Measuring Comprehension of a Text Compression Algorithm for SMS

Alexander Shenton

14 March 2001

This is the report on a project submitted for the degree of Computer Systems and Software Engineering in the Department of Computer Science at the University of York

Measuring Comprehension of a Text Compression Algorithm for SMS

Abstract

This report details the design and testing of a text compression algorithm for SMS messages. Few existing algorithms could produce compressed output that was readable without any training. The method employed here was a simple two-stage process, making common abbreviations and then stripping the remaining of words of all but leading and trailing vowels. The algorithm was written in Perl.

A questionnaire, presented as a web page, was used to measure the accuracy and speed of comprehension of 80 subjects. 15 of these 80 were presented uncompressed statements as a control. The results show, that the algorithm produces around 25% compression, but that comprehension speed and accuracy for the compressed statements are significantly less than speed and accuracy for the uncompressed statements.

Measuring Comprehension of a Text Compression Algorithm for SMS

Contents

Measuring Comprehension of a Text Compression Algorithm for SMS

Measuring Comprehension of a Text Compression Algorithm for SMS

Preamble

Modern devices with small displays, such as PDAs (Personal Digital Assistants), mobile telephones (particularly those with WAP capability) and GPS (Global Positioning System) receivers, could benefit from some method of fitting the maximum amount of information on to the screen at one time. The task was to devise and evaluate an algorithm that could compress text so that more can be fit on the screen while retaining readability, i.e. maximising information while minimising characters.

The most common form of small display is the ubiquitous mobile telephone. SMS (Short Message Service) is a text messaging system defined within the digital GSM mobile `phone standard, and is a popular system with high public exposure. It has various properties, but the most relevant to this project is the fact that SMS text messages are limited to 160 characters.

SMS is a technology with massive market infiltration. In the last two years, mobile messaging has really taken off, particularly since the introduction of pre-pay mobile telephones. In December 2000 an estimated 20 billion text messages were sent worldwide, a seven-fold increase on the same period the previous year. Monthly figures are increasing an average 1000% year-on-year (All figures from the Guardian).

However, though 24 million messages are sent in Britain each day, there are definite limits to its usefulness. 160 characters, though it may sound a lot, is actually quite limiting in terms of content. While no one is likely to be sending novellas by text message, it can be difficult to fit much information into a single text message. For this reason, SMS users frequently abbreviate words and use shortened phrases that can, to the uninitiated, look like complete gobbledegook. An automated system that could compress messages, while retaining readability so that comprehension is high even among infrequent ‘texters’, might be useful.

Measuring Comprehension of a Text Compression Algorithm for SMS

Initial Thoughts

First it should be specified just what is meant by compression. In this case, the term refers to the minimisation of the number of *characters* in a message, while retaining a *readable* message.

The following quote is taken from http://www.utpress.utoronto.ca/editorial/castingoff.htm:

> *"Character counts are actual, whereas word counts are hypothetical" (Martin, Book Design: A Practical Introduction, Van Nostrand Reinhold, 1989, pg. 115). Although the average length of the English word plus the space which follows it is generally given to be six characters, in actual fact it will vary depending on the reading level of the text. A scholarly text will typically have longer words than most books for general readership. Therefore, word counts can fluctuate wildly depending on which standard is being used and are generally inaccurate.*

All compression rates in this document are measured by character, for the reasons given above and because mobile telephones typically have character-based LCD displays.

The type of information to be compressed has a bearing on the algorithm: for example, if compressing place names for a GPS receiver, a simple database of common abbreviations for the places might be sufficient (e.g. B'ham for Birmingham, Salop for Shropshire). This may soon become unpractical if we expand into the more general English language of SMS messages.

A major problem is that the English language is extremely irregular. If we look at the simple mechanical process involved in the two place-name examples above, we can see that that the rules used there are very specific. 'Salop' has virtually no relation to the word 'Shropshire', being a common abbreviation for historical reasons. Giving the algorithm a tutoring in history and social culture is obviously not a particularly useful suggestion.

On the other hand, because it is descended from so many sources, English is full of redundancy. American English has, in many words, stripped away some of this character redundancy. Consider the American 'color' versus the international English 'colour'. While some might prefer the English spelling, the American version conveys the same information in fewer characters.

Measuring Comprehension of a Text Compression Algorithm for SMS

Reading

Existing Algorithms

There are a few existing text compression systems. I was able to find the following three:

Email2SMS

This program was written in Perl by Adam Spiers using the Lingua::EN::Squeeze module (documentation for which can be found at www.new.ox.ac.uk/~adam/computing/email2sms/Lingua_EN_Squeeze.html) The homepage for the program itself is http://www.new.ox.ac.uk/~adam/computing/email2sms/

The result achieved by this program is something of the form:

> comp.lang.perl.announce
>
> moderator$[thiIsAutmtedMsgFromPrlScript.]/Postng
>
> 4Cmp.lng.prl.nnnceHasBenRcivAndQueu.l/wSndUMsgWhenIChoseT
>
> oEithrAcceptOr
>
> RejectPostngITryToProcPostngAtLstOncEverySevenDD,ButGenrlyIt
>
> HasBenLot+/
>
> OftnThan(wthnMINIfImOn-lin,F_xmple).AprovedChartSay

(this example is taken from the email2sms page). While this has the advantage of high compression rates, (up to 60% is claimed) it is really quite unreadable and confusing at first glance.

The compression rate is variable. In the following examples, taken from the lingua::en::squeeze module documentation page, we go from zero compression:

> *You can use this module e.g. to preprocess text before it is sent to electronic media that has some maximum text size limit. For example pagers have an arbitrary text size limit, typically 200 characters, which you want to fill as much as possible. Alternatively you may have GSM cellular phone which is capable of receiving Short Messages (SMS), whose message size limit is 160 characters. For demonstration of this module's SqueezeText() function , the description text of this paragraph has been converted below. See yourself if it's readable (Yes, it takes some time to get used to). The compress ratio is typically 30-40%*

…to medium compression:

> *u _n use thi mod e.g. to prprce txt bfre i_s snt to*
> *elrnic mda has som max txt siz lim. f_xmple pag*
> *hv abitry txt siz lim, tpcly 200 chr, W/ u wnt*
> *to fll as mch as psbleAlternatvly u may hv GSM cllar P8*
> *w_s cpble of rcivng Short msg (SMS), WS/ msg siz*
> *lim is 160 chr. 4 demonstrton of thi mods SquezText*
> *fnc , dsc txt of thi prgra has ben cnvd_ blow*
> *See uself if i_s redble (Yes, it tak som T to get usdto*
> *compr rat is tpcly 30-40*

…to full compression:

> *u_nUseThiModE.g.ToPrprceTxtBfreI_sSntTo*
> *elrnicMdaHasSomMaxTxtSizLim.F_xmplePag*
> *hvAbitryTxtSizLim,Tpcly200Chr,W/UWnt*
> *toFllAsMchAsPsbleAlternatvlyUMayHvGSMCllarP8*
> *w_sCpbleOfRcivngShortMsg(SMS),WS/MsgSiz*
> *limIs160Chr.4DemonstrtonOfThiModsSquezText*
> *fnc,DscTxtOfThiPrgraHasBenCnvd_Blow*
> *SeeUselfIfI_sRedble(Yes,ItTakSomTToGetUsdto*
> *comprRatIsTpcly30-40*

Training improves ones ability to read messages compressed in this way, but this is of no use to first time users. The algorithm to be designed should produce more instantly accessible outputs, which could be read with practically no practice.

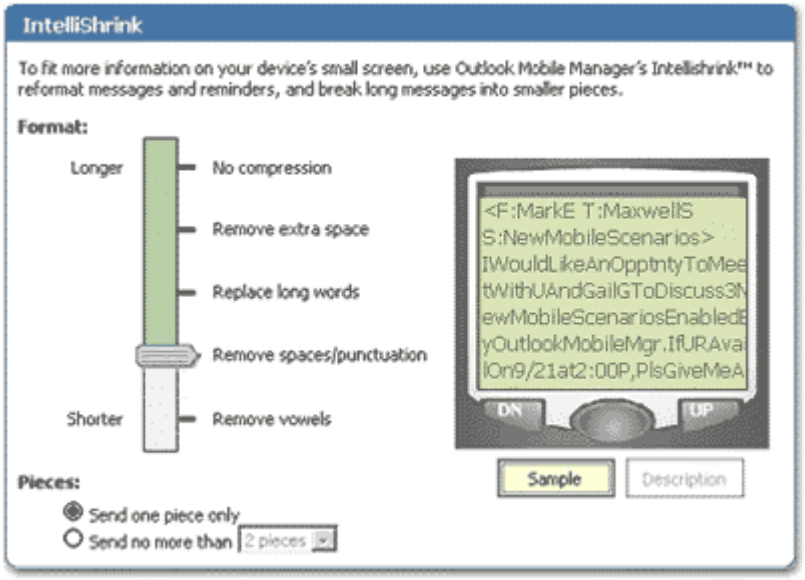
Measuring Comprehension of a Text Compression Algorithm for SMS

Microsoft Intellishrink

Microsoft has devised a plug-in for Outlook (part of Microsoft Office) to deal with mobile devices. This plug-in, still in beta form at time of writing, is called Microsoft Intellishrink, and can be used to compress messages in a very similar way to that intended for this project. The following is from the Microsoft Outlook Mobile Manager web page located at http://www.microsoft.com/office/outlook/mobile/default.htm:

> ***Benefit from Mobility Made Simple***
> *Using natural language processing, Outlook Mobile Manager automatically compresses large messages so you receive only the most important message data on your mobile device. Microsoft Intellishrink™, a text compression system, lets you choose options such as removing spaces, replacing long words with known abbreviations, removing punctuation, and even removing vowels.*

A sample of the compressed text can be seen in the following image, also from the same page:



Microsoft's system seems to be designed for forwarding emails to mobile devices, and employs an interesting and simple scalable compression algorithm. This algorithm was discovered *after* the algorithm used for this project was devised, and any similarities are coincidental.

Measuring Comprehension of a Text Compression Algorithm for SMS

AmikaNow! Highlights

AmikaNow! Corp has developed a system using a different idea to simple word-by-word compression. The 'Highlights' system automatically summarises emails for communication to mobile devices. The following article is from Computerwire - October 26, 2000, found on AmikaNow!'s web site at www.amikanow.com/newsroom/NewsPostings/wowfi/WOW!Wireless.htm:

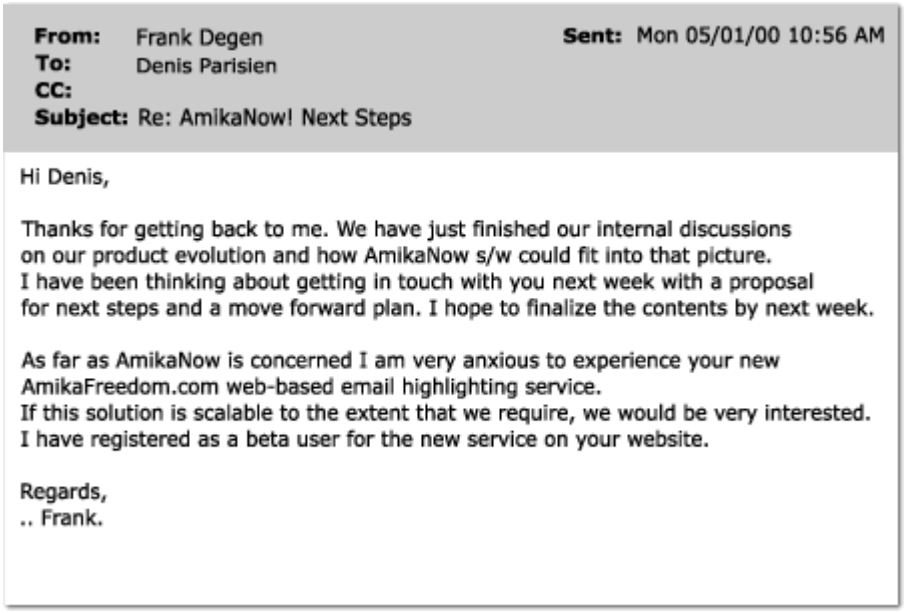
> *AmikaNow! Corp claimed a breakthrough in wireless email delivery at Internet World Fall 2000 on Wednesday, demonstrating a system that can automatically summarize an email and forward the results to a wireless device.*
>
> *The software uses artificial intelligence to precis the contents of an email so that it [can] be sent as a short message - or several short messages - of around 15 words. This means that the information can be sent to, and easily read, on an SMS or WAP phone or an interactive pager. The technology is expected to be on the market in the first quarter of next year.*
>
> *The AI agent looks for patterns in emails and summarizes what it considers is important content for the wireless message system. "A lot of English text and style of writing is redundant," said AmikaNow's founder, president and CEO,* ***Sue Abu-Hakima****, explaining how a 250-word email can be boiled down to a 160 character text message. In case the user wants more detail, the system can also forward the entire email, if requested.*
>
> *AmikaNow will offer the service through its AmikaFreedom.com web site, which can pull emails from corporate servers as long as they support the POP3 or IMAP mail protocols, Abu-Hakima said. The software can also be downloaded and used as a Microsoft Exchange plug-in, which might be preferable for users behind a firewall. She said she expects to see the AI engine built into unified messaging systems eventually.*

Measuring Comprehension of a Text Compression Algorithm for SMS

An example of the Highlights system can be found at https://www.amikafreedom.com/afc1/amikafreedom.com/example.htm

It is reproduced here.

Original email:



From: Frank Degen **Sent:** Mon 05/01/00 10:56 AM
To: Denis Parisien
CC:
Subject: Re: AmikaNow! Next Steps

Hi Denis,

Thanks for getting back to me. We have just finished our internal discussions on our product evolution and how AmikaNow s/w could fit into that picture. I have been thinking about getting in touch with you next week with a proposal for next steps and a move forward plan. I hope to finalize the contents by next week.

As far as AmikaNow is concerned I am very anxious to experience your new AmikaFreedom.com web-based email highlighting service. If this solution is scalable to the extent that we require, we would be very interested. I have registered as a beta user for the new service on your website.

Regards,
.. Frank.

Highlights generated:

finished internal product evolution AmikaNow s w fit picture...AmikaNow concerned am anxious experience new AmikaFreedom.com web-based email highlighting service./../

While this is an interesting system, it is more appropriate for converting emails to SMS than conversion of SMS messages sent between telephones.

Of the programs surveyed here, the Intellishrink system is closest to the system desired. Indeed, the final algorithm used here is very similar to Intellishrink in operation.

No literature was found on the measuring of comprehension of compressed text.

Measuring Comprehension of a Text Compression Algorithm for SMS

How Humans Read

It seemed logical to consult texts on reading in order to better understand the process that the algorithm would be catering for.

The process of reading is not fully understood at anything above the purely mechanical level. As we read, our eyes skitter across the page or screen in jumpy steps called saccades, fixating on lumps of text of about 8-10 characters in size (though this is obviously dependent on the size of the text) at a time before moving on. Interestingly, it appears that a serial scan of characters (i.e. one in which characters are read one at a time until a whole word is created, which is then comprehended) does **not** take place. Indeed, evidence suggests that in some cases whole words are understood *faster* than lone characters, which implies that some kind of physical pattern matching takes place – that is, the *shape* of the word helps us to comprehend it. This is supported by experiments that show that words in all capital letters are read more slowly than words in all lower case – we are more familiar with words in lower case, so we match them faster.

There is evidence that the sound of the words we read is relevant to the speed at which we read them. Unpronounceable 'words' like 'kwjkhgy' are read far more slowly than pronounceable pseudowords such as 'fronk'.

> "*Baron and Thurston (1973), showed that performance* [of reading speed and comprehension] *on legal nonwords is very good indeed, relative to true words, and vastly better than performance on illegal nonwords*." – Psychology of reading, p87.

It should be noted here that this experiment was (obviously) performed without performing any meaning analysis. The test subjects were simply memorising the words and repeating them. These non-words had no meaning, whereas the non-words produced in compression of English text will have meaning. Both decoding and meaning analysis require attention.

> "*The point of the … argument is that if the decoding half of the conflict could be made truly automatic, then attention could be reserved for where it belongs… with the meaning."* – Psychology of reading, p106.

This fact could be exploited – words that still 'flow' after compression would be more easily read than those that are completely unpronounceable would. Unfortunately, this could also have detrimental effects, both in reduced compression and in the fact that a word that has been compressed in such a way as to render it more readable might be more easily mistaken for another word when it reaches the comprehension process. Perhaps it would be better to ignore this rather artificial method of compression. Since read words are sub-vocalised, it might be better to attempt to retain the original sound of the word in the compressed artefact.

There are a few methods of doing this. Journalistic shorthand uses a system that encodes words by their phonemes, massively compressing them. Unfortunately, it requires extensive training to be able to read and comprehend shorthand, whereas we want the reader to be able to comprehend compressed messages with absolutely no training (they may improve with practice, but it is important that initial comprehension be high). A simple system would be to strip vowels from words. In most words, consonants provide the bulk of the sound, with vowels only filling out the sound – for example, 'willing' is still perfectly readable as 'wllng'. This has the advantage that much of the visual appearance of the original word is preserved, which may help readers in the 'pattern-matching' part of the reading process. In some words vowels are a greater part of the structure, particularly where they begin the word – for example the 'a' in 'atypical' completely changes the meaning (negating it) and sound of the word. Therefore, it would be wise to retain leading vowels, in order to keep comprehension high.

Sometimes trailing vowels, too, form a more important part of the structure: take for example the words 'compost' and 'composite'. Removing the vowels from these gives 'cmpst' in both cases, which increases the cognitive burden on the reader as they have to use contextual knowledge to work out what the compressed string represents. In this case, of course, the two meanings are quite different, and this problem is liable to be quite simple. However, keeping the trailing 'e' and the problem is rendered simpler. There are a large number of words in the English language that are terminated in an 'e', and appending an 'e' can frequently change the meaning and sound of a word, such as

sit →site

(primary school children learning English are taught about 'magic e' because of this property.) Note that in the table below, 'cmpste' could still mean 'campsite', but hopefully other words will be rendered unambiguous!

Of course, 'e' is not the only vowel in which words may end. Words of Latin origin, such as quota, also possess a trailing vowel that alters their sound. The table shows several possible confusions that can arise from a lack of a trailing vowel in this case.

Measuring Comprehension of a Text Compression Algorithm for SMS

The effect of leading and trailing vowels

Word	*All vowels removed*	*Vowels removed but leading & trailing vowels retained*
atypical	typcl	atypcl
compost	cmpst	cmpst
composite	cmpst	cmpste
quit	qt	qt
quota	qt	qta
quite	qt	qte

Comprehension of words is aided by the context in which they are found. A phenomenon known as ‘syntactic priming’ enables us to comprehend better words that are contextually related to recently heard words. This is a useful fact because it means that if two words compress to the same thing, then the context in which they were sent should assist comprehension – this is, of course, already used in English where we frequently come across words with two or more meanings, but are able to ascertain the correct one from context. For example, the word ‘set’ has a multitude of meanings: ‘Set the table’, ‘Train set’, ‘Game, set and match’, ‘The jelly has set’ etc.

Measuring Comprehension of a Text Compression Algorithm for SMS

The Algorithm

The intention is result oriented: we do not care how we get there, but we do care what the result is. Given an input string, the algorithm should output a string with the following properties.

- Maximum possible compression in order that more can be fit on to a screen or into a single SMS message at one time. This is the same for our purposes as maximum information density, which is frequently best altered by changing the structure of a sentence. To do this automatically would require complicated natural language processing.

- Maximum possible 'readability'. More specifically, high comprehension rates by our test subjects.

- Maximum possible speed/ ease of reading. Though this is linked to comprehension, it is not inextricable. It might be possible to send a message that can be deciphered with one hundred percent comprehension, but which takes eight hours with a pen and pencil to decipher (the simplest example of this would be a system whereby every word in the dictionary is numbered, and the message consists of these numerical codes. This would give high compression and comprehension but would be utterly impractical.) The algorithm is aimed at 'the layman', and should produce results that are readable without any special prior knowledge or training.

Choices

Given the information we have about reading, the algorithm could follow several possible directions. Most interesting of those considered are:

1. **A phoneme based system.**

2. **A simple vowel removal system.**

3. **A hybrid system.**

All of these options result in some loss of information. In every case the compression process is a unidirectional function: no inverse function exists to decompress their output to the unique, original input. This is to be expected from a simple encoding system.

Measuring Comprehension of a Text Compression Algorithm for SMS

Phoneme based compression

A phoneme-based system would preserve the sound of words by putting shortened phoneme representations in the place of character strings that represented those phonemes.

For example, the commonly used abbreviation of ‘later’ uses a phoneme system (shown in table below) and the common abbreviation of ‘see’ is ‘C’, in which a single character replaces the three previously employed. Use of the American pronunciation for the letter ‘Z’ (‘zee’, as opposed to the British ‘zed’) is also common, as in the last two entries in the table below, the latter of which is famously used by the LaZ Boy (Lazy Boy) furniture company. The ‘zee’ sound crops up frequently in the language, whereas the syllable ‘zed’ is extremely rare (in such words as ‘grazed’ and ‘amazed’ the ‘zed’ section is pronounced more like ‘zud’ than ‘zed’, missing out the vowel sound almost completely).

Examples of Phoneme Compression

Word	***Compression using a phoneme system***
later	l8r (or l8er)
see	C
seedy	CD
random	r&om
easy	EZ
lazy	LaZ

Phoneme parsers exist that could examine strings and then output the phonemes that make up these strings. Such parsers are used in text-to-speech synthesizers, for example.

It is important for the purposes of this project that the resulting output is easy to read by an untrained user, so a heavily encoded phoneme string would not be ideal. As previously mentioned, this is very much like journalistic shorthand, which takes years of training to learn. Instead, the phoneme stream would have to be combined with the original string in the way used in the examples above, i.e. one in which the reader forms the originally intended sounds when he sub-vocalises the string.

The output from a phoneme parser could be passed to a find/replace algorithm that replaced certain phonemes with simple, easily understood representations. The output from this would then be combined with the original string input to produce results similar to those above.

Using this, we should achieve something like this:

'Wait for me!' → *'w8 4 me!'*: Compression rate = 33%

Or, even better:

'*See you for tea at eight*' → '*C U 4 T @ 8'*: Compression rate = 54.2%

One problem with this system is that there are few phonemes that compress to a simple, single character, representation, so the compression that we could achieve is not as high as we might imagine (though with certain phrases and a good choice of words it can be very high, as seen above). In addition, the output can start to look badly spelt, rather than compressed, if we strive for a high compression rate. This is dependant on the precise implementation chosen. It may be possible to avoid this effect, though this may entail a loss of compression.

Phoneme based compression – advantages and disadvantages

Advantages	***Disadvantages***
Potentially high compression	Highly inconsistent compression rates
Maintains the sound of the word	Destroys the shape of the word
Comparatively low information loss	Some phonemes are uncompressible
	Damages flow of sentences
	High cognitive load on user
	Difficult to implement

Vowel removal

This method is particularly simple, but surprisingly effective. This system has already been discussed, and it has been mentioned that it may be wise to retain leading and trailing vowels. This sort of algorithm can be implemented in a single mechanical pass over the input string. We do not need to restrict the algorithm to removing vowels, other redundant characters such as duplicated consonants might be removed, and of course certain less important punctuation symbols (hyphens, apostrophes, etc. Certainly apostrophes, as these do not affect the flow of a sentence). We might even go so far as to remove all punctuation, including spaces, as the mail2sms program does. Each word would have its initial letter capitalized in order that the boundaries between words should be visible. Using the text in this document as an example, spaces make up about 16% of the total characters, so their removal could have a large effect. Spaces are extremely important in the structure of English, however, so it may be unwise to delete them. Consider the following sentences:

This is a particularly brief and simple example of the effect that removing spaces has on even simple, uncompressed sentences.

ThisIsAParticularlyBriefAndSimpleExampleOfTheEffectThatRemovingSpacesHasOnEvenSimpleUncompressedSentences.

The latter is noticeably more difficult to read. Spaces assist in the visual ‘chunking’ process, helping the eye to spot word boundaries. While practice may assist in the deciphering of sentences containing no spaces, it is not a nice solution.

If we disregard the removal of spaces, then, how good is the vowel removal solution?

If you take a close look at this sentence, you will find that it is free of vowels.

If yu tke a clse lk at ths sntnce, yu wll fnd tht it is fre of vwls.

Besides being false, the sentence has the following properties; the compression is fair at 17%, and the result is quite readable. Because the use of vowels in English words is quite evenly distributed (the proportion of vowels to consonants in most English writing is fairly stable), this method gives a consistent compression level, which is an improvement over the phoneme system.

A casual reader looking at the above examples, however, might ask why 'you' has been abbreviated to 'yu', and 'at' is unabbreviated. Both of these have better, more common and therefore possibly simpler abbreviations. Also, consider 'it is', which could easily be shortened to 'its' (apostrophe dropped for further reduction in size) while retaining all of its meaning.

Common abbreviations versus vowel removal

Word	***Common abbreviation***	***Abbreviation using simple vowel-removal system***
you	U	yu
at	@	at (unabbreviated)
and	&	and (unabbreviated)

All of the words in this table are common and have commonly used abbreviations, yet all are abbreviated differently (or not at all) by the vowel removal system. While the use of non-standard abbreviations is not a problem as long as they are understandable, there is obviously a better way to compress many common words.

Vowel removal – advantages and disadvantages

Advantages	***Disadvantages***
Very simple to implement	Imperfect output for some common words
Consistent compression rates	Some different words compress to identical strings
Moderate cognitive load on the user	Does not always retain word's sound
Retains much of original word's shape	Larger information loss than phoneme system.
Retains some sentence flow	

Hybrid method

Strictly speaking, this system is not a true hybrid of the other two, though the result is similar to that which might be expected from such a system.

It would be useful to combine the advantages of the vowel removal system with some sort of database of common abbreviations that could be used to heal some of vowel-removal's deficiencies. Many of the common abbreviations are in fact based on phoneme sounds, so this method could be considered a hybrid of the other two methods. In addition, it would have the ability to perform common abbreviations that were not phonemically based, such as the 'it is' mentioned in the previous section, and others such as 'can not' and 'he would'. While it is likely that SMS users would use these abbreviations themselves when typing in their message, making them automatic as well could assist the users. Longer abbreviations than two words, such as AFAIK (As Far As I Know) are expected to be performed by the user (though it may be possible to extend the algorithm to cope with these).

The method is very simple, being the same as the vowel removal system, except that the mechanical vowel removal stage is preceded by a stage that scans the input string for commonly abbreviated terms, then replaces them with their normal replacements. This is the method that was chosen for the final algorithm.

Hybrid System – Advantages and Disadvantages

Advantages	***Disadvantages***
Simple to implement	Does not always retain word sound
Consistent compression rates	Some different words compress to identical strings
Retains much of the word structure	
Moderate cognitive load on the user	

The ideal compression rate would be high, as in this example, already seen with the phoneme system:

'*See you for tea at eight*' → '*C U 4 T @ 8'*: Compression rate = 54.2%

Hoping to achieve this compression rate consistently is idealistic in the extreme: very few words compress to single character representations (though several common words do, which is useful). More likely would be something like:

'*Meet you tonight at eight for tea at the Café Swanky, be there, Ted*' → '*Mt U 2nite @ 8 4 T @ the Cfe Swnky, B thre, Td*': Compression rate = 31.3%

This is closer to the level of compression that would be achieved in general use (though it still plays to the algorithm's strengths somewhat, and as such may represent a higher level of compression than might be found in the general case). The actual compression rate achieved will depend entirely on the content of the original message, as some phrases will be more amenable to compression with this algorithm than others. Nevertheless, assuming a constant compression of at least 10% (which seems reasonable) then we can fit an extra 16 characters into an SMS message. With the average word length in English being between four and five letters, this might mean up to four more words. In practice, the algorithm generates on average 20 – 30% compression, or roughly 8-12 extra words per message.

Note that the word 'the' is left uncompressed in this example. 'The' is a very common word, so compressing this might lead to a noticeable increase in overall compression. It has been suggested that people skip over this word as a routine strategy when reading: 'the' has been shown to receive *"...reliably fewer fixations than one would expect, even fewer than other three-letter words (O'Regan, 1979)"*(Psychology of Reading, P. 13). However, another experiment *"discourages the view that people skip over THE as a routine* strategy" and that people are *"actually better at detecting misspellings in the word THE than in other words"* (both quotes from Psychology of Reading, P. 82). For this reason it was decided to leave the word 'the' uncompressed.

In the final algorithm, then, two types of compression are employed. The first is compression by a data dictionary of common abbreviations, for example:

See → C

You → U

At → @

Could have → Could've

…and a few other simple abbreviations, like

One → 1

Two → 2

The second compression is purely mechanical. Consonants form the most important part of the structure of words, with vowels usually only carrying the sound and enabling it to flow when pronounced. As previously discussed, the exception to this is when a vowel is in place as the initial or terminating letter of a word.

Pairs of identical consonants may also be unnecessary, where one will suffice, though this has not been implemented in an effort to retain more of the original structure of the word. Therefore, simply, the algorithm removes all vowels except those that commence or terminate a word, for example

Prs of idntcl cnsnnts my also be unncssry, whre one wll sffce, thgh ths hs nt bn implmntd in an effrt to rtn mre of the orgnl strctre of the wrd. Thrfre, smply, the algrthm rmvs all vwls excpt thse tht cmmnce or trmnte a wrd.

The above is a replication of the two sentences that precede it (and, incidentally, a 20% compression of the originals).

With both methods used on the above example, we get…

Prs of idntcl cnsnnts my also B unncssry, whre 1 wll sffce, thgh ths hs nt bn implmntd in an effrt 2 rtn mre of the orgnl strctre of the wrd. Thrfre, smply, the algrthm rmvs all vwls excpt thse tht cmmnce or trmnte a wrd.

This is 22% compression. Clearly, in this case, most of the compression takes place in the mechanical section. This result has gone from 283 characters to 221. Note also that in practice, most SMS users would not bother to put apostrophes in their messages (this is probably in part due to the tiresome text input method on most mobile telephones).

Implementation Issues

For the purposes of this project, speed was not a major issue to be considered in implementing the algorithm. It can be safely assumed that only the slowest, worst constructed programming language would take a noticeably long time to compress a message of around 160 characters using the algorithm.

Obviously, in a 'real-world' situation, efficiency and execution speed might become more important: for example if a server were processing many hundreds of messages at high speed. Even with the current minimal processing of messages, networks frequently have problems with SMS that can lead to slow delivery of messages, particularly at periods of high traffic such as the New Year. On the other hand, compression is more likely to be implemented at the handset level, where the user can see the results of compression before sending the message.

Perl seemed the most appropriate language for implementation of the algorithm. Perl (short for Practical Extraction and Report Language) is an interpreted language which, to quote Schwartz and Christiansen in 'Programming Perl', is

> "*...designed to assist the programmer with common tasks that are probably too heavy or portability-sensitive for the shell, and yet too weird or short-lived or complicated to code in C or some other UNIX glue language*."

The language has powerful constructs, particularly for string manipulation, that allow simple implementation of the algorithm. Most usefully, Perl contains a built in regular expression search and replace system. In addition, Perl is traditionally used in CGI scripting for the web, and it was always the intention to produce a web-page interface for the experiment to test the effectiveness of the algorithm. The 'hash' construct, an array of strings each of which is associated with another string, is particularly useful and is a basic construct of the language.

The code given here takes its input as the contents of a file, and puts its output on standard out, along with a step-by-step description of all the processing involved. While this is clearly not an ideal interface for actual text messaging, it sufficed for the purposes of this experiment, and is easily modified for other forms of I/O. Neither the length of the input message nor the length of the output is measured, so either or both could be beyond the SMS limits.

The first part of the code is a ‘hash’ called stab (short for STandard ABbreviations). This is used to implement the standard abbreviations function that is the first pass over the input. The input string is taken in a word at a time and compared with the hash table. If it matches, the word is replaced with its abbreviation and flagged so that it is not abbreviated further in the second pass. The flag used is the ASCII ‘bell’ symbol, represented by ‘\a’, though any non-printable ASCII character could be used. SMS supports accented characters, so it does not use the ASCII system, but Perl runs ASCII natively, and for English ASCII can produce most possible phrases (the most notable lack being the pound sterling symbol £). If the word cannot be abbreviated immediately, the next word is appended (with a space) and this is compared with the hash table to check for abbreviations. If it matches, it is flagged as before. If no standard abbreviations can be found, the program proceeds to the next stage of processing.

The stab table is held in the code to make it easier to read. Ideally, it would be stored in an external file where the user would be able to add personal entries (known abbreviations for their friends’ names, for example). This is simple to do with Perl.

The mechanical vowel removal is performed using Perl’s regular expression matching and substitution. First, the word is checked to see whether it is flagged. If so, the flag is removed, but processing passes on to the next word. This is so that standard abbreviations are not processed further. If the word is not flagged, it is checked to see whether it begins or ends in a vowel. If so, the variable $prefix or $suffix is set appropriately. After this stage, all vowels are removed, using the single Perl statement: s/[aeiou]+//g

The initial ‘s’ means that substitution is to take place. The token to be substituted is that between the first forward slash and the next, and it is to be replaced by the token between the second forward slash and the last. In this case, we replace any lowercase vowel or combination of lowercase vowels with nothing, effectively deleting them. The final ‘g’ indicates that this process is to be repeated throughout the word.

After being stripped of vowels, the word has its prefix and suffix appended back on to it.

Only lowercase vowels are removed. This allows a user to specify that a word is not to be compressed by leaving it in capitals. In general, capitalised words signify importance, so any possible misunderstanding that compression might cause can be relieved by capitalising the word. If this function is to be removed, the trailing ‘g’ in the substitute statement can be joined by an ‘i’, which commands the statement to ignore case.

The mechanical process makes a mistake in the case of lone vowels, where the first and last letters of the word are both vowels because there is only one letter. As this only occurs in two valid English words, ‘I’ and ‘a’, they have been included in the hash table at the start of the program so that they are flagged and not mechanically processed.

The complete code for the algorithm can be found in appendix 1.

Experiment

Once the algorithm had been implemented, it needed to be tested. Recalling the intentions of the algorithm:

- **Maximal compression**
- **Maximal speed of reading**
- **Maximal comprehension**

Compression is fixed as only one compression strategy is used. Compression rate is variable dependent on the exact input phrase, but averages around 25%, which is a fair rate.

Therefore, we needed to measure comprehension rates and deciphering speed. Because a large quantity of data was required, the difficulties of a directly administered experiment, such as an interview session, were too great. Instead, a questionnaire approach was chosen. To encourage participation in the questionnaire, it was decided to make it more interactive than a simple 'sheet-of-paper' system. The test content would be delivered via a web page, produced dynamically from a Perl script. This also allowed a timing system to be used, a feature more traditionally associated with directly administered experiments that is impossible with a mail-shot questionnaire.

On viewing the introductory page, each participant was allocated a unique anonymous identifier (a number). Results were only stored on completion of the test, when they were all sent to a text file named after the identifier. This was done to prevent half-complete results from entering the system, where they might produce confusion. Since every set of results would be complete, automated tools could be used to extract the data.

It should be noted that no dynamic compression was performed in the experiment. All phrases were pre-compressed and stored statically in a data file from which the experiment drew the questions.

The complete code for the script that produced the questionnaire can be found in appendix 2.

Measuring Comprehension of a Text Compression Algorithm for SMS

The questionnaire

The questionnaire introduced itself with a page of instructions informing the participant of the nature of the test, and that it was to be timed. It stressed, however, that accuracy was more important than time, so that the user should not rush.

Questions were multiple-choice, the correct answer being selected with the mouse and committed by clicking on 'Continue'. Following initial tests, it became apparent that users were having difficulty with the question style: the explanation provided at the start of the experiment did not prepare them sufficiently for the style of questions that they would encounter. One subject suggested that giving an example question and indicating its correct answer might be useful, and this was duly implemented.

Early test also indicated that users wanted to know how well they had performed. One subject suggested that a rating system would provide a good incentive to pass on the URL of the experiment page to the others in order to compete. A rating system was implemented, and may indeed have positively affected the quantity of results received, as the page got far more 'hits' than expected.

On moving on from the introductory page, a few simple yes/no questions were asked of the subjects:

1. **Do you own a mobile telephone?**

2. **Do you regularly use SMS (text) messages?**

3. **Would you be interested in using an automated compression algorithm for your text messages?**

This data is used to determine the subject's familiarity with text messaging, to see if that was a factor in their ability to read the compressed messages.

The age group of the user was then determined from the following discrete ranges: 0 – 17, 18 – 25, 26 – 35 and 35+. This was done for two reasons:

- To determine roughly the proportion of students performing the test. Subjects were pointed in the direction of the experiment via emails or via a message alerting them to its presence on a newsgroup. Consequently, it was expected that most of the people performing the experiment would be students at York. In order to attract further subjects, a rating system gave the user their 'score' (percentage of questions answered correctly) at the end of the test, and a message suggested that they challenge their friends to beat their score. While a box asking the subject to select their occupation would perform this task more efficiently, it brings in increased complexity to the analysis, and the purpose of this experiment was to measure comprehension, rather than to survey SMS users in great detail.

- To determine what age group most commonly uses SMS. There is certainly anecdotal evidence that, as one might expect, younger people were the first to take up the new technology. According to the Guardian newspaper, SMS text messaging… "*is mostly used by tech-savvy types aged 24 or less who are attracted by the low cost and the fact that what they have to say does not necessarily require a full telephone conversation"* (John Cassy, Guardian, Thursday January 4th 2001), but it would be interesting to see whether other age groups had also embraced text messaging. In addition, one might expect younger people to be more interested in a new compression technology.

The experiment then proceeds to a series of simple comprehension questions. These are of the form:

***Phrase**: I just want to thank you for the gifts. They were fantastic, and I love the teddy!*

***Compressed as**: I jst wnt 2 thnk U 4 the gfts. Thy wre fntstc, & I lve the tddy!*

- *I went to think about gifts for you. I saw a fantastic teddy.*
- *Thank you for the gifts, they were great, especially the teddy.*
- *I went to think about gifts for you and Todd.*
- *Thank you for the futuristic gifts, I love the tidy.*

The subject sees either the compressed sentence or the uncompressed version (the uncompressed data is used as control data). They must then select which of the following four options most accurately paraphrases the meaning of the first sentence. The answer they gave is recorded, and a Javascript script is used to record the time it took them to select that answer (strictly speaking, the time recorded is the time from the page appearing until the 'submit' button is pressed, but this should give a result that is very similar to the comprehension time). Tests were not performed under controlled conditions, which may cause errors in the results though the subjects were warned that time was a factor.

Four options were provided to remove the psychological tendency that flustered experiment subjects have to select the middle option in an odd-numbered set, selecting, for example, option two of three or three of five.

There were sixteen of these questions in all. This number was a compromise between the quantity of data collected and user frustration at the length of time required to perform the test. It took users an average of about six minutes to perform the test with this number of questions.

Every fifth person to take the test was given a control version of the test. In this version, the same question set was used, but the statements displayed to the user were the uncompressed versions of the test statements. This was done to determine what comprehension level is achieved without compression, and therefore the relative change in comprehension when a message is compressed using the algorithm. Although one might expect 100% comprehension when reading 'normal' messages, the control group can help determine the background level of inaccuracy in the test subjects.

The decision to limit the control group to a far smaller group than the test group was made because it was feared that the experiment would be visited by very few participants. The most interesting results would be those derived from the subjects taking the compressed version of the test, so the test was set up so that more of these results would be collected. Ideally, perhaps, the two groups would be of equal size (though this has no effect on the viability of the statistical analysis technique that was used).

Participants were slightly more likely to give up if they found that they were performing the control version of the test. In the case where the participant gave up, no data was stored.

Question Design

Since comprehension questions such as these are typically used in examining the comprehension of foreign languages, it seemed that there might be some standard question sets or styles of questioning that could be used.

A psycholinguistics expert, Dr G. Altman of the Psychology Department, University of York, was contacted for advice. He suggested that, though question sets for testing the comprehension levels of aphasics do exist, it was better to make up new questions for this test than to use an existing set, so this is what was done.

The question statements are unrelated to each other, being random quotes from a variety of books and periodicals, snatches of song lyrics, or imaginary scenarios. Questions were deliberately not tailored to a 'typical SMS style', such as messages planning meetings (though one of the examples is just that). SMS is potentially extremely general in its use, so the phrases chosen were as different as possible. Some of the questions are specifically targeted at the algorithm's weaknesses, specifically the following:

Phrase: Welcome to the mix and match world of composites.

Compressed as: Wlcme 2 the mx & mtch wrld of cmpsts.

- Welcome to the mix and match world of composts.
- Welcome to the mix and match world of campsites.
- Welcome to the mix and match world of composites.
- Welcome to the mix and match wielder of composites

The compressed phrase would result if any of the first three statements were compressed, so they are all correct, although the third answer is the actual statement that was compressed and this answer was used as the marker for judging accuracy in the rating system.

The correct answer for the question might be a perfect copy of the master statement, or a paraphrasing. Both styles were used, though most questions used the former.

The complete list of questions can be found in appendix 3.

Measuring Comprehension of a Text Compression Algorithm for SMS

Results

Eighty people performed the test in total, with fifteen of these performing the control test. While one might expect exactly 20% of the results to be control results, the experiment gave the users the option of giving up before the results were stored, so there is one fewer set of control results than might be expected.

Age distribution across the two sets was heavily weighted toward the 18 – 25 age group, which meant that no correlation test on age against speed or accuracy could be performed.

As might be expected, the control group were more accurate. Mean accuracy for the group reading compressed messages was 87.3%, while the control group achieved a mean of 94.8% accuracy, a difference of 7.5%.

Mean time for the compressed group was 21565ms per question. For the control group this was 15761 ms per question, a difference of 5804 ms, or nearly six seconds per question.

Besides performing faster on average, the control group were faster on every question, though to varying degrees.

To check for statistical significance to the results, the t-test was used.

$$t = \frac{M_1 - M_2}{\sqrt{\frac{\left(\sum x_1^{\,2} - \frac{\left(\sum x_1\right)^2}{n_1}\right) + \left(\sum x_2^{\,2} - \frac{\left(\sum x_2\right)^2}{n_2}\right)}{(n_1 - 1) + (n_2 - 1)} \left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

Where:

M_1 = Mean for the first group

M_2 = Mean for the second group

x_1 = The results for the first group

x_2 = The results for the second group

n_1 = Number of subjects in first group

n_2 = Number of subjects in second group

Measuring Comprehension of a Text Compression Algorithm for SMS

Statistics for the Control Group

Total subjects: 15

Age distribution for control group:

Age group	*Number of subjects*
0 – 17	0
18 – 25	13
26 – 35	1
35+	1

Times:

Question number	*Mean time (ms)*
1	14709
2	11469
3	25224
4	9992
5	19388
6	14555
7	17884
8	19732
9	14713
10	26035
11	19067
12	16748
13	9758
14	16673
15	18825
16	13704

Total time = 267936 ms
Mean time for control group = 16746.0 ms

Sum of squares of times = 2756472788.8

Percentage accuracy:
Total percentage correct = 1422
Mean accuracy = 94.8%
Sum of percentage squares = 135084

Measuring Comprehension of a Text Compression Algorithm for SMS

SMS users within the control group
SMS users: 12
Interested in a compression algorithm: 5
Percentage interested in a compression algorithm: 42

Times:

Question number	*Mean time (ms)*
1	14687
2	11100
3	24472
4	9935
5	19287
6	13862
7	17471
8	19666
9	13799
10	27210
11	18944
12	17288
13	9734
14	16718
15	17993
16	13138

Total time = 265304 ms
Mean time = 16581.5 ms

Sum of squares = 4751751607.6

Percentage accuracy:
Total percentage for SMS users = 1140
Mean accuracy = 95%
Sum of percentage squares = 108504

Measuring Comprehension of a Text Compression Algorithm for SMS

Statistics for the Test Group

Total subjects: 65

Age distribution for test group:

Age group	*Number of subjects*
0 – 17	1
18 – 25	60
26 – 35	3
35+	1

Times:

Question number	*Mean time (ms)*
1	18915
2	26143
3	27777
4	10028
5	23399
6	18190
7	24211
8	25157
9	20918
10	30551
11	26023
12	25310
13	20310
14	30399
15	25108
16	14159

Total time = 366598 ms
Mean time = 22912.4 ms

Sum of squares of times = 5156913428.5

Percentage Correct:
Total of percentages = 5672
Mean accuracy = 87.3%
Sum of percentage squares = 508352

Measuring Comprehension of a Text Compression Algorithm for SMS

SMS users within the test group:
SMS users: 46
Interested in a compression algorithm: 29
Percentage interested in a compression algorithm: 63

Times:

Question number	*Mean time (ms)*
1	20031
2	26852
3	28883
4	10703
5	23903
6	19858
7	26741
8	27437
9	22372
10	32589
11	26451
12	26495
13	19682
14	3792
15	25735
16	14789

Total time = 386311 ms
Mean time = 24144.5 ms

Sum of squares = 9879060152.1

Percentage accuracy:
Total of percentages for SMS users = 4007
Mean accuracy = 87.1%
Sum of percentage squares = 353285

Non-SMS users within the test group:
Non-SMS users: 18

Question number	*Mean time (ms)*
1	17081
2	25784
3	26493
4	8861
5	23411
6	14937
7	19089
8	20727
9	18325
10	27004
11	26308
12	23652
13	22924
14	23368
15	24896
16	13334

Total time = 336192 ms
Mean time = 21012.0 ms

Sum of squares = 7487846085.8

Percentage accuracy:
Total of percentages = 1665
Mean accuracy = 92.5%
Sum of percentage squares = 155067

Accuracy

All subjects

When the t-test was used for the 'percentage correct' statistic, the t value came out as –1.9874. The sign of the t value is ignored.

This means that there is a statistically significant reduction in accuracy when reading messages compressed with the algorithm, with the significance at the 5% level.

SMS Users

Within the test group, SMS users were compared against non-SMS users. The t value produced is 0.9341. This means that there is no statistically significant difference between the accuracy of the two groups.

Time

All subjects

When the t-test was used for the timing statistics, the t value came out as 5.840.

This means that there is a statistically significant difference in time between the control group and the test group. Compressing messages significantly increases the time required to comprehend them, with the significance at the 0.05% level.

SMS Users

Within the test group, SMS users were compared against non-SMS users. The t value produced is 0.485808. This means that there is no statistically significant difference between the accuracy of the two groups.

Problems

There were a number of minor problems with the experiment that may have affected the fairness or validity of the results.

1. As mentioned, the timing system was slightly inaccurate. The script that measured the time taken started as soon as the page loaded. In some situations, this could be before the subject was able to see the question, which will have skewed the timing system. The pages were very simple plain-text HTML in order to reduce downloading times, and most of the hits were probably from local machines (within the york.ac.uk domain), which should minimise errors due to this potential problem. A more accurate system might have been to use a Java program that waited until it got all the data before showing the question, simultaneously starting the timer.

2. In addition, the timing only stopped when the user clicked the 'submit' button on each page, not precisely when the user has their answer. This would appear to be reasonable however, as it signifies when the user has committed to an answer, and is probably the only practicable method for terminating timing.

3. Most of the subjects for the test were science students from York University, which is hardly the most representative slice of the population. The majority were probably Computer Science and Biochemistry students in the 18 – 25 age group. Most were SMS users. Future experiments would use more randomised test groups.

4. Conditions for the experiment were not tightly controlled. Though users were given a description of the form of the experiment, it is not clear how many read this. In addition, it was possible to cheat at the test by either simply using the browser's 'Back' command, or even through direct manipulation of the data, which was appended to the URL of the page as the user progressed. Ideally, the 'Post' method for forms would have been used so that the user would not be able to see the results in the URL and could not manipulate them, but this failed to work with the Perl code for the experiment.

Conclusion

Though the algorithm created produced a consistent compression of around 25%, there was a trade-in for accuracy and speed of comprehension of the generated messages.

Comprehension speed took the most significant loss, with an extremely strong positive correlation in speed reduction when reading compressed messages as compared using the t-test. The speed at which messages are read is rarely a factor in SMS, but a low speed is indicative of high cognitive load which makes the reading process more irritating.

The results were somewhat less strong for accuracy, but still highly significant. Accuracy levels for comprehending compressed messages were quite high compared to accuracy comprehending the same messages in an uncompressed form, but the difference was relevant.

It appears that existing SMS users are no better or worse able to decipher messages compressed with this algorithm than people who do not frequently use SMS messaging are. This shows that deciphering of the algorithm does not benefit from prior use of short text messages. On the other hand, since many of the questions were of an unusual style (unlikely to actually be sent as text messages in common use) these results may require further investigation.

While the algorithm generates output that is significantly less easy to comprehend than uncompressed text, this does not render it useless. It may still be easy enough to comprehend for general use, though it should never be used where correct understanding of the fine details of a message is critical.

Further work

The user should be able to modify their own data dictionary to include further abbreviations that they know (such as names). This would further increase compression rates on common messages for that user.

Give the algorithm the ability to perform abbreviations of strings with more than two words (several words, such as 'As Far As I Know → AFAIK'). These are less likely to be known by inexperienced texters, but usually the abbreviation is reversible, so that the abbreviated string is sent, but it could be automatically unabbreviated when the user receives it in order that full comprehension is achieved.

A better hybridisation method between the phoneme and vowel removal systems may exist. Certain substrings in English are always pronounced in the same way, and these could be automatically replaced, rather than the abbreviation being performed at the whole word level as it is in the current algorithm. For example:

ate → 8

zy →Z

ew → U

In a sentence such as 'My view of the crates was hazy' both of these could be replaced:

My vU of the cr8s was haZ.

If this is then combined with the standard abbreviations (not used in this example) and vowel removal we get:

My viU of the cr8s ws haZ.

Further experimentation with a more randomly chosen group of subjects to see if the same trends occur across, for example, different age groups, would be interesting.

If the experiment were to be performed again, a better timing system would be useful, perhaps with a Java applet that allowed the user to view each question when ready. It might also be useful to more accurately control conditions. It would be interesting to simulate a character-limited display so that the subject must scroll to see the entire message, as they would have to with a telephone.

Measuring Comprehension of a Text Compression Algorithm for SMS

Acknowledgements

Mr. A Ormerod for assistance with Perl programming for the web.
Dr. Gerry Altman in Psychology for answering questions about comprehension questions.
Dr. Alistair Edwards in Computer Science for his suggestions.

References

Web Links:

Email2SMS homepage
www.new.ox.ac.uk/~adam/computing/email2sms/

Lingua::EN::Squeeze page
www.new.ox.ac.uk/~adam/computing/email2sms/Lingua_EN_Squeeze.html

Microsoft Intellishrink
http://www.microsoft.com/office/outlook/mobile/default.htm

AmikaNow! Highlights press release
www.amikanow.com/newsroom/NewsPostings/wowfi/WOW!Wireless.htm

AmikaNow! Highlights example
https://www.amikafreedom.com/afc1/amikafreedom.com/example.htm

Reading and Comprehension Processes:
The Psychology of Reading – An Introduction, 2nd Edition. Crowder and Wagner, Oxford University Press, 1992.

Comprehension Processes in Reading. Balota, Flores d'Arcais, Rayner, Lawrence Erlbaum Associates Inc, 1990.

Experiment Design:
Real World Research. Robson, Blackwell Publishers,1993.

Statistics:
Learning to Use Statistical Test in Psychology, Greene and D'Oliveira, Open University Press, 1982

Measuring Comprehension of a Text Compression Algorithm for SMS

Measuring Comprehension of a Text Compression Algorithm for SMS

Appendices

Measuring Comprehension of a Text Compression Algorithm for SMS

Appendix 1: Compression Algorithm code

```
#! /usr/bin/perl -w

#Variable names are highlighted in BOLD font. Comments (like this) are
in #italic bold

# This is a limited version of an abbreviation table.
# There are probably numerous obvious omissions from this table
#Note that all replacements are flagged with the bell character ('\a')

%stab = (
        "you", "U\a",
        "are", "R\a",
        "to", "2\a",
        "see", "C\a",
        "be", "B\a",
        "tea", "T\a",
        "a", "a\a",       #Required because of the regular expressions
        "I", "I\a",
        "at", "@\a",
        "for", "4\a",
        "any", "NE\a",
        "and", "&\a",
        "at", "@\a",
        "tonight", "2nite\a",
        "today", "2day\a",
        "one", "1\a",
        "two", "2\a",
        "three", "3\a",
        "four", "4\a",
        "five", "5\a",
        "six", "6\a",
        "seven", "7\a",
        "eight", "8\a",
        "nine", "9\a",
        "ten", "10\a",
        "Monday", "Mon.\a",
        "Tuesday", "Tues.\a",
        "Wednesday", "Wed.\a",
        "Thursday", "Thurs.\a",
        "Friday", "Fri.\a",
        "Saturday", "Sat.\a",
        "Sunday", "Sun.\a",
        "could have", "couldve\a",
        "would have", "wouldve\a"
);
```

Measuring Comprehension of a Text Compression Algorithm for SMS

```
#The following section handles input from a file, including splitting it into
#chunks of one word in size. Newlines in the input file are not tolerated.
print ("Please specify the file you wish to shrink: ");
$file = <STDIN>;
chomp $file;

        open (INPUT, "$file") || die "Can't open file!!: $!";
        print ("File opened...\n");

        @file = <INPUT>;

        close (INPUT) || die "Unable to close file!!: $!";
        print ("File closed...\n");

        $initial_size = length($file[0]);

        @split = split (/ /, "@file");
        $size = @split;

#The replacement of standard abbreviations is handled here
        $iterator = 0;
        while ($iterator < $size) {
                if(defined($stab{$split[$iterator]}) ) {
                        print ("$stab{$split[$iterator]}" . " ");
                        $intermediate[$iterator] =
("$stab{$split[$iterator]}");
                }
                else {
                        $pair = ("$split[$iterator]" . " " ."$split[($iterator +
        1)]");
                        if (defined($stab{$pair}) ) {
                                print ("$stab{$pair} ");
                                $intermediate[$iterator] = ("$stab{$pair}");
                                $iterator++;
                                $intermediate[$iterator] = ("INVALID\t");
#The space previously filled by the second word in a pair is replaced by the
#string 'INVALID\t'. This is later stripped out.
                        }
                        else {
                                print("$split[$iterator]" . " ");
                                $intermediate[$iterator] =
("$split[$iterator]");
                        }
                }
                $iterator++;
        }
```

```
#Mechanical processing using regular expressions begins
        $iterator = 0;
        $word = $intermediate[0];
        $final_string = ("");
        $size = @intermediate;


        while ($iterator < $size) {
                $word = $intermediate[$iterator];
                if ($word =~m/\t/) {
#Ignore completely words tagged with /t
                        $iterator++;
                }
                elsif ($word =~ m/\a/) {
#Ignore word if flagged as already abbreviated
                        $word =~ s/\a//;        #Remove the flag
                        print ("$word ");
                        $final_string = ("$final_string" . "$word ");
                        $iterator++;
                }
                else {
                        $prefix = '';

                        #If word begins with vowel
                        if ($word =~ m/(^[aeiou])/) {
                                $prefix = $1;
                        }
                        $suffix = '';
                        #If word ends with vowel
                        if ($word =~ m/([aeiou]$)/) {
                        $suffix = $1;
                }

                        $punctuation = '';
                        #If word ends in punctuation
                        if ($word =~ m/([\!\?\.\:\-]$)/) {
                        $punctuation = $1;
                }
                $word =~ s/[aeiou\'\!\?\.\:\-]+//g;       #Remove ALL vowels

                print ("$prefix"."$word"."$suffix" . "$punctuation ");
                $final_string = ("$final_string".
                "$prefix"."$word"."$suffix" . "$punctuation ");
                $iterator++;
                }
        }
```

```
        $final_string =~ s/( $)//;              #Remove trailing spaces
        $final_size = length($final_string);
        $compression_rate = (((($initial_size - $final_size) / $initial_size )
* 100 );
#Output some useful data
        print ("Original size was $initial_size characters.\n");
        print ("Final size is $final_size.\n");
        print ("Compression rate is $compression_rate %\n");
```

Measuring Comprehension of a Text Compression Algorithm for SMS

Appendix 2 : Experiment code

```
#!/york/bin/gnu/perl

%FORM=&read_input;

if (!defined($FORM{"page_number"}))       #Introductory page
        {
                #Get a unique user number
                open(FH, "/usr/as152/web/cgi-bin/user_counter")      or die
"can't open user_counter: $!";
                &lock(FH);
                $user_number = <FH> || 0;
                close FH;

                $user_number = $user_number+1;

                open(FH, ">/usr/as152/web/cgi-bin/user_counter")      or die
"can't open user_counter: $!";
                (print FH $user_number, "\n")   or die "can't write
user_counter: $!";
                &unlock(FH);
                close FH;

                print "Content-type: text/html\n\n";
                print "
<HTML>
<TITLE>SMS Experiment</TITLE>
<CENTER>
<H1>Welcome.</H1><BR><H2>Please read the following carefully:</H2>
</CENTER>
<HR>
<P>First, thank you for giving up a few moments of your time to participate
in
this experiment. I have devised an algorithm that compresses message text,
hopefully while retaining readability
<BR>The structure is as follows: First you will be shown an introductory
page asking you about 'You and SMS'. Then, you will be shown a series of
statements and asked to select which of the statements below it most
accurately describes the meaning of the first statement. Note that the 'word-
for word' meaning is not necessarily going to be one of the options: a
paraphrasing may be closer to the correct result.
<BR>For most people the first message will be compressed, but every fifth
user will be performing a control experiment, in which the first message will
not be compressed. So don't worry if it looks far too easy, as I need control
results for this experiment.
```

```
<BR><B>Response time is a factor</B>, but accuracy is more important, so please don't rush. At the end you will be rated on your performance.</P>
<HR><P>Here is an example of the style of question you will encounter:
        <P>Given the following statement:</P>
        <H2>Ths is an exmple tst stmnt</H2>
        <BR>Which of the following statements do you feel best describes the statement above?<BR>
        <FORM>
        <INPUT TYPE=radio NAME=test VALUE=1>This is an exemplary toasted stamina.<BR>
        <INPUT TYPE=radio NAME=test VALUE=2>This test statement is an example.<BR>
        <INPUT TYPE=radio NAME=test VALUE=3>This toast is exemplary.<BR>
        <INPUT TYPE=radio NAME=test VALUE=4>This shall test your stamina.<BR>
        </FORM>
        <BR>In this case the correct answer is the second answer.
        <BR>
        When you are ready, click 'Continue' below to start the experiment.
<FORM METHOD=GET ACTION=cgiexperiment2.pl>
<INPUT TYPE=hidden NAME=page_number VALUE=1>
<INPUT TYPE=hidden NAME=user_number VALUE=$user_number>
<INPUT TYPE=submit VALUE=Continue>
</FORM>
</HTML>
                ";
        }
```

```
elsif ($FORM{"page_number"} == 1)          #If first page
        {
        $user_number = $FORM{"user_number"};
      print "Content-type: text/html\n\n";

      print "
<HTML>
<TITLE>You and SMS</TITLE>
<BODY>
This is page number 1 of 18.
<P>Please answer the questions below before moving on to the next part of
the experiment.</P>
<FORM METHOD=get ACTION=cgiexperiment2.pl>
Please select your age group:
<SELECT NAME=age_group>
<OPTION VALUE=1>0-17
<OPTION VALUE=2 SELECTED>18-25
<OPTION VALUE=3>26-35
<OPTION VALUE=4>35+
</SELECT>
<BR>
<BR>Please tick all the options which apply to you:<BR>
<INPUT TYPE=hidden NAME=page_number VALUE=2>
<INPUT TYPE=hidden NAME=user_number VALUE=$user_number>
<INPUT TYPE=checkbox NAME=has_phone VALUE=yes>I own a mobile
telephone.<BR>
<INPUT TYPE=checkbox NAME=uses_sms VALUE=yes>I use SMS (text
messaging) services regularly.<BR>
<INPUT TYPE=checkbox NAME=interested VALUE=yes>I would be
interested in an automatic message compression system.<BR>
<INPUT TYPE=submit VALUE=Continue>
</FORM>
</BODY>
</HTML>
      ";
        }
```

```
elsif ($FORM{"page_number"} < 18)          #This number must be 2 above
the number of questions
   {
        open (WORDSLIST, "/usr/as152/web/cgi-bin/compressed") || die
"Can't open wordslist!!!: $!";
        @FILE_CONTENTS = <WORDSLIST>;
        close WORDSLIST;

        #File contents are accessed via the following array syntax:
        #print $FILE_CONTENTS[1];

        #Unique user identifier from the initial page
        $user_number = $FORM{"user_number"};

        $page_number = ($FORM{"page_number"} + 1);
        $actual_page = ($page_number - 1);

        #Calculating where in the file to look for the statements

        if ($user_number % 5)
                {
                #UnCompressed statement
                $statement = ((($actual_page -2) * 6) + 1);
                $option1 = ($statement + 1);
                }
        else {
                #Compressed statement
                $statement = ((($actual_page -2) * 6));
                $option1 = ($statement + 2);
        }
        $option2 = ($option1 + 1);
        $option3 = ($option2 + 1);
        $option4 = ($option3 + 1);


        #Hidden values from the 'questionnaire' page
        $age_group = $FORM{"age_group"};
        $has_phone =  $FORM{"has_phone"};
        $uses_sms = $FORM{"uses_sms"};
        $interested = $FORM{"interested"};
```

```
print "Content-type: text/html\n\n";
print "
<HTML>

<SCRIPT LANGUAGE=javascript>
                timeA = new Date();

                function get_time_taken()
                {
                timeB = new Date();
                timeDifference = timeB - timeA;
                document.forms[0].question$actual_page\_time_taken.value
= timeDifference;
                }

                function start_timer()
                {
                timeA = new Date();
                }
</SCRIPT>

<TITLE>SMS Experiment: Questions.</TITLE>
<BODY onload='start_timer()' NAME=the_form>
 This is page $actual_page of 18
<P>Given the following statement:</P>
<H2>$FILE_CONTENTS[$statement]</H2>
<BR>Which of the following statements do you feel best describes the
statement above?<BR>
<FORM onSubmit='get_time_taken()' ACTION=cgiexperiment2.pl
METHOD=put>
<INPUT TYPE=hidden NAME=page_number VALUE=$page_number>

<INPUT TYPE=hidden NAME=user_number VALUE=$user_number>

<INPUT TYPE=hidden NAME=age_group VALUE=$age_group>
<INPUT TYPE=hidden NAME=has_phone VALUE=$has_phone>
<INPUT TYPE=hidden NAME=uses_sms VALUE=$uses_sms>
<INPUT TYPE=hidden NAME=interested VALUE=$interested>

<INPUT TYPE=radio NAME=question$actual_page
VALUE=1>$FILE_CONTENTS[$option1]<BR>

<INPUT TYPE=radio NAME=question$actual_page
VALUE=2>$FILE_CONTENTS[$option2]<BR>

<INPUT TYPE=radio NAME=question$actual_page
VALUE=3>$FILE_CONTENTS[$option3]<BR>
```

```
<INPUT TYPE=radio NAME=question$actual_page
VALUE=4>$FILE_CONTENTS[$option4]<BR>

<INPUT TYPE=hidden NAME=question$actual_page\_time_taken>

        ";

#All previous question answers are stored as hidden variables in the
HTML
        foreach $key (sort keys %FORM) {
                if (substr($key, 0, 8) eq "question"){
                        print "<INPUT TYPE=hidden NAME=$key
VALUE=$FORM{$key}>\n";
                }
        }

                print "

<INPUT TYPE=submit VALUE=Next>
</FORM>
</BODY>
</HTML>
      ";

   }

#Last page displayed
else {
        #Calculate the user's accuracy
        $correct = 0;
        if ($FORM{"question2"} == 2)
                {
                $correct++;
                }
        if ($FORM{"question3"} == 2)
                {
                $correct++;
                }
        if ($FORM{"question4"} == 1)
                {
                $correct++;
                }
        if ($FORM{"question5"} == 2)
                {
                $correct++;
                }
```

```
if ($FORM{"question6"} == 3)
        {
        $correct++;
        }
if ($FORM{"question7"} == 3)
        {
        $correct++;
        }
if ($FORM{"question8"} == 4)
        {
        $correct++;
        }
if ($FORM{"question9"} == 2)
        {
        $correct++;
        }
if ($FORM{"question10"} == 1)
        {
        $correct++;
        }
if ($FORM{"question11"} == 4)
        {
        $correct++;
        }
if ($FORM{"question12"} == 2)
        {
        $correct++;
        }
if ($FORM{"question13"} == 4)
        {
        $correct++;
        }
if ($FORM{"question14"} == 3)
        {
        $correct++;
        }
if ($FORM{"question15"} == 4)
        {
        $correct++;
        }
if ($FORM{"question16"} == 1)
        {
        $correct++;
        }
if ($FORM{"question17"} == 2)
        {
        $correct++;
```

```
                }

        $percent_correct = (($correct / 16) * 100);
        $percentage = sprintf "%.0f" ,  $percent_correct;


        #Write all the results to a file named after the user_ID
        $user_number = $FORM{"user_number"};
        open (OUTPUT, ">/usr/as152/web/cgi-bin/$user_number") || die
"Can't open output file $user_number! $!";
        foreach $key (sort keys %FORM) {
                print OUTPUT "<INPUT TYPE=hidden NAME=$key
                VALUE=$FORM{$key}>\n";
        }
        print OUTPUT "Percent correct overall: $percentage\n";
        close (OUTPUT) || die "Can't close output file $user_number! $!";

        print "Content-type: text/html\n\n";
        print "
<HTML>
<TITLE>Thank you for performing the experiment.</TITLE>
<BODY>
<CENTER><H1>Thank you.</H1></CENTER>
<HR>
<P>You got $correct of 16 questions right, which means you were
$percentage% accurate.</P>
Today's rating system: Name of rating system<P>Your rating:<H2>";
        if ($percent_correct <= 10)
                {
                print "HTML assessing extremely  poor performance";
                }
        elsif ($percent_correct <= 25)
                {
                print "HTML assessing poor performance ";
                }
        elsif ($percent_correct <= 50)
                {
                print "HTML assessing fairly poor performance";
                }
        elsif ($percent_correct <= 75)
                {
                print "HTML assessing 'not bad'";
                }
        elsif ($percent_correct <= 90)
                {
                print "HTML assessing 'very good'";
                }
```

```
        elsif ($percent_correct < 100)
                {
                print "HTML assessing 'extremely good'";
                }
        elsif ($percent_correct == 100)
                {
                print "HTML assessing 'perfect'";
                }
        print "</H2></P>
<P>Your results have now been stored. Your assistance is most
appreciated.<BR>
<B><I>Please do not attempt the experiment again, as this is likely to harm
my results.</B></I>
<BR>Instead why not encourage your friends to have a go, and compare
ratings?</P>
<HR>
Comments? Then <A HREF=mailto:as152\@york.ac.uk>mail me</A>!
</BODY>
</HTML>
        ";
        exit 0;
}
exit 1;

sub read_input
{
   local ($buffer, @pairs, $pair, $name, $value, %FORM);
   # Read in text
   #$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;
   #if ($ENV{'REQUEST_METHOD'} eq "POST")
   #{
   #    read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
   #} else
   #{
      $buffer = $ENV{'QUERY_STRING'};
   #}
   # Split information into name/value pairs
   @pairs = split(/&/, $buffer);
   foreach $pair (@pairs)
   {
      ($name, $value) = split(/=/, $pair);
      $value =~ tr/+/ /;
      $value =~ s/%(..)/pack("C", hex($1))/eg;
      $FORM{$name} = $value;
   }
   %FORM;
}
```

```
sub lock {
$LOCK_EX=2;
  local($file)=@_;
 flock($file, $LOCK_EX);
}


sub unlock {
$LOCK_UN=8;
  local($file)=@_;
 flock($file, $LOCK_UN);
}
```

Measuring Comprehension of a Text Compression Algorithm for SMS

Appendix 2 : Questions used in the experiment

Phrase: I just want to thank you for the gifts. They were fantastic, and I love the teddy!
Compressed as: I jst wnt 2 thnk U 4 the gfts. Thy wre fntstc, & I lve the tddy!
Compression rate: 22%

- I went to think about gifts for you. I saw a fantastic teddy.
- Thank you for the gifts, they were great, especially the teddy.
- I went to think about gifts for you and Todd.
- Thank you for the futuristic gifts, I love the tidy.

Note: This one is particularly hard, with some nasty high vowel-consonant ratio words (and bad grammar).
Phrase: The pipes leaked real bad again last night. The kitchen is soaked.
Compressed as: The pps lkd rl bd agn lst nght. The ktchn is skd.
Compression rate: 26%

- The Pope's liked roll-back akin to the lost nought. The kitchen is skewed.
- Last night the pipes leaked again, soaking the kitchen.
- Last night the pips looked terrible, and now the kitchen is skewed.
- The pipes looked very bad last night, and now the kitchen is soaked.

Phrase: Have you seen the line-up for the game on Saturday? There's some crazy team planning going on there.
Compressed as: Hve U sn the lne-up 4 the gme on Sat.? Thrs sme crzy tm plnnng gng on thre.
Compression rate: 25%

- Have you seen the line-up for Saturday's game? It's madness!
- It's a sin, all the grime on the lanes. The Crazy Planning gang are going to win.
- The line-up for the game on Saturday is a sin. The team set-up is bonkers.
- Saturday's team set-up is grim. The management gang are mad.

Phrase: Keep your friends close, but your enemies closer.
Compressed as: Kp yr frnds clse, bt yr enms clsr.
Compression rate: 30%

- Keep your friends clues, but your enemas closer.
- Keep your friends close, but your enemies closer.
- Kip your fronds clues, but your enemas clearer.
- Keep your friends close, but your enemies clearer.

Measuring Comprehension of a Text Compression Algorithm for SMS

Phrase: Just seen Crouching Tiger: it's great, watch it! The fight scenes are incredible, and the story is a real epic.
Compressed as: Jst sn Crchng Tgr: its grt, wtch it! The fght scns R incrdble, & the stry is a rl epc.
Compression rate: 23%

- Watch Crouching Tiger: it has great fight scenes and an epic story.
- I've just seen Crouching Tiger: the fights scan incredibly.
- Watch Crouching Tiger, the story is a rolling epic.

Phrase: Remind me, is it safe to acidify potassium cyanide? Reply quickly, I'm about to try it.
Compressed as: Rmnd me, is it sfe 2 acdfy ptssm cynde? Rply qckly, Im abt 2 try it.
Compression rate: 22%

- Can I acidify potassium cyanide?
- Remind me how to make possum candy.
- I'm about to acidify potassium cyanide. Is this safe?
- How do you apply potassium candy?

Phrase: Greg Bueller has just been rushed to hospital! I think his appendix has exploded. Come to the City General right away!
Compressed as: Grg Bllr hs jst bn rshd 2 hsptl! I thnk hs appndx hs expldd. Cme 2 the Cty Gnrl rght awy!
Compression rate: 25%

- Greg Bueller has been taken to the hostel, where his appendix was expelled. Come to the City General right away.
- Greg Bueller's appendix has exploded. Come to the City General hospital right away.
- Greg Bueller has rushed to the hostel. He was apparently expelled. Come to the City General before it all goes awry.
- Greg Bueller has been rushed to the City General. Come before it all goes awry.

Phrase: Not sure I can get there at seven. I can see you by the bridge at eight, though.
Compressed as: Nt sre I cn gt thre @ 7. I cn C U by the brdge @ 8, thgh.
Compression rate: 29%

- Not sure I can get three by 7 pm. I can meet for bridge at 8.
- Can't get there for 7, but I can meet you by the bridge at 8.
- Can't get three by 7, but I can meet you by the bridge at 8.
- Can't get there for 7, but I can meet you for bridge at 8.

Measuring Comprehension of a Text Compression Algorithm for SMS

Phrase: You are a Scorpio, right? Your horoscope is predicting some BAD stuff this week, I'm afraid.
Compressed as: U R a Scrpo, rght? Yr hrscpe is prdctng sme BAD stff ths wk, Im afrd.
Compression rate: 26%

- As I recall, you are a Scorpio. Your horoscope this week is very bad.
- Your recipe pricing has some stiff competition this week, and I can't afford it.
- As I recall, you are a Scorpio. Your horoscope says that pricing this week is bad, I'm afraid.
- As I recall, you are a Scorpio. You will be up against stiff competition this week.

Phrase: If you ever feel neglected, if you think that all is lost, I'll be counting up my demons, hoping everything's not lost.
Compressed as: If U evr fl nglctd, if U thnk tht all is lst, Ill B cntng up my dmns, hpng evrythngs nt lst.
Compression rate: 24%

- If you ever feel neglected, if you think that all is lost, I'll be counting up my demons, hoping everything's not lost.
- If you ever feel neglected, and give thanks that all is lost, I'll be counting up my damns, happening on everything that's not last.
- If you ever feel neglected, if you think that nothing's left, I'll be counting up my demons, hoping everything's not lost.
- If you ever feel neglected, then give thanks that all isn't lost, I'll be counting up my demons, and seeing how long they last.

Phrase: The path of the righteous man is beset on all sides by the inequities of the selfish and the tyranny of evil men.
Compressed as: The pth of the rghts mn is bst on all sds by the inqts of the slfsh & the tyrrny of evl mn.

- The pith of the rights men is best on all shows by the inquiries of the selfish and the tyranny of every man.
- The path of the righteous man is beset on all sides by the inequities of the selfish and the tyranny of evil men.
- The paths of the righteous man are best on all sides by the inquiries of the selfish and the tyranny of every man.
- The path of the righteous man is beset on all sides by the inquiries of the selfish and the tyranny of evil men.

Measuring Comprehension of a Text Compression Algorithm for SMS

Phrase: I have given no small attention to that not unvexed subject, the skin of the whale.
Compressed as: I hve gvn no smll attntn 2 tht nt unvxd sbjct, the skn of the whle.

- I have given no small attention to that nutty Unix subject, the skein of the while.
- I have given no smell at all to that nutty unvexed subject, the skein of the while.
- I have given no smell at all to that not unvexed subject, the skein of the whale.
- I have given no small attention to that not unvexed subject, the skin of the whale.

Note: *This one is interesting because all but one of the 'answers' would compress to the message shown.*
Phrase: Welcome to the mix and match world of composites.
Compressed as: Wlcme 2 the mx & mtch wrld of cmpsts.

- Welcome to the mix and match world of composts.
- Welcome to the mix and match world of campsites.
- Welcome to the mix and match world of composites.
- Welcome to the mix and match wielder of composites.

Phrase: The rules of poker are surprisingly simple – it's winning that can be hard.
Compressed as: The rls of pkr R srprsngly smple – its wnnng tht cn B hrd.

- The roles of pikers are surprisingly simple – it's whining that can be hard.
- The rules of poker are surprisingly simple – it's winning that can be hard.
- The roles of pikers are surprisingly simple – it's winning that can be hard.
- The roles in poker are surprisingly simple – it's winning that can be hard.

Measuring Comprehension of a Text Compression Algorithm for SMS

Phrase: George Romero started it all off in 1968 with his seminal horror classic, Night of The Living Dead.
Compressed as: Grge Rmro strtd it all off in 1968 wth hs smnl hrrr clssc, Nght of The Lvng Dd.

- Garage Romeo started it all off in 1968 with his small-hours classic, Night of The Living Dead.
- George Romero started it all off in 1968 with his small-hours classic, Night of The Living Dead.
- Garage Romeo started it all off in 1968 with his seminal horror classic, Night of The Living Dead.
- George Romero started it all off in 1968 with his seminal horror classic, Night of The Living Dead.

Phrase: It would be easy to think that Stevenson's Rocket was the only rail-going steam engine of its time, but this is of course untrue.
Compressed as: It wld B esy 2 thnk tht Stvnsns Rckt ws the only rl-gng stm engne of its tme, bt ths is of crse untre.

- It would be easy to think that Stevenson's Rocket was the only rail-going steam engine of its time, but this is of course untrue.
- It would be easy to think that Stevenson's Racket was the only rule-gaining stem engine of its team, but this is of course untrue.
- It would be easy to think that Stevenson's Rocket was the only rule-abiding steam engine of its team, but this is of course untrue.
- It would be easy to think that Stevenson's Rocket was the only rail-going steam engine of its time, but this is of course untried.