increased the size of the message, while LD-based encoding decreased it by more than half. However, since the message in all cases can be transmitted as a single SMS message, there are no cost savings in this case. The actual size of an SMS message matters little as long as that size is less than the maximum length of a single SMS message. In all these cases, all the characters were accommodated in a single SMS message, meaning a consistent 43 characters.

| Ref# 0043 | The quick brown fox jumps over the lazy dog | the quick brown fox jumps over the lazy dog | 43 |
| Hit ratio | 100% | 100% | |
| Words not found in dictionary | NA - All words found in dictionary | NA - All words found in dictionary | |

| Result | Uncompressed | Huffman | LD-based |
| Bits | 301 | 400 | 128 |
| Bits per input char | 7.00 | 9.30 | 2.98 |
| Transmission bits | 424 | 520 | 256 |
| Transmission bits per input char | 9.86 | 12.09 | 5.95 |
| Number of SMS messages | 1 | 1 | 1 |
| Characters per SMS | 43.00 | 43.00 | 43.00 |

Table 5.8: Results of encoding and sending SMS message Ref# 0043
Ref# 0064

<table>
<thead>
<tr>
<th>Result</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>448</td>
<td>504</td>
<td>176</td>
</tr>
<tr>
<td>Bits per input char</td>
<td>7.00</td>
<td>7.88</td>
<td>2.75</td>
</tr>
<tr>
<td>Transmission bits</td>
<td>568</td>
<td>624</td>
<td>304</td>
</tr>
<tr>
<td>Transmission bits per input char</td>
<td>8.88</td>
<td>9.75</td>
<td>4.75</td>
</tr>
<tr>
<td>Number of SMS messages</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Characters per SMS</td>
<td>64.00</td>
<td>64.00</td>
<td>64.00</td>
</tr>
</tbody>
</table>

Table 5.9: Results of encoding and sending SMS message Ref# 0064

Looking at the results from Table 5.9, a pattern starts to emerge; for each of the previous three messages, the Huffman algorithm increased the number of bits when compared to the uncompressed message, while the LD-based algorithm reduced the size. The reduction this time is quite drastic; the LD-based output is 60% smaller than that of the uncompressed message. We expect Huffman encoding to clearly demonstrate its advantages as messages become longer, surpassing the LD-based algorithm.
Ref# 0179

Message: Thanks! Hope you have great Xmas too! What you up to these days? Still in London? Can't believe I've been back in NZ for nearly 18 months, but I'm still loving every minute of it!

Decompressed message: thanks! hope you have great xmas too! what you up to these days? still in london? can't believe i've been back in nz for nearly 18 months, but i'm still loving every minute of it!

Characters: 179

Hit ratio: 91%

Words not found in dictionary: 18, nz, xmas

<table>
<thead>
<tr>
<th>Result</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>1253</td>
<td>1184</td>
<td>592</td>
</tr>
<tr>
<td>Bits per input char</td>
<td>7.00</td>
<td>6.61</td>
<td>3.31</td>
</tr>
<tr>
<td>Transmission bits</td>
<td>1592</td>
<td>1520</td>
<td>720</td>
</tr>
<tr>
<td>Transmission bits per input char</td>
<td>8.89</td>
<td>8.49</td>
<td>4.02</td>
</tr>
<tr>
<td>Number of SMS messages</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Characters per SMS</td>
<td>89.50</td>
<td>89.50</td>
<td>179.00</td>
</tr>
</tbody>
</table>

Table 5.10: Results of encoding and sending SMS message Ref# 0179

Ref#0179 (results in Table 5.10) is the first string tested that is longer than the 160-character limit. In addition, in this example not all of the words can be found in the dictionary; the hit ratio stood at 91% with words like "18", "nz" or "xmas" left unresolved. For each of these words, the LD-based algorithm relied on per-character encoding. Notice how, for instance, "xmas" was encoded as "x" + "m" + "a" + "s". Despite not finding all of the words in the dictionary, the LD-based compression reduced the total number of SMS messages necessary to send the text from two to one. Note that, as the strings become a little bit longer, Huffman encoding shows some benefit; thus, for the first time, Huffman compression shrinks the size of the message, albeit by a modest amount insufficient to shorten it into a single SMS message.
Ref# 0205

Message
Yes, but to be honest, when I am working on a problem I never think about beauty. I only think about how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong.

Decompressed message
yes, but to be honest, when i am working on a problem i never think about beauty. i only think about how to solve the problem. but when i have finished, if the solution is not beautiful, i know it is wrong.

Characters
205

Hit ratio
100%

Words not found in dictionary
NA - All words found in dictionary

<table>
<thead>
<tr>
<th>Result</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>1435</td>
<td>1112</td>
<td>608</td>
</tr>
<tr>
<td>Bits per input char</td>
<td>7.00</td>
<td>5.42</td>
<td>2.97</td>
</tr>
<tr>
<td>Transmission bits</td>
<td>1776</td>
<td>1232</td>
<td>736</td>
</tr>
<tr>
<td>Transmission bits per input char</td>
<td>8.66</td>
<td>6.01</td>
<td>3.59</td>
</tr>
<tr>
<td>Number of SMS messages</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Characters per SMS</td>
<td>102.50</td>
<td>205.00</td>
<td>205.00</td>
</tr>
</tbody>
</table>

Table 5.11: Results of encoding and sending SMS message Ref# 0205

The experiments were conducted in order of increasing message size. Ref#0205 (results shown in Table 5.11) and those that follow are all more than the 160-character limit for a single SMS message. The trends previously observed continue in that Huffman encoding produces more of a saving, but this saving is still not nearly as much as that resulting from LD-based compression. This time, however, Huffman encoding did reduce the number of SMS text messages to one. This finding emphasises just how much Huffman compression is biased against short strings. In the previous experiment, the character count was smaller than in this case, yet two separate SMS messages were required. Now that the character count has increased, Huffman compression has more text to optimise and thus can reduce the output to a single SMS message. It is not the number of bits that matter directly but rather the number of SMS messages, and so in this case the outcome favours Huffman and LD-based compression equally. Both Huffman and LD-based compression fit 205 characters into a single SMS message, thereby coming under the 160-character limit.
Hey mate, how are you? Meant to email you after a trip to Cape Town to let you know I'd finally seen your beautiful city. What are you doing in London when you can call Cape Town home? Was a nice change from life in the slums too. Hopefully coming over for a wedding in April next year - will keep you posted. And would love you to come visit Australia!

Decompressed message

Hey mate, how are you? Meant to email you after a trip to Cape Town to let you know I'd finally seen your beautiful city. What are you doing in London when you can call Cape Town home? Was a nice change from life in the slums too. Hopefully coming over for a wedding in April next year - will keep you posted. And would love you to come visit Australia!

Characters 353
Hit ratio 98%
Words not found in dictionary slums

<table>
<thead>
<tr>
<th>Result</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>2471</td>
<td>1856</td>
<td>1080</td>
</tr>
<tr>
<td>Bits per input char</td>
<td>7.00</td>
<td>5.26</td>
<td>3.06</td>
</tr>
<tr>
<td>Transmission bits</td>
<td>2984</td>
<td>2192</td>
<td>1208</td>
</tr>
<tr>
<td>Transmission bits per input char</td>
<td>8.45</td>
<td>6.21</td>
<td>3.42</td>
</tr>
<tr>
<td>Number of SMS messages</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Characters per SMS</td>
<td>117.67</td>
<td>176.50</td>
<td>353.00</td>
</tr>
</tbody>
</table>

Table 5.12: Results of encoding and sending SMS message Ref# 0353

Table 5.12 shows the results for test run Ref#0353. In this test, the hit ratio is 98%, because "slums" does not exist in the dictionary. The uncompressed message would be sent in three parts, while the Huffman-compressed version would be sent in two parts. Finally, the LD-based message would still only require a single SMS message to accommodate all 353 characters of the message.
Ref# 0600

Message

And so it was indeed: she was now only ten inches high, and her face brightened up at the thought that she was now the right size for going through the little door into that lovely garden. First, however, she waited for a few minutes to see if she was going to shrink any further: she felt a little nervous about this; 'for it might end, you know,' said Alice to herself, 'in my going out altogether, like a candle. I wonder what I should be like then?' And she tried to fancy what the flame of a candle is like after the candle is blown out, for she could not remember ever having seen such a thing.

Decompressed message

and so it was indeed: she was now only ten inches high, and her face brightened up at the thought that she was now the right size for going through the little door into that lovely garden. first, however, she waited for a few minutes to see if she was going to shrink any further: she felt a little nervous about this; 'for it might end, you know,' said alice to herself, 'in my going out altogether, like a candle. i wonder what i should be like then?' and she tried to fancy what the flame of a candle is like after the candle is blown out, for she could not remember ever having seen such a thing.

Characters 600
Hit ratio 99%
Words not found in dictionary brightened

<table>
<thead>
<tr>
<th>Result</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>4200</td>
<td>2768</td>
<td>1768</td>
</tr>
<tr>
<td>Bits per input char</td>
<td>7.00</td>
<td>4.61</td>
<td>2.95</td>
</tr>
<tr>
<td>Transmission bits</td>
<td>4880</td>
<td>3272</td>
<td>2112</td>
</tr>
<tr>
<td>Transmission bits per input char</td>
<td>8.13</td>
<td>5.45</td>
<td>3.52</td>
</tr>
<tr>
<td>Number of SMS messages</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Characters per SMS</td>
<td>150.00</td>
<td>200.00</td>
<td>300.00</td>
</tr>
</tbody>
</table>

Table 5.13: Results of encoding and sending SMS message Ref# 0600

The excerpted text for Ref#0600 (results shown in Table 5.13) was taken from Alice in Wonderland. Only "brightened" was not found in the dictionary, bringing the hit ratio to 99%. In all experiments thus far, the dictionary provided adequate coverage, as was the case for the novels, making it possible to achieve a satisfactory compression ratio. The uncompressed, Huffman-compressed and LD-based messages required four, three and two parts respectively. Note that, during transmission, the message was sent as two separate SMS-SUBMIT PDUs. In addition, on delivery there were two unread SMS-DELIVERY PDUs. Significantly, in this case, we have proven that we are able to use LD-based
encoding to send and reassemble multipart SMS messages, just as in standard concatenated SMS. However, instead of the handset reassembling the message, we had to reassemble it ourselves.

### Ref# 0723

<table>
<thead>
<tr>
<th>Message</th>
<th>We understand it still that there is no easy road to freedom. We know it well that none of us acting alone can achieve success. We must therefore act together as a united people, for national reconciliation, for nation building, for the birth of a new world. Let there be justice for all. Let there be peace for all. Let there be work, bread, water and salt for all. Let each know that for each the body, the mind and the soul have been freed to fulfill themselves. Never, never and never again shall it be that this beautiful land will again experience the oppression of one by another and suffer the indignity of being the skunk of the world. Let freedom reign. The sun shall never set on so glorious a human achievement!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decompressed message</td>
<td>we understand it still that there is no easy road to freedom. we know it well that none of us acting alone can achieve success. we must therefore act together as a united people, for national reconciliation, for nation building, for the birth of a new world. let there be justice for all. let there be peace for all. let there be work, bread, water and salt for all. let each know that for each the body, the mind and the soul have been freed to fulfill themselves. never, never and never again shall it be that this beautiful land will again experience the oppression of one by another and suffer the indignity of being the skunk of the world. let freedom reign. the sun shall never set on so glorious a human achievement!</td>
</tr>
<tr>
<td>Characters</td>
<td>723</td>
</tr>
<tr>
<td>Hit ratio</td>
<td>99%</td>
</tr>
<tr>
<td>Words not found in dictionary</td>
<td>indignity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>5061</td>
<td>3016</td>
<td>2064</td>
</tr>
<tr>
<td>Bits per input char</td>
<td>7.00</td>
<td>4.17</td>
<td>2.85</td>
</tr>
<tr>
<td>Transmission bits</td>
<td>5912</td>
<td>3520</td>
<td>2408</td>
</tr>
<tr>
<td>Transmission bits per input char</td>
<td>8.18</td>
<td>4.87</td>
<td>3.33</td>
</tr>
<tr>
<td>Number of SMS messages</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Characters per SMS</td>
<td>144.60</td>
<td>241.00</td>
<td>361.50</td>
</tr>
</tbody>
</table>

*Table 5.14: Results of encoding and sending SMS message Ref# 0723*

The last message, Ref#0723, is fairly long at 723 characters; messages longer than this would probably be quite scarce. The uncompressed message would require five parts, while the Huffman
algorithm would encode the message into three parts. Finally, the LD-based message would only require two parts, fitting roughly 361 characters into a single SMS message. Next, we take a look at our results across the experiments.

<table>
<thead>
<tr>
<th>Length</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>0012</td>
<td>7.00</td>
<td>13.33</td>
<td>3.33</td>
</tr>
<tr>
<td>0043</td>
<td>7.00</td>
<td>9.30</td>
<td>2.98</td>
</tr>
<tr>
<td>0064</td>
<td>7.00</td>
<td>7.88</td>
<td>2.75</td>
</tr>
<tr>
<td>0179</td>
<td>7.00</td>
<td>6.61</td>
<td>3.31</td>
</tr>
<tr>
<td>0205</td>
<td>7.00</td>
<td>5.42</td>
<td>2.97</td>
</tr>
<tr>
<td>0353</td>
<td>7.00</td>
<td>5.26</td>
<td>3.06</td>
</tr>
<tr>
<td>0600</td>
<td>7.00</td>
<td>4.61</td>
<td>2.95</td>
</tr>
<tr>
<td>0723</td>
<td>7.00</td>
<td>4.17</td>
<td>2.85</td>
</tr>
</tbody>
</table>

**Figure 5.1:** The average number of bits used to encode each of the characters in the compressed message. The lower the number, the better the compression.

Figure 5.1 shows that the uncompressed message always uses the standard 7 bits per character as encoded per ETSI GSM 03.38 and thus acts only as a baseline. When compressing messages with Huffman encoding, the first four messages, which are the shortest ones, actually result in an increase
in the number of bits required to capture a character. We see that the Huffman algorithm performs consistently better as the message size increases, and thus we expect this trend to continue as message length becomes longer, beyond that considered in these experiments. This was indeed observed when encoding novels. However, the range of message sizes evaluated is considered representative of the length of messages users would typically type when sending an SMS message. The performance of the LD-based method does not show any improvement for a longer message size, nor does it show a penalty for shorter messages. Huffman encoding is heavily dependent on the size of the messages, as it depends on the opportunity to exploit repeating patterns. In contrast, LD-based encoding shows no such bias – the more words found in the dictionary, the better the result. The encoding shows results consistent in the range of 2.85 to 3.33 bits per character.
<table>
<thead>
<tr>
<th>Length</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>0012</td>
<td>17.33</td>
<td>23.33</td>
<td>14.00</td>
</tr>
<tr>
<td>0043</td>
<td>9.86</td>
<td>12.09</td>
<td>5.95</td>
</tr>
<tr>
<td>0064</td>
<td>8.88</td>
<td>9.75</td>
<td>4.75</td>
</tr>
<tr>
<td>0179</td>
<td>8.89</td>
<td>8.49</td>
<td>4.02</td>
</tr>
<tr>
<td>0205</td>
<td>8.66</td>
<td>6.01</td>
<td>3.59</td>
</tr>
<tr>
<td>0353</td>
<td>8.45</td>
<td>6.21</td>
<td>3.42</td>
</tr>
<tr>
<td>0600</td>
<td>8.13</td>
<td>5.45</td>
<td>3.52</td>
</tr>
<tr>
<td>0723</td>
<td>8.18</td>
<td>4.87</td>
<td>3.33</td>
</tr>
</tbody>
</table>

*Figure 5.2: The average number of bits per character used in the PDU. Once the message is encoded with the compression method, it is then passed through a second stage of encoding to package the payload into the PDU transmission format.*

Whereas Figure 5.1 shows the bits per character as a result of compressing the message, Figure 5.2 shows the bits per character necessary to transmit the message. The figures are larger as a result of the addition of meta-information required to structure the data into PDU format. It is worth considering this result in that it clearly shows the so-called “effective” bits per character or, put another way, the total bits required to transmit the message against the size of the content. The overhead is consistent in that the same number of bits are added under the uncompressed, Huffman-compressed and LD-based compressed methods within a single experiment. In the first instance, regarding message 0012, the penalty is about 10 bits. The bit count per character for an uncompressed message increases from 7 to 17.33, whereas the count for the Huffman method increases from 13.33 to 23.33. The LD-based count increases from 3.33 to 14.00. The trend continues, but becomes less severe as the message size increases.
To use as few parts as possible per message, it is important to fit as many characters as possible in each SMS message. The hard limit is 160 characters for a single SMS message, or 153 characters for every part of a concatenated SMS message (i.e. 1120 bits minus a 6-byte header leaves 1072 bits that can fit 153 7-bit characters). The values in Figure 5.3 are the average taken across all of the parts making up the whole, but, of course, none of the uncompressed messages exceeded the 153-character limit.

Figure 5.3: The number of characters fitted into each SMS part. A standard SMS can accommodate 160 characters and a concatenated SMS can take max 153 characters per part. Using LD-based compression, the maximum was extended to 361 characters.
limit. With the aid of Huffman encoding, a maximum count of 241 characters was achieved; this figure was 361 with LD-based encoding.

<table>
<thead>
<tr>
<th>Length</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>0012</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0043</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0064</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0179</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0205</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0353</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0600</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>0723</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 5.4: The number of SMS messages (or concatenated SMS parts) to transmit each of the test messages.

Moving from compressed bits per character to transmission bits per character to characters per SMS message, we finally consider the overall aim of this study, that is, to reduce the number of SMS messages for communicating a given message (shown in Figure 5.4). Neither Huffman nor LD-based encoding left the message count worse off in any of the instances under consideration. There was
either no benefit for some of the cases or a reduction. However, LD-based encoding yielded a better result in all instances. As the formal compression scheme adopted in GSM specification, Huffman encoding did produce savings for longer messages, while LD-based compression yielded even better results.
Chapter 6 - Conclusion

This paper explored the design, implementation and evaluation of a dictionary-based method for compressing English SMS text messages. The dictionary was constructed using frequency lists of those words that appear most often in film scripts and was then tested against some popular novels in English literature. The design is based on lower-case words in order to avoid reserving space for different case permutations (i.e. upper case, mixed case, title case and so on). This decision meant that more unique words could be added to the dictionary, resulting in a greater hit ratio for the input stream and, therefore, greater compression efficiency. The decompressed message thus always appears in lower case, meaning that this decision also leads to a trade-off between efficiency and quality. Therefore, this decompression method becomes a lossy one. The loss of quality in this case only involves the case of the message, and since SMS is largely an informal method of communication, there are situations in which people might be willing to make this trade-off in order to save on their bills. Furthermore, it would be a waste to use 7 bits to encode a space character. Since most words are followed by a space, this is a significant waste of space in that the number of spaces will usually be equal to 1 less than the number of words in the message. The dictionary therefore allows using a single bit to indicate whether or not a word is followed by a space. LD-based dictionary stores more than 16,000 of the most commonly used English words. Nation [38] showed that between 8,000 and 9,000 words are needed for reading and writing and 6,000 to 7,000 words for speaking and listening in order to understand 98% of typical English language. The dictionary setup for instance contains only 45.75% of the unique words in Origin of the Species, but can still compress 86.23% of the content. This is because, even though highly specialised in content and vocabulary, the specialised words still do not occur as frequently as the more common words from the dictionary. Table 6.1 gives an overview of how LD-based compression rates against methods discussed in the “Related Work” section.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compression ratio in bits per character</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETSI GSM 03.38</td>
<td>7.00</td>
</tr>
<tr>
<td>Huffman</td>
<td>6.01</td>
</tr>
<tr>
<td>CleverTexting</td>
<td>5.00</td>
</tr>
<tr>
<td>ETSI GSM 03.42</td>
<td>4.67</td>
</tr>
<tr>
<td>Nakayama</td>
<td>4.05</td>
</tr>
<tr>
<td>Rein, Guehmann and Fitzek</td>
<td>3.29</td>
</tr>
<tr>
<td><strong>LD-based</strong></td>
<td><strong>2.98</strong></td>
</tr>
</tbody>
</table>

Table 6.1: Comparing the LD-based method against related work.
The GSM specification officially supports Huffman encoding, but we have shown that Huffman encoding is not particularly well-suited to short messages, since the overhead of encoding the dictionary into the data stream adds to the message size, often increasing the size of messages rather than reducing them. The results indicate that the method outperforms Huffman encoding for small messages in the size range expected for SMS texts and that it might offer a better alternative. The major drawback of the proposed compression method is that the dictionary would need to be installed on both the sending and the receiving handsets, thereby using some of the already limited memory on the phone. The dictionary measures 316 Kb in size, which should easily fit on modern handsets, many of which can store MP3 albums and films.
Chapter 7 - Future work

The current version of the compression scheme presented here has been shown to achieve the goal of decreasing the size of text messages, even outperforming the standard GSM method of compression. Further fine-tuning could be performed to improve the results even more. If a word were misspelt, for example, it would not be found in the dictionary and thus would need to be encoded one character at a time. It might make sense to use the dictionary as a spell-checker prior to encoding the message.

The current form of the dictionary stores either characters or words, but further savings could be attained by also storing commonly used phrases. There is no support for encoding hyphenated words, and so something like “devil-may-care” would not yield a hit from the dictionary. The algorithm could be improved by splitting up such words at the point of the hyphen and storing the result as “[ devil ][ - ][ may ][ - ][ care ]”. This improvement could be made even more generic in order to scan any words not found in the dictionary to see whether they could be split into parts that do exist in the dictionary. Something like “easement” could then be encoded as “[ ease ][ m ][ e ][ n ][ t ]”.

Further revisions to the dictionary would require the inclusion of a scheme by which to indicate the appropriate version of the dictionary to use when decompressing the message. This could be achieved by including dictionary version information in the compression header. Dictionaries could also be created for other languages, as there is nothing inherent in the algorithm that prevents support for other languages. That is, language-specific dictionaries could be created and then indicated in the compression header.
Chapter 8 - References


7. ETSI GSM 03.40, Digital cellular telecommunications system (Phase 2+); Technical realization of the short message service point-to-point, 3GPP Technical Specification, version 7.5.0, 2001.


9. ETSI GSM 03.38, Digital cellular telecommunications system (Phase 2+); Alphabets and language-specific information, 3GPP Technical Specification, v. 5.3.0, 1996.


17. DA Huffman, “A method for the construction of minimum-redundancy codes”,


28. ETSI GSM 03.42, Digital cellular telecommunications system (Phase 2+); Compression algorithm for text messaging services, 3GPP Technical Specification, v. 7.1.1, 1998.


Appendix A - Printout Ref# 0012

Details
=================================================================
GSM Modem SerialPort: /dev/ttyUSB0
PhoneNumber: 07894555501
Message: Hello World!

Starting...

Stable Library
=================================================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Tx: ATE0
Rx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGS=20
Rx: >
Tx: 0011000B817098545505F100F4FF0600814BC14702#
Rx: +CMGS: 56
Rx: OK

Sending done!

Waiting 60 seconds before trying to read message. This gives it time to be delivered from the
GSM network...
OK, here we go.

Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CPMS?
Rx: +CPMS: "ME",1,25,"SM",0,50,"SM",0,50
Rx: OK
Tx: AT+CMGR=1
Rx: +CMGR: 0,,25
Rx: 0791448720003023040C9144874955551000F4900192810512000600814BC14702
Rx: OK

Found Unread Message: 0791448720003023040C9144874955551000F4900192810512000600814BC14702

Delete all messages in store as part of housekeeping

Tx: AT+CMGD=1
Rx: OK
Deleted message at index 1
Decompressed Message: hello world!

Receiving done!

Please wait, generating report...
Report generated.

SMS Compression Report
=================================================================

54
Text Message [12 characters]: Hello World!

Words Not Found in Dictionary [100.00 percent hit ratio]

[Dictionary Symbol] [Dictionary Index] [Encoded Hex Byte(s)] [Encoded Binary Byte(s)]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Index</th>
<th>Encoded Hex</th>
<th>Encoded Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td>331</td>
<td>814B</td>
<td>10000001 01001011</td>
</tr>
<tr>
<td>world</td>
<td>16711</td>
<td>C147</td>
<td>11000001 01000111</td>
</tr>
<tr>
<td>!</td>
<td>2</td>
<td>02</td>
<td>00000010</td>
</tr>
</tbody>
</table>

Experimental: JNI_OnLoad called.
Appendix B - Printout Ref# 0043

Details
========================================
GSM Modem SerialPort: /dev/ttyUSB0
PhoneNumber: 07894555501
Message: The quick brown fox jumps over the lazy dog
Starting...

Stable Library
========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGS=31
Rx: +
Tx: 001100BB170995345505F100F4FF110048842E86DC8EBB9BE780B9489612C316#
Rx: +CMGS: 57
Rx: OK

Sending done!

Waiting 60 seconds before trying to read message. This gives it time to be delivered from the
GSM network...
OK, here we go.

Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CPMS?
Rx: +CPMS: "ME",1,25,"SM",0,50,"SM",0,50
Rx: OK
Tx: AT+CMGR=1
Rx: +CMGR: 0,,36
Rx: D91448720003023040C9144874955551000F490019281158300110048842E86DC8EBB9BE780B9489612C316
Rx: OK

Found Unread Message:
0791448720003023040C9144874955551000F490019281158300110048842E86DC8EBB9BE780B9489612C316
Rx: OK

Delete all messages in store as part of housekeeping

Tx: AT+CMGD=1
Rx: OK
Deleted message at index 1

Decompressed Message: the quick brown fox jumps over the lazy dog

Receiving done!

Please wait, generating report...
Report generated.

SMS Compression Report
=================================================================
Text Message [43 characters]: The quick brown fox jumps over the lazy dog

Words Not Found in Dictionary [100.00 percent hit ratio]

[N/A - All words exist in dictionary]

<table>
<thead>
<tr>
<th>Dictionary Symbol</th>
<th>Dictionary Index</th>
<th>Encoded Hex Byte(s)</th>
<th>Encoded Binary Byte(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[compression indicator]</td>
<td>0</td>
<td>00</td>
<td>00000000</td>
</tr>
<tr>
<td>the[sp]</td>
<td>72</td>
<td>48</td>
<td>01001000</td>
</tr>
<tr>
<td>quick[sp]</td>
<td>1070</td>
<td>842E</td>
<td>10000100 00101110</td>
</tr>
<tr>
<td>brown[sp]</td>
<td>1756</td>
<td>86DC</td>
<td>10000110 11011100</td>
</tr>
<tr>
<td>fox[sp]</td>
<td>3723</td>
<td>88BB</td>
<td>10001110 10001011</td>
</tr>
<tr>
<td>jumps[sp]</td>
<td>7143</td>
<td>9BE7</td>
<td>10011011 11100111</td>
</tr>
<tr>
<td>over[sp]</td>
<td>185</td>
<td>80B9</td>
<td>10000000 10111001</td>
</tr>
<tr>
<td>the[sp]</td>
<td>72</td>
<td>48</td>
<td>01001000</td>
</tr>
<tr>
<td>lazy[sp]</td>
<td>5650</td>
<td>9612</td>
<td>10010110 00010010</td>
</tr>
<tr>
<td>dog</td>
<td>17174</td>
<td>C316</td>
<td>11000011 00010110</td>
</tr>
</tbody>
</table>

Experimental: JNI_OnLoad called.
Appendix C - Printout Ref# 0064

Details
==================================================================
GSM Modem SerialPort: /dev/ttyUSB0
PhoneNumber: 07894555501
Message: Now is the time for all good men to come to the aid of the party

Starting...

Stable Library
==================================================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7

Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGS=37
Rx: >
Tx: 001100B817098545505F100F4FF17007B4F48808E54658087823E4780864748F7E4C48C1C2#
Rx: +CMGS: 58
Rx: OK

Sending done!

Waiting 60 seconds before trying to read message. This gives it time to be delivered from the GSM network...
OK, here we go.

Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CPMS?
Rx: +CPMS: "ME",1,25,"SM",0,50,"SM",0,50
Rx: OK
Tx: AT+CMGR=1
Rx: +CMGR: 0,,42
Rx:
0791448720003023040C9144874955551000F49001928125550017007B4F48808E54658087823E4780864748F7E4C48C1C2
Rx: OK

Found Unread Message:
0791448720003023040C9144874955551000F49001928125550017007B4F48808E54658087823E4780864748F7E4C48C1C2

Delete all messages in store as part of housekeeping

Tx: AT+CMGD=1
Rx: OK
Deleted message at index 1

Decompressed Message: now is the time for all good men to come to the aid of the party

Receiving done!

Please wait, generating report...
Report generated.
SMS Compression Report

REF# 0064

Text Message [64 characters]: Now is the time for all good men to come to the aid of the party

Words Not Found in Dictionary [100.00 percent hit ratio]

[N/A - All words exist in dictionary]

<table>
<thead>
<tr>
<th>Dictionary Symbol</th>
<th>Dictionary Index</th>
<th>Encoded Hex Byte(s)</th>
<th>Encoded Binary Byte(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[compression indicator]</td>
<td>0</td>
<td>00</td>
<td>00000000</td>
</tr>
<tr>
<td>now[sp]</td>
<td>123</td>
<td>7B</td>
<td>01111011</td>
</tr>
<tr>
<td>is[sp]</td>
<td>79</td>
<td>4F</td>
<td>01001111</td>
</tr>
<tr>
<td>the[sp]</td>
<td>72</td>
<td>48</td>
<td>01001000</td>
</tr>
<tr>
<td>time[sp]</td>
<td>142</td>
<td>808E</td>
<td>10000000 10001110</td>
</tr>
<tr>
<td>for[sp]</td>
<td>84</td>
<td>54</td>
<td>01010100</td>
</tr>
<tr>
<td>all[sp]</td>
<td>101</td>
<td>65</td>
<td>01100101</td>
</tr>
<tr>
<td>good[sp]</td>
<td>135</td>
<td>8087</td>
<td>10000000 10000011</td>
</tr>
<tr>
<td>men[sp]</td>
<td>574</td>
<td>823E</td>
<td>10000010 00111110</td>
</tr>
<tr>
<td>to[sp]</td>
<td>71</td>
<td>47</td>
<td>01000111</td>
</tr>
<tr>
<td>come[sp]</td>
<td>134</td>
<td>8086</td>
<td>10000000 10000011</td>
</tr>
<tr>
<td>to[sp]</td>
<td>71</td>
<td>47</td>
<td>01000111</td>
</tr>
<tr>
<td>the[sp]</td>
<td>72</td>
<td>48</td>
<td>01001000</td>
</tr>
<tr>
<td>aid[sp]</td>
<td>3966</td>
<td>8F7E</td>
<td>10001111 01111110</td>
</tr>
<tr>
<td>of[sp]</td>
<td>76</td>
<td>4C</td>
<td>01010100</td>
</tr>
<tr>
<td>the[sp]</td>
<td>72</td>
<td>48</td>
<td>01001000</td>
</tr>
<tr>
<td>party</td>
<td>16834</td>
<td>C1C2</td>
<td>11000001 11000010</td>
</tr>
</tbody>
</table>

REF# 0064 [64 chars] Uncompressed Huffman LD-Based

| Bits: | 448 | 504 | 176 |
| Bits Per Input Char: | 7.00 | 7.88 | 2.75 |
| Transmission Bits: | 568 | 624 | 304 |
| Transmission Bits Per Input Char: | 8.86 | 9.75 | 4.75 |
| Number of SMS Messages: | 1 | 1 | 1 |
| Characters per SMS: | 64.00 | 64.00 | 64.00 |

Experimental: JNI_OnLoad called.
Appendix D - Printout Ref# 0179

Details
========================================
GSM Modem SerialPort: /dev/ttyUSB0
PhoneNumber: 07894555501
Message: Thanks! Hope you have great Xmas too! What you up to these days? Still in London? Can’t believe I’ve been back in NZ for nearly 18 months, but I’m still loving every minute of it!

Starting...

Stable Library
========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGF=89
Rx: >
Tx: 0011000B17098545505F100F4FF4B00C0EB02018155465680E73F34283AD01C08E02014E46774780F9C1CF200180D5C09A42D01809280EF80B5809E809150354101548855121901C7F0D01645380DC85C2813581694CC02702
Rx: +CMGS: 59
Rx: OK
Sending done!

Waiting 60 seconds before trying to read message. This gives it time to be delivered from the GSM network...

OK, here we go.

Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CPMS?
Rx: +CPMS: "ME",1,25,"SM",0,50,"SM",0,50
Rx: OK
Tx: AT+CMGR=1
Rx: +CMGR: 0,,94
Rx: 0791448720003023040C9144874955551000F4900192814531004B00C0EB02018155465680E73F34283AD01C08E02014E46774780F9C1CF200180DC50C9A42D01809280EF80B5809E809150354101548855121901C7F0D01645380DC85C2813581694CC02702
Rx: OK

Found Unread Message:

OK, here we go.

Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CPMS?
Rx: +CPMS: "ME",1,25,"SM",0,50,"SM",0,50
Rx: OK
Tx: AT+CMGR=1
Rx: +CMGR: 0,,94
Rx: 0791448720003023040C9144874955551000F4900192814531004B00C0EB02018155465680E73F34283AD01C08E02014E46774780F9C1CF200180DC50C9A42D01809280EF80B5809E809150354101548855121901C7F0D01645380DC85C2813581694CC02702
Rx: OK

Found Unread Message:

0791448720003023040C9144874955551000F4900192814531004B00C0EB02018155465680E73F34283AD01C08E02014E46774780F9C1CF200180DC50C9A42D01809280EF80B5809E809150354101548855121901C7F0D01645380DC85C2813581694CC02702
Rx: OK

Found Unread Message:

0791448720003023040C9144874955551000F4900192814531004B00C0EB02018155465680E73F34283AD01C08E02014E46774780F9C1CF200180DC50C9A42D01809280EF80B5809E809150354101548855121901C7F0D01645380DC85C2813581694CC02702
Rx: OK

Found Unread Message:

Delete all messages in store as part of housekeeping
Tx: AT+CMGD=1
Rx: OK
Deleted message at index 1
Decompressed Message: thanks! hope you have great xmas too! what you up to these days? still
in london? can't believe i've been back in nz for nearly 18 months, but i'm still loving every minute of it!

Receiving done!

Please wait, generating report...

Report generated.

SMS Compression Report

Text Message [179 characters]: Thanks! Hope you have great Xmas too! What you up to these days? Still in London? Can't believe I've been back in NZ for nearly 18 months, but I'm still loving every minute of it!

Words Not Found in Dictionary [91.00 percent hit ratio]

<table>
<thead>
<tr>
<th>Dictionary Symbol</th>
<th>Dictionary Index</th>
<th>Encoded Hex Byte(s)</th>
<th>Encoded Binary Byte(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>thanks</td>
<td>16619</td>
<td>C0EB</td>
<td>10000000 11101011</td>
</tr>
<tr>
<td>!</td>
<td>2</td>
<td>02</td>
<td>00000010</td>
</tr>
<tr>
<td>[sp]</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>hope[sp]</td>
<td>341</td>
<td>8155</td>
<td>10000001 01010101</td>
</tr>
<tr>
<td>you[sp]</td>
<td>70</td>
<td>46</td>
<td>01000110</td>
</tr>
<tr>
<td>have[sp]</td>
<td>86</td>
<td>56</td>
<td>01010110</td>
</tr>
<tr>
<td>great[sp]</td>
<td>231</td>
<td>80EB</td>
<td>10000000 11100111</td>
</tr>
<tr>
<td>x</td>
<td>63</td>
<td>3F</td>
<td>01111111</td>
</tr>
<tr>
<td>m</td>
<td>52</td>
<td>34</td>
<td>01101000</td>
</tr>
<tr>
<td>s</td>
<td>58</td>
<td>3A</td>
<td>01110101</td>
</tr>
<tr>
<td>[sp]</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>too</td>
<td>16526</td>
<td>C0EB</td>
<td>11000000 11001110</td>
</tr>
<tr>
<td>!</td>
<td>2</td>
<td>02</td>
<td>00000010</td>
</tr>
<tr>
<td>[sp]</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>what[sp]</td>
<td>78</td>
<td>4E</td>
<td>01001110</td>
</tr>
<tr>
<td>you[sp]</td>
<td>70</td>
<td>46</td>
<td>01000110</td>
</tr>
<tr>
<td>up[sp]</td>
<td>19</td>
<td>77</td>
<td>01110111</td>
</tr>
<tr>
<td>to[sp]</td>
<td>71</td>
<td>47</td>
<td>01000111</td>
</tr>
<tr>
<td>these[sp]</td>
<td>249</td>
<td>80F9</td>
<td>10000000 11111001</td>
</tr>
<tr>
<td>days</td>
<td>16847</td>
<td>C1CF</td>
<td>11000001 11001111</td>
</tr>
<tr>
<td>?</td>
<td>32</td>
<td>20</td>
<td>01010000</td>
</tr>
<tr>
<td>[sp]</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>still[sp]</td>
<td>220</td>
<td>80DC</td>
<td>10000000 11011100</td>
</tr>
<tr>
<td>in[sp]</td>
<td>80</td>
<td>50</td>
<td>01010000</td>
</tr>
<tr>
<td>london</td>
<td>18852</td>
<td>C9A4</td>
<td>11001001 10101000</td>
</tr>
<tr>
<td>?</td>
<td>32</td>
<td>20</td>
<td>01010000</td>
</tr>
<tr>
<td>[sp]</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>can't[sp]</td>
<td>146</td>
<td>8092</td>
<td>10000000 10010010</td>
</tr>
<tr>
<td>believe[sp]</td>
<td>239</td>
<td>80EF</td>
<td>10000000 11101111</td>
</tr>
<tr>
<td>i've[sp]</td>
<td>181</td>
<td>80B5</td>
<td>10000000 10110101</td>
</tr>
<tr>
<td>been[sp]</td>
<td>158</td>
<td>809E</td>
<td>10000000 11011110</td>
</tr>
<tr>
<td>back[sp]</td>
<td>145</td>
<td>8091</td>
<td>10000000 10100001</td>
</tr>
<tr>
<td>in[sp]</td>
<td>80</td>
<td>50</td>
<td>01010000</td>
</tr>
<tr>
<td>n</td>
<td>53</td>
<td>35</td>
<td>00110101</td>
</tr>
<tr>
<td>z</td>
<td>65</td>
<td>41</td>
<td>01000001</td>
</tr>
<tr>
<td>[sp]</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>for[sp]</td>
<td>84</td>
<td>54</td>
<td>01101000</td>
</tr>
<tr>
<td>nearly[sp]</td>
<td>2133</td>
<td>8855</td>
<td>10011000 01010101</td>
</tr>
<tr>
<td>l</td>
<td>18</td>
<td>12</td>
<td>00011001</td>
</tr>
<tr>
<td>s</td>
<td>25</td>
<td>19</td>
<td>00011100</td>
</tr>
<tr>
<td>[sp]</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>months</td>
<td>17023</td>
<td>C27F</td>
<td>11000010 01111111</td>
</tr>
<tr>
<td>,</td>
<td>13</td>
<td>0D</td>
<td>00001101</td>
</tr>
<tr>
<td>[sp]</td>
<td>1</td>
<td>01</td>
<td>00000001</td>
</tr>
<tr>
<td>but[sp]</td>
<td>100</td>
<td>64</td>
<td>01101000</td>
</tr>
<tr>
<td>i'm[sp]</td>
<td>83</td>
<td>53</td>
<td>01010011</td>
</tr>
<tr>
<td>still[sp]</td>
<td>220</td>
<td>80DC</td>
<td>10000000 11011100</td>
</tr>
<tr>
<td>loving[sp]</td>
<td>1474</td>
<td>85C2</td>
<td>10001101 11000010</td>
</tr>
<tr>
<td>every[sp]</td>
<td>309</td>
<td>8135</td>
<td>10000001 01101011</td>
</tr>
<tr>
<td>minute[sp]</td>
<td>361</td>
<td>8169</td>
<td>10000001 01101001</td>
</tr>
<tr>
<td>of[sp]</td>
<td>74</td>
<td>4C</td>
<td>01011000</td>
</tr>
<tr>
<td>it</td>
<td>16423</td>
<td>C027</td>
<td>11000000 00100111</td>
</tr>
<tr>
<td>Bits:</td>
<td>1253</td>
<td>1184</td>
<td>592</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------</td>
<td>------</td>
<td>-----</td>
</tr>
<tr>
<td>Bits Per Input Char:</td>
<td>7.00</td>
<td>6.61</td>
<td>3.31</td>
</tr>
<tr>
<td>Transmission Bits:</td>
<td>1592</td>
<td>1520</td>
<td>720</td>
</tr>
<tr>
<td>Transmission Bits Per Input Char:</td>
<td>8.89</td>
<td>8.49</td>
<td>4.02</td>
</tr>
<tr>
<td>Number of SMS Messages:</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Characters per SMS:</td>
<td>89.50</td>
<td>89.50</td>
<td>179.00</td>
</tr>
</tbody>
</table>

Experimental: JNI_OnLoad called.
Appendix E - Printout Ref# 0205

Details
=========================================
GSM Modem SerialPort: /dev/ttyUSB0
PhoneNumber: 07894555501
Message: Yes, but to be honest, when I am working on a problem I never think about beauty. I only think about how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong

Starting...

Stable Library
=========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGS=91
Rx: >
Rx:
Tx:
00110000B17098545505F100F4FF4D00C0780D0164475CC3270D01B8BD3001B80C081D85D280181683001B80B796AC79DF013001B05796A7F478A3448C1440F0164808D300156C70D0173488B1A4F5AC1B0D0013001524B4FC0EF
Rx: +CMGS: 60
Rx: OK

Sending done!

Waiting 60 seconds before trying to read message. This gives it time to be delivered from the GSM network...
OK, here we go.

Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGR=1
Rx: +CMGR: 0,96
Rx:
0791448720003023040C9144874955551000F4900192815503004D00C0780D0164475CC3270D01B8BD3001B80C081D85D280181683001B80B796AC79DF013001B05796A7F478A3448C1440F0164808D300156C70D0173488B1A4F5AC1B0D0013001524B4FC0EF
Rx: OK

Found Unread Message:
0791448720003023040C9144874955551000F4900192815503004D00C0780D0164475CC3270D01B8BD3001B80C081D85D280181683001B80B796AC79DF013001B05796A7F478A3448C1440F0164808D300156C70D0173488B1A4F5AC1B0D0013001524B4FC0EF
Rx: OK

Delete all messages in store as part of housekeeping
Tx: AT+CMGD=1
Rx: OK
Deleted message at index 1
Decompressed Message: yes, but to be honest, when I am working on a problem I never think
about beauty, I only think about how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong.

Receiving done!

Please wait, generating report...
Report generated.

SMS Compression Report
=================================================================================================

REF# 0205

Text Message [205 characters]: Yes, but to be honest, when I am working on a problem I never think about beauty. I only think about how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong.

Words Not Found in Dictionary [100.00 percent hit ratio]
=================================================================================================

[N/A - All words exist in dictionary]

Dictionary Symbol | Dictionary Index | Encoded Hex Byte(s) | Encoded Binary Byte(s)
--- | --- | --- | ---
[compression indicator] | 0 | 00 | 00000000
yes | 16504 | C07B | 11000000 01111000
, | 13 | 00 | 00001101
but | 100 | 64 | 01100100
to | 71 | 47 | 01000111
be | 92 | 5C | 01011100
honest | 17191 | C327 | 11000011 01001111
, | 13 | 00 | 00000101
when | 141 | 80BD | 10000000 10010010
i | 48 | 30 | 00110000
am | 192 | 80C0 | 10000000 11000000
working | 472 | 81DB | 10000001 11011000
a | 40 | 28 | 00101000
problem | 360 | 816B | 10000001 01010000
i | 48 | 30 | 00110000
never | 176 | 80B0 | 10000000 10110000
think | 121 | 79 | 01111001
about | 106 | 6A | 01101010
beauty | 16333 | C7D9 | 11000011 10011101
, | 15 | 0F | 00001111
i | 48 | 30 | 00110000
only | 213 | 80D5 | 10000000 11010101
think | 121 | 79 | 01111001
about | 106 | 6A | 01101010
how | 127 | 7F | 01111011
to | 71 | 47 | 01000111
solve | 2612 | 8A34 | 10001010 00110100
the | 72 | 48 | 01010000
problem | 16708 | C144 | 11000000 01000100
, | 15 | 0F | 00001111
i | 48 | 30 | 00110000
but | 100 | 64 | 01101000
when | 141 | 80BD | 10000000 10011010
i | 48 | 30 | 00110000
have | 86 | 56 | 01010110
finished | 17351 | C3C7 | 11000011 11000111
, | 13 | 0D | 00001101
i | 48 | 30 | 00110000

[sp] 1 01 00000001
know[sp] 82 52 01010010
it[sp] 75 4B 01001011
is[sp] 79 4F 01001111
wrong 16623 C0EF 11000000 11101111

REF# 0205 [205 chars] Uncompressed Huffman LD-Based
==================================================================
Bits: 1435 1112 608
Bits Per Input Char: 7.00 5.42 2.97
Transmission Bits: 1776 1232 736
Transmission Bits Per Input Char: 8.66 6.01 3.59
Number of SMS Messages: 2 1 1
Characters per SMS: 102.50 205.00 205.00

Experimental: JNI_OnLoad called.
Appendix F - Printout Ref# 0353

Details
========================================================================
GSM Modem SerialPort: /dev/ttyUSB0
PhoneNumber: 07894555501
Message: Hey mate, how are you? Meant to email you after a trip to cape town to let you know
i'd finally seen your beautiful city. What are you doing in london when you can call cape
town home? Was a nice change from life in the slums too. Hopefully coming over for a wedding
in April next year - will keep you posted. And would love you to come visit Australia!

Starting...

Stable Library
========================================================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC351
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGS=150
Rx: >
Tx: 0011000B817098545505F10DF4FF80008097CA150D017F67C022001828C47B09F4680F72801835A4798D081D5478
0BE4652810162681905E81E2C2DD0F14B674680D25089C880BD467680E598D081D5CD220015F2801812881CF80
9680C5D483A333C343A01C0EF0F018A35B147809542801B122509500B6581D90E01B0F90FB46D3040F014980
8B0B3464780B684C1DF5F602#
Rx: +CMGS: 61
Rx: OK

Sending done!

Waiting 60 seconds before trying to read message. This gives it time to be delivered from the
GSM network...

OK, here we go.
Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC351
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGR=1
Rx: +CMGR: 0,155
Rx: OK
Tx: AT+CPMS?
Rx: +CPMS: "ME",1,25,"SM",0,50,"SM",0,50
Rx: OK

Found Unread Message:
0791448720003023040C9144874955551000F4900192816594008008097CA150D017F67C022001828C47B09F468
0F72801835A4798D081D54780BE4652810162681905E81E2C2DD0F014E674680D25089C880BD467680E598D081D5
C0D220015F2801812881CF809680C55483A333C343A01C0EF0F018A35B147809542801B122509500B6581D90E0
1800F80FB46D3040F0149808B0B3464780B684C1DF5F602
Rx: OK

Delete all messages in store as part of housekeeping

66
Decompressed Message: hey mate, how are you? meant to email you after a trip to cape town to let you know i'd finally seen your beautiful city. what are you doing in london when you can call cape town home? was a nice change from life in the slums too. hopefully coming over for a wedding in april next year - will keep you posted. and would love you to come visit australia!

Receiving done!

Please wait, generating report...
Report generated.

SMS Compression Report

Text Message [353 characters]: Hey mate, how are you? Meant to email you after a trip to cape town to let you know i'd finally seen your beautiful city. What are you doing in london when you can call cape town home? Was a nice change from life in the slums too. Hopefully coming over for a wedding in april next year - will keep you posted. And would love you to come visit australia!

Words Not Found in Dictionary [98.00 percent hit ratio]
was a nice change from life.

too hopefully coming over for a wedding in april next year.

will keep you posted.

australia!

Experimental: JNI_OnLoad called.
Appendix G - Printout Ref# 0600

Details
========================================
GSM Modem SerialPort: /dev/ttyUSB0
PhoneNumber: 07894555501
Message: And so it was indeed: she was now only ten inches high, and her face brightened up
at the thought that she was now the right size for going through the little door into that
lovely garden. First, however, she waited for a few minutes to see if she was going to shrink
any further: she felt a little nervous about this; 'for it might end,' you know,' said Alice
to herself, 'in my going out altogether, like a candle. I wonder what I should be like then?'
And she tried to fancy what the flame of a candle is like after the candle is blown out, for
she could not remember ever having seen such a thing.

Starting...

Stable Library
========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGS=154
Rx: >
Tx: 0051000B17098545505F100F4FF8C0500037302010049634865FC87C1B01755F7B80D5829E9565C2300DD1497481A
329393022F3B2C352C2801777E4880D64A755F7B486B874854708114880AC820480D41AB410CB200F010CA0D01
C72AD0017586154280181AAB3F4780573755F70478B56C68E1B01758281280180AC84976AC02D010544
B8133C01AC0D0146C02E0D080180CB6C#
Rx: +CMGS: 62
Rx: OK
Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGS=108
Rx: >
Tx: 0051000B17098545505F100F4FF5E0500037302027347C3AA0D01085057706FBA9F0D01728101CD810F01300182E
84E300180C95C71C0B01200801497582024788C46E4895DD4C29018DA54F7180F7488DA54F8BF3C04B0D015475B09A
5A811B0EA8174819081B92801C09B0F#
Rx: +CMGS: 63
Rx: OK

Sending done!

Waiting 60 seconds before trying to read message. This gives it time to be delivered from the
GSM network...
OK, here we go.

Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC35i
Rx: OK
Tx: AT+CMGF=0
Found Unread Message:

Delete all messages in store as part of housekeeping

Decompressed Message: and so it was indeed: she was now only ten inches high, and her face brightened up at the thought that she was now the right size for going through the little door into that lovely garden. First, however, she waited for a few minutes to see if she was going to shrink any further: she felt a little nervous about this; 'for it might end, you know,' said Alice to herself, 'in my going out altogether, like a candle. I wonder what I should be like then?' and she tried to fancy what the flame of a candle is like after the candle is blown out, for she could not remember ever having seen such a thing.

Receiving done!

Please wait, generating report...

SMS Compression Report

---

**REF# 0600**

Text Message [600 characters]: And so it was indeed: she was now only ten inches high, and her face brightened up at the thought that she was now the right size for going through the little door into that lovely garden. First, however, she waited for a few minutes to see if she was going to shrink any further: she felt a little nervous about this; 'for it might end, you know,' said Alice to herself, 'in my going out altogether, like a candle. I wonder what I should be like then?' and she tried to fancy what the flame of a candle is like after the candle is blown out, for she could not remember ever having seen such a thing.

Words Not Found in Dictionary [99.00 percent hit ratio]

---

```
<table>
<thead>
<tr>
<th>Dictionary Symbol</th>
<th>Dictionary Index</th>
<th>Encoded Hex Byte(s)</th>
<th>Encoded Binary Byte(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>and[sp]</td>
<td>73</td>
<td>00</td>
<td>00000000</td>
</tr>
<tr>
<td>so[sp]</td>
<td>99</td>
<td>63</td>
<td>01001001</td>
</tr>
<tr>
<td>it[sp]</td>
<td>75</td>
<td>4B</td>
<td>01001001</td>
</tr>
<tr>
<td>was[sp]</td>
<td>95</td>
<td>5F</td>
<td>01011111</td>
</tr>
<tr>
<td>indeed</td>
<td>18556</td>
<td>C87C</td>
<td>11001000 01111100</td>
</tr>
</tbody>
</table>
```

---

The University of Cape Town
nervous

little

a

felt

she

:

further

shrink
to

few

for

waited

she

, however

lovely
garden

. first

thought

that

she

was

now

the

now

the

at

up

for

size

right

the

now

the

at

up

d
n
e
t
h
g
b

few

minutes
to

see

if

she

was

going

to

shrink

any

further

: 

[sp] 1 0 0

she[sp] 117 75 0 01110101

was[sp] 95 5F 0 01111111

now[sp] 123 7B 0 01110111

only[sp] 213 80D5 1 10000000 11010101

ten[sp] 670 829E 1 10000010 10011110

inches[sp] 5477 9565 1 10010101 01100101

high 16944 C230 1 11000010 01100000

. 13 0D 0 00001101

[sp] 1 0 0

and[sp] 73 49 0 01011000

her[sp] 116 74 0 01110100

face[sp] 419 81A3 1 10000001 10100011

b 41 29 0 01010001

r 59 39 0 00111101

i 48 30 0 00111000

g 46 2E 0 00111110

h 47 2F 0 00111111

t 59 3B 0 01110111

n 53 35 0 00111010

e 44 2C 0 01101000

d 43 2B 0 01010011

[sp] 1 0 0

up[sp] 119 77 0 00111101

at[sp] 126 7E 0 00111110

the[sp] 72 48 0 01000100

thought[sp] 214 80d6 1 10000000 11011110

that[sp] 74 4A 0 01001010

she[sp] 117 75 0 00111011

was[sp] 95 5F 0 01111111

now[sp] 123 7B 0 01110111

the[sp] 72 4B 0 10101000

right[sp] 107 6B 0 01101011

size[sp] 1864 8748 1 10001111 01100100

for[sp] 84 54 0 01010000

going[sp] 112 70 0 01110000

through[sp] 276 8114 1 10000000 00110100

the[sp] 72 48 0 10101000

little[sp] 172 80AC 1 10000000 10101100

doors[sp] 516 8204 1 10000000 10110010

into[sp] 222 80DE 1 10000000 11011110

that[sp] 74 4A 0 10101011

lovely[sp] 1073 8431 1 10010011 00100000

garden 19232 CB20 1 11001011 00010000

. 15 0F 0 00011111

[sp] 3 0 0 00000000

first 16586 C0CA 1 11000000 11010100

, 13 0D 0 00001111

[sp] 1 0 0 00000000

however 18218 C72A 1 11001111 00101010

, 13 0 0 00001111

she[sp] 117 75 0 01110101

waited[sp] 2145 8861 1 10010000 01100000

for[sp] 84 54 0 01010000

a 40 28 0 01010000

[sp] 1 0 0 00000000

few[sp] 427 81AB 1 10000001 10101011

minutes[sp] 501 81F5 1 10000000 11111011

to[sp] 71 47 0 01000101

see[sp] 133 8085 1 10000000 10001001

if[sp] 115 73 0 01110110

she[sp] 117 75 0 01110101

was[sp] 95 5F 0 01111111

going[sp] 112 70 0 01110000

to[sp] 71 47 0 01000111

shrink[sp] 3045 8BE5 1 10010101 11101011

any[sp] 205 80CD 1 10000000 11010101

further 18062 C6BE 1 11001110 10011110

: 27 1B 0 00111011

[sp] 1 0 0 00000001

she[sp] 117 75 0 01110101

felt[sp] 641 8281 1 10000001 10000000

a 40 28 0 01010000

[sp] 1 0 0 00000000

little[sp] 172 80AC 1 10000000 10101000

nervous[sp] 1175 8497 1 10001000 11011111

about[sp] 106 6A 0 10101100
this 16429  C02D 11000000 00101101
; 28  1C 00011100
, 1  01 00000001
for 84  54 00010100
it 75  4B 01010111
might 307  8133 10000001 00110011
end 16812  C1AC 11000001 10101100
, 13  00 00011101
, 1  01 00000001
you 70  46 01000110
know 16430  C02E 11000000 00111010
, 13  00 00011101
, 8  08 00010100
in 80  50 01010000
my 87  57 01010111
going 112  70 01100000
out 114  6F 01101111
altogether 23319  DA9F 10110110 10011111
, 13  00 00011101
, 1  01 00000001
like 113  71 01100001
a 40  28 00101000
[sp] 1  01 00000001
candle 19841  CD81 11001110 10000001
, 15  0F 00001111
i 48  30 01100000
[sp] 1  01 00000001
wonder 744  82E8 10000010 11101000
what 78  4E 01011110
i 48  30 01100000
[sp] 1  01 00000001
should 201  80C9 10000000 11000101
be 92  5C 01011100
like 113  71 01100001
then 16513  C081 11000000 10000001
? 32  20 00100000
[sp] 8  08 00010100
and 73  49 01010001
she 117  75 01110101
tried 523  820B 10000010 00010111
to 73  47 01000011
fancy 2244  88C4 10001000 11111000
what 78  4E 01011110
the 72  4B 01010100
flame 5853  9EDC 10101110 11101111
of 76  4C 01011100
a 40  28 00101000
[sp] 1  01 00000001
candle 3493  8DA5 10001101 10101011
is 79  4F 01011111
like 113  71 01110001
after 247  80F7 10000000 11110111
the 72  4B 01010100
candle 3493  8DA5 10001101 10101011
is 79  4F 01011111
blown 3059  8BF3 10010111 11110011
out 16459  C04B 11000000 00101101
, 13  00 00011101
[sp] 1  01 00000001
for 84  54 01010100
she 117  75 01110101
could 154  809A 10000000 11010111
not 90  5A 01011110
remember 286  81E8 10000001 00011110
ever 234  80EA 10000000 11101100
having 372  8174 10000001 01110100
seen 400  8190 10000001 10010000
such 441  81B9 10000001 10111001
<table>
<thead>
<tr>
<th>REF# 0600 [600 chars]</th>
<th>Uncompressed</th>
<th>Huffman</th>
<th>LD-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits:</td>
<td>4200</td>
<td>2768</td>
<td>1768</td>
</tr>
<tr>
<td>Bits Per Input Char:</td>
<td>7.00</td>
<td>4.61</td>
<td>2.95</td>
</tr>
<tr>
<td>Transmission Bits:</td>
<td>4880</td>
<td>3272</td>
<td>2112</td>
</tr>
<tr>
<td>Transmission Bits Per Input Char:</td>
<td>8.13</td>
<td>5.45</td>
<td>3.52</td>
</tr>
<tr>
<td>Number of SMS Messages:</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Characters per SMS:</td>
<td>150.00</td>
<td>200.00</td>
<td>300.00</td>
</tr>
</tbody>
</table>

Experimental: JNI_OnLoad called.
Details
=========================================
GSM Modem SerialPort: /dev/ttyUSB0
PhoneNumber: 07894555501
Message: We understand it still that there is no easy road to freedom. We know it well that none of us acting alone can achieve success. We must therefore act together as a united people, for national reconciliation, for nation building, for the birth of a new world. Let there be justice for all. Let there be peace for all. Let there be work, bread, water and salt for all. Let each know that for each the body, the mind and the soul have been freed to fulfill themselves. Never, never and never again shall it be that this beautiful land will again experience the oppression of one by another and suffer the indignity of being the skunk of the world. Let freedom reign. The sun shall never set on so glorious a human achievement!

Starting...

Stable Library
=========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC351
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGS=154
Rx: >
Tx:
0051000B817098545505F100F4FF8C050003040201006081264880DC4A8814F55820883E347CBB10F0160524B664A82F54C80AB8423817F76987BC93BF0160814687D831B8121808A280188ACEB806D0154833BC7BDD0154926BC3A8D015487575C32018117C1470F0180BE80815C80410F0180BE80815CCDCFDD01C8A30D01829C498E7554C0410F#
Rx: +CMGS: 64
Rx: OK
Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC351
Rx: OK
Tx: AT+CMGF=0
Rx: OK
Tx: AT+CMGS=145
Rx: >
Tx:
0051000B817098545505F100F4FF830500030402020180BE81A7524A5481A748C286DD014881344948846256889EA268479D9C6AD0F10C8C0D0180B0498B080E183B4B54A5181E285E6808F80E184E488B7C4C808280C7813F498A274830352302E35303B40014C811848B1514C48C1470F0180BE88D5E4CEEF01488654838B0B082055D63961428018355E844D2#
Rx: +CMGS: 65
Rx: OK

Sending done!

Waiting 60 seconds before trying to read message. This gives it time to be delivered from the GSM network...

OK, here we go.

Tx: ATE0
Rx: OK
Tx: AT+CGMI
Rx: SIEMENS
Rx: OK
Tx: AT+GMM
Rx: TC351
Found Unread Message:

Delete all messages in store as part of housekeeping

Decompressed Message: we understand it still that there is no easy road to freedom. we know it well that none of us acting alone can achieve success. we must therefore act together as a united people, for national reconciliation, for nation building, for the birth of a new world. let there be justice for all. let there be peace for all. let there be work, bread, water and salt for all. let each know that for each the body, the mind and the soul have been freed to fulfill themselves. never, never and never again shall it be that this beautiful land will again experience the oppression of one by another and suffer the indignity of being the skunk of the world. let freedom reign. the sun shall never set on so glorious a human achievement!

Receiving done!

Please wait, generating report...

Report generated.

SMS Compression Report

Text Message [723 characters]: We understand it still that there is no easy road to freedom. We know it well that none of us acting alone can achieve success. We must therefore act together as a united people, for national reconciliation, for nation building, for the birth of a new world. Let there be justice for all. Let there be peace for all. Let there be work, bread, water and salt for all. Let each know that for each, the body, the mind and the soul have been freed to fulfill themselves. Never, never and never again shall it be that this beautiful land will again experience the oppression of one by another and suffer the indignity of being the skunk of the world. Let freedom reign. The sun shall never set on so glorious a human achievement!

Words Not Found in Dictionary [99.00 percent hit ratio]
let[sp] 190 80BE 10000000 10111110
there[sp] 129 80B1 10000000 10000001
be[sp] 92 5C 01011100
work 16591 C0CF 11000000 11001111
, 13 0D 00001101
[sp] 1 01 00000001
bread 19369 CBA9 11010011 10101001
, 13 0D 00001101
[sp] 1 01 00000001
water[sp] 668 829C 10000010 10011100
and[sp] 73 49 01001001
salt[sp] 3701 8E75 10011110 01110101
for[sp] 84 54 01010100
all 16449 CO41 11000000 01000001
. 15 0F 00001111
[sp] 1 01 00000001
let[sp] 190 80BE 10000000 10111110
each[sp] 423 81A7 10000001 10100111
know[sp] 82 52 01010010
that[sp] 74 4A 01001010
for[sp] 84 54 01010100
each[sp] 423 81A7 10000001 10100111
the[sp] 72 48 01010000
body 17030 C286 11000010 10011110
, 13 0D 00001101
[sp] 1 01 00000001
never 16524 CO8C 11000000 10111100
, 13 0D 00001101
[sp] 1 01 00000001
never[sp] 176 80B0 10000000 10100100
and[sp] 73 49 01010010
never[sp] 176 80B0 10000000 10110000
again[sp] 225 80E1 10000000 11100001
shall[sp] 907 838B 10000011 10001101
it[sp] 75 4B 01011100
be[sp] 92 5C 01011100
that[sp] 74 4A 01001010
this[sp] 81 51 01010001
beautiful[sp] 482 81E2 10000011 11001100
land[sp] 1510 85E6 10010010 11100110
will[sp] 143 808F 10000000 10001111
again[sp] 225 80E1 10000000 11100001
experience[sp] 1262 84BE 10000010 11100110
the[sp] 72 48 01010000
oppression[sp] 15996 BB7C 11011110 00111100
of[sp] 76 4C 01001100
one[sp] 130 80B2 10000000 10000000
by[sp] 199 80C7 10000000 11000011
another[sp] 319 813F 10000001 00111101
and[sp] 73 49 01010010
suffer[sp] 2599 8A27 10000010 00101101
the[sp] 72 48 01010000
i 48 30 00101100
n 53 35 00110101
d 43 2B 00101100
i 48 30 00101100
g 46 2E 00111100
n 53 35 00110101
i 48 30 00101100
t 59 3B 00111011
y 64 40 01000000
[sp] 1 01 00000001
of[sp] 76 4C 01001100
being[sp] 280 8118 10000001 00110000
the[sp] 72 48 01010000
skunk[sp] 12625 B151 10110001 01001001

77
of[sp] 76 4C 01001100
do[sp] 72 48 01001000
world 16711 C147 11000001 01000111
.
[sp] 1 0F 00001111
let[sp] 190 80BE 10000000 10111110
freedom[sp] 2261 88D5 10001000 11010101
reign 25806 E4CE 11100100 11001110
.
[sp] 1 01 00000001
the[sp] 72 48 01001000
sun[sp] 1620 8654 10000110 01010100
shall[sp] 907 838B 10000011 10001011
never[sp] 176 80B0 10000000 10110000
set[sp] 517 8205 10000000 10110000
on[sp] 93 5D 01011101
so[sp] 99 63 01100011
 glorious[sp] 5652 9614 10010110 00010100
.
[sp] 1 01 00000001
human[sp] 853 8355 10000011 01010101
achievement 26692 E844 11101000 01000100
!
[sp] 2 02 00000010

REF# 0723 [723 chars] Uncompressed Huffman LD-Based
==================================================================
Bits: 5061 3016 2064
Bits Per Input Char: 7.00 4.17 2.85
Transmission Bits: 5912 3520 2408
Transmission Bits Per Input Char: 8.18 4.87 3.33
Number of SMS Messages: 5 3 2
Characters per SMS: 144.60 241.00 361.50

Experimental: JNI_OnLoad called.