Designing mobile phones

Harold Thimbleby
Gresham Professor of Geometry

Thursday 28 November 2002

As long ago as 1084BC people wanted to communicate over long distances. A series of beacon fires lit on hills enabled Queen Clytemnestra to learn of the fall of Troy when she was five hundred miles away. According to the Greek historian Polybius, by 300BC, a more sophisticated method of signalling using a code was invented and in use. In this case, each letter of the Greek 24 letter alphabet was written in a 5-by-5 grid, and the code for each letter was first its row number then its column number. This code could be tapped out, and indeed similar versions of the code have been used by prisoners to tap to each other through cell walls, at least since medieval times. In 1551 the mathematician Cardano suggested using five torches in five towers. Even the Gresham Professor of Geometry Robert Hooke (1653–1703) made several proposals for long distance communication, stimulated by the invention of the telescope and the greater distances of communication this permitted.

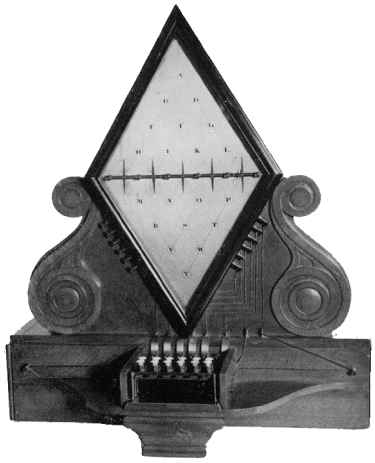
From the early eighteenth century, people were proposing to use electricity, but the only methods known then were based on static electricity and were horribly prone to leakage. An anonymous letter written in 1753 proposed using 26 separate wires for communication, one for each letter of the alphabet, but still using unreliable swinging pith balls attracted by static to read the message. Galvani's frog leg experiments were in 1790, and might have provided a more reliable approach, but serious developments had to wait until Volta invented the battery and the work of Ampere and Ørsted in electromagnetism for methods that were practical.

Depending on which side of the Atlantic your sympathies lie, attention then turns to either Samuel Morse in the US or to Sir Charles Wheatstone who worked near to Gresham College, at Kings College London. Almost simultaneously these two — and many others — were working on the telegraph. Both characters were highly creative; in fact Morse was an accomplished painter who exhibited in the London Royal Academy and was founder of the US National Academy of Design; whereas Wheatstone invented a wide variety of gadgets, including a pipe organ, the concertina and the stereoscope (but he didn't invent the Wheatstone bridge), as well as the Playfair cipher.

Wheatstone and Morse's approaches to communication were very different.

Wheatstone, with his colleague William Cooke, was first to develop a viable system that was made available to the public. It was first demonstrated in July 1837 with a telegraph line along the railway track from Euston to Camden Town, covering a distance of about 1.5 miles. They successfully transmitted and received a message. The impact must have been enormous; bear in mind that the Penny Post was introduced three years *later* — and a decent electric light bulb had to wait to the 1870s, and Sir Joseph Wilson Swan's patent (granted before Edison's, which fact, by the way, explains the brand name Ediswan).

Wheatstone's telegraph had five needles. Each needle could be swung to the left or the right by electromagnets, so it would then point along a diagonal line of letters arranged in two triangular grids. When not energised, the needles rested in a vertical position.



To transmit a letter, two switches were pressed which caused the corresponding needles at the other end to move and together point along two lines in the grid. The letter at their intersection was the chosen letter. Their grid allowed 20 letters, and J, C, Q, U, X and Z had to be omitted. If only one

needle moved, it was taken to point at a digit, so numbers could be sent easily. Despite its shortcomings, the advantage was that this telegraph could be used by unskilled operators: using the grid was pretty self-explanatory.

One of Wheatstone's telegraphs helped catch a murderer, John Tawell, on New Year's Day 1845. The message read:

> A MURDER HAS JUST BEEN COMMITTED AT SALT HILL AND THE SUSPECTED MURDERER WAS SEEN TO TAKE A FIRST CLASS TICKET FOR LONDON BY THE TRAIN WHICH LEFT SLOUGH AT 7H 42M PM HE IS IN THE GARB OF A KWAKER WITH A GREAT COAT ON WHICH REACHES NEARLY DOWN TO HIS FEET HE IS IN THE LAST COMPARTMENT OF THE SECOND FIRST CLASS CARRIAGE

The message travelled faster than the train. Tawell was apprehended near Paddington and later hung for his crime. Although the message was sent using a later two-needle machine, there was still no Q.*

The five needle telegraph required five wires (Wheatstone had an ingenious way of avoiding a sixth wire for the common return that would normally have been needed). Over miles, the cost of wire was not insignificant! The telegraph was soon replaced by a single needle instrument, to cut down on the number of wires required. Each letter of the alphabet was then given a code of right and left needle movements, but this now required skilled operators. One was needed to send the message, and two skilled operators to receive it, one to read the needles and one to write the message down. As events proved, Wheatstone's modified scheme was more tedious than Morse code.

Almost the same time Samuel Morse, working in the US, was developing his own telegraph. His first design was a machine that recorded the message on a moving paper tape, which he felt important for record-keeping purposes. Recording also made his telegraph uniquely different from all previous distance-communication approaches, such as smoke signals and semaphore, which leave no record of the message. Morse was aware of this, and thought it had potential. He soon found, however, that it was easier and faster for operators to listen to a buzzing sound, rather than read the dots and dashes and transcribe them — you can listen and write easily, but you can't watch and write easily, as Wheatstone had found out with his system. After several false starts, he devised the eponymous code, Morse Code although it was not at all like what we recognise today — and it isn't totally obvious that Morse himself invented it: there is a lively dispute whether he did, his assistant Vail did and whether third parties helped. Regardless, Morse was a great promoter of 'his' code.

Originally, Morse had each letter or digit coded using a metal ruler. One of Vail's innovations was to use a tapping key so an operator could tap out any code, an innovation that has both advantages and drawbacks. Probably the main advantages were the reduced cost achieved by not having to make the rulers, and the improved reliability not having to keep them clean. The operators no longer needed to hunt and find the right ruler, but they now needed to know exactly what code to send. Thus, unlike the Wheatstone system, operators of the modified Morse system needed serious training to use it effectively. Even so, speeds of ten or more words per minute were easy, and only two wires were required (or only one if the earth is used as a common return), which is a great commercial benefit especially for long distances.

Morse had some problems getting his ideas accepted in Europe, not only because of patents (his code was patented in the US in 1840), but also because of the tedious European habit of using accents.

The first US telegraph message sent over a decent distance, from the Supreme Court room in the Capitol to Baltimore, was transmitted by Morse and recorded on his paper tape. Morse gave credit to Annie Ellsworth, the daughter of a friend, for suggesting the message "What hath God wrought?" She obtained the text of the message from the Bible, Numbers 23:23. See http://memory.loc.gov/ammem/atthtml/morse2.html

As the first message sent suggests, it seems Morse was acutely aware of the potential for misuse of the telegraph: "be especially careful not to give a partisan character to any information you may transmit" he wrote to his assistant Vail. We could well heed this advice today!

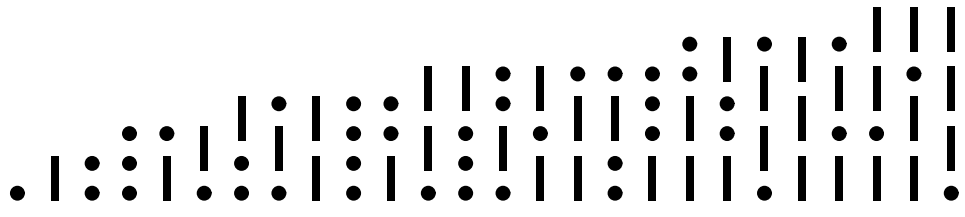
In Morse code, each letter and digit (and a few other signs) are represented by dots and dashes. For example Q is ▬ ▬ ● ▬, dash dash dot dash. Note that the dots and dashes must be separated by short silences, otherwise we'd just hear a continuous buzz. Q takes the time of 16 dots (the usual convention is that a dash is worth 3 dots, the gap between dots and dashes is 1 (silent) dot unit, and individual letters are separated by 3 (silent) dot units. Words are separated by 6 silent dot-length units. If all letters were coded like Q, every letter would take 16 dot units to transmit.

---

* There is a U in this message. I have not seen the original message, and I suspect centuries of transcription have taken their toll.

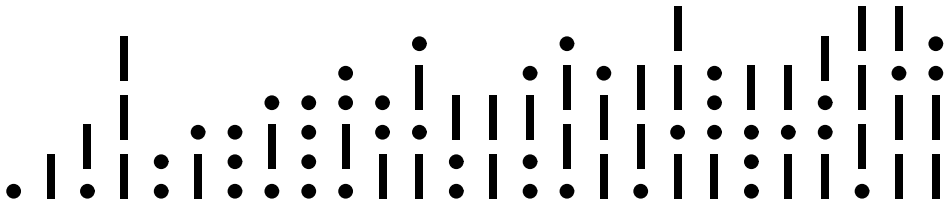
But Vail, apparently, realised that if the more common letters had shorter codes, even at the expense of making other letters longer, the average length of a message could be shortened. Thus E, the most common English letter, is represented in Morse code by a single dot. Curiously, the first Morse message had no Es.

The story is that Vail went to a newspaper printers and saw that they had more letter Es in lead type than other letters, because any page printed needed more Es than other letters to be in stock. Roughly, the Morse code length of each letter is inversely proportional to the amount of lead needed for it at the printers. If this story is correct, then as the bar chart below shows, the printers had most Es, then T, I, S, N, A, U, R, and so on, to Y, Q and J being least popular.



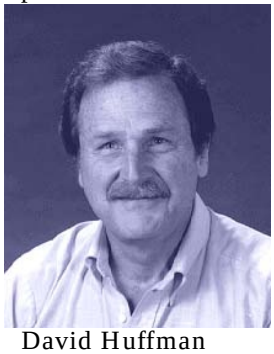
Morse could have got two more letters in at the same duration as Z, X, P, O or C, using dot dot dash dash and dot dash dot dash. These are codes that aren't used anywhere else: so with this insight we could have made the codes for two of Y, Q and J a bit shorter. Whether Morse wanted a six dot code, which would be the same duration, but harder to send is another matter; perhaps it would be confused with digit codes?

In fact, the letters in English ordered by decreasing frequency is more like ETAOIN… not Morse's ETISNA… The same graph plotted but with the letters in their actual frequency order (from the British National Corpus of 90 million words) looks like the graph below. Clearly, Morse is not optimal for English. As it happens the Brown Corpus, an American collection, has the same letter order as the British corpus, so Morse isn't ideal for US English either. It's even worse for Welsh, which has Y as its second most common letter.



On December 12, 1901, Marconi used radio to communicate across the Atlantic, and the era of Morse code took off, particularly for shipping — it was used famously to signal the sinking of the Titanic. Morse code faded out in 1999 when, by international agreement, it was superseded by modern digital methods: today if you have the necessary digital equipment to transmit messages, you aren't going to need Morse. Your mobile phone today, for example, is far too sophisticated just to send beeps.

Jumping past the development of codes in World War II (which we covered in two Gresham lectures last year), for our story today the next important development was Claude Shannon's mathematical theory of information and communication. This theory suggested that codes could be designed that were much more efficient, and there were fundamental limits to the efficiencies that could be achieved. Shannon, and independently Robert Fano found ways of creating efficient codes. The quest was on for a good way to generate efficient codes, and the breakthrough was achieved by an MIT undergraduate student, David Huffman. Huffman was a student of Fano, who had developed a coding scheme that was efficient. The story goes that Huffman stayed up all night and found a better way of coding than his professor. He passed his exams. Huffman died in 1999.



David Huffman

Although Morse is not binary, and nor are the examples we'll give later, we'll now describe how to construct a binary Huffman code — binary happens to be simple to explain.

In binary, we have two symbols 0 and 1 that are transmitted — in Morse there was dot, dash and the gaps between letters and the longer gaps between words, which amounts to using 4 symbols. Imagine a stream of 0s and 1s: those are the only symbols we have — there are no third symbols for gaps between letters. Suppose (as in the table below) that we have coded E as 011. That means that no other code can start 011. For example if T was 0110, we could not tell whether this was E followed by a 0 (starting something following) or a T (with nothing following). If we decide that E is 011, then there can be no other code starting with the sequence 011. We can use 00…, 1… and 010… for starting other codes, but we cannot use any code starting 011 (other than E). This property of a code is called *prefix free*: it means that no letter codes share the same prefix: in other words, in this case no other code has the prefix 011, otherwise it would be confusable with E.

Now consider the two least probable letters in English, Q and Z. Because they are least probable, then they must have the longest codes, but whatever their codes are they must be different. Suppose the longest code is XXXXX; if we could tell whether this was a Q or Z before we read to the end of it, the code could be shorter. Thus Q and Z must differ in their last digit. They will have codes such as XXXX0 and XXXX1, which differ in their last digit so that they can be distinguished. In the table below, we have chosen 000011001 and 000011000 for Q and Z. All other codes must differ from Q and Z *earlier* than the last digit — they can't differ later, because all longer codes starting like Q or Z obviously share the same prefix as Q or Z. So, for example, 00001101 is the next shortest *different* prefix; we can either allocate this to a more frequent letter (say, S) or we can distinguish at least two further codes after it 000011010 and 000011011. In fact, this is what we do for X and J. However, we allocate the next shortest code 0000111 to K, and this differs from X, J, Q and Z in K's last digit, which is a 1, whereas this digit is a 0 in X, J, Q and Z. There are clearly lots of decisions to be made; Huffman worked out how to make these choices optimally. The details are quite elegant and straightforward, but they don't concern us now (not least because *canonical* Huffman coding is better, but too complicated to explain here). The table below gives a Huffman code for English, based on the letter frequencies in the Brown Corpus.

E	011	L	11111	Y	101001
T	001	D	11110	B	101000
A	1110	C	01011	V	000010
O	1101	U	01010	K	0000111
I	1100	M	00011	X	000011011
N	1011	F	00010	J	000011010
S	1001	P	00000	Q	000011001
R	1000	G	101011	Z	000011000
H	0100	W	101010		

These codes look longer than Morse codes, and indeed they are because Morse code has more symbols to play with. Morse is prefix free, but doesn't look prefix free — E and T are dot and dash, respectively, and every other letter looks like it must therefore start with either an E or a T! Well, we didn't shown the end-of-letter pauses — no letter other than E starts with dot *and* end-of-letter-pause. Morse code has the advantage over a basic Huffman code that losing a letter or two (as could happen with a dodgy communication line) does not really matter; whereas, in the Huffman code above, if even one digit is lost, the rest of the message will be garbled.

There are lots of different Huffman codes for the same problem. For example, in the table above if we swapped 0 and 1 everywhere, we would have a different code, but it would have all the same desirable properties the original has. Or we could swap 0 and 1 in the second column, or whatever. There are lots of ways of making new codes, and all of them will be prefix free and compact.

Huffman codes are widely thought to be optimal in terms of producing the shortest coded messages. But there is no reason to suppose a *particular* message I send, like "What has God wrought?" should comply with the overall letter frequencies of the Brown Corpus. A Huffman scheme that adapts to the particular message's letter frequencies would be better for that message: this gives us the idea of adaptive coding. Furthermore, as a message proceeds, the relative frequency of letters changes: in some parts of a message E may be rare and in other parts common. An adaptive compression scheme changes the code as it proceeds.

Morse's original message contains only 10 different letters, and we could design a special code for them, rather than the whole alphabet, more efficiently — we don't waste code space for letters that are never going to be sent. There are many ways of doing even better, for instance if we coded pairs of letters (digrams) rather than individual letters, in most messages many pairs don't occur at all

(such as QX, QY, QZ…) and we don't need to code them. In Morse's message, we effectively reduce the alphabet to one about 1.5% the original size of 26×26=676 pairs, an even better saving than getting to 10 from 26. We can code *that* even more efficiently. It effectively codes letters with fractional numbers of bits.

"Recoding" a Nokia mobile phone handset

So far we have considered using coded messages for *people* to communicate with each other, when that communication is mediated by a telegraph or radio link that can only handle certain symbols, such as dots and dashes (and spaces) or 0s and 1s. We saw that compression techniques allowed us to reduce the number of wires needed from 26 to 1 (or 27 to 2 if we count the common return), and we saw that the ideas of Morse and Huffman allowed the codes to be chosen to reduce the time taken to send a message. All these ideas work equally well when we are sending a message from one *thing* to another *thing*: the 'things' needn't be people, and the messages needn't be English.

Now consider the following problem. I have ten fingers and I want them to send 'messages' to my mobile phone. The messages are going to be coded instructions to get the phone do useful things for me. For practical reasons, namely that I need one hand to hold the mobile phone, I am only going to be able to use one hand for typing my message. Furthermore, the phone, being a phone, has 10 digit buttons already on it. To ask for many more buttons would make the phone bigger and heavier, which I don't want.

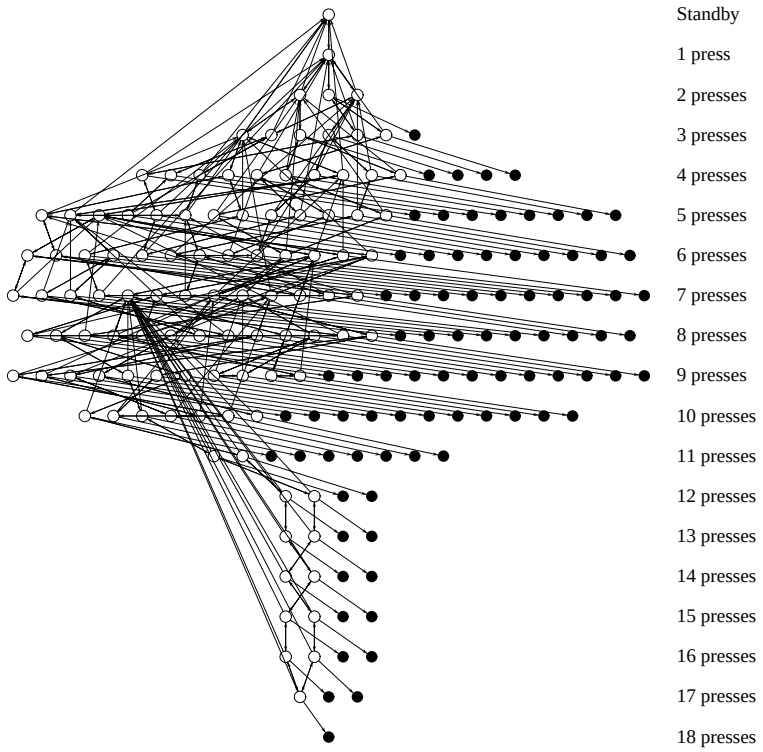
If I want to select functions on my phone fast, I can use a Huffman code that uses ten digits (rather than two binary digits) to select from the phone's functions. We will need a prefix free code, for the obvious reason that if XXXXX is the code for redialling, we don't want to have to say XXXXXY to actually dial, we want the code to specify dialling unambiguously at the point when we've finished saying do it.

For concreteness, take the Nokia 5110 mobile phone. We will be concerned with the Nokia 5110 mobile handset's menu functions, though there are a number of essential functions that are not in the menu (such as quick alert settings and keypad lock). There are 84 features accessible through the menu. A softkey, labelled '–' called *Navi* by Nokia, selects menu items; keys ∧ and ∨ move up and down within menus. The correction key C takes the user up one level of the menu hierarchy, whose structure is illustrated in the summary (below). With reference to the summary, the function *Service Nos* can be accessed from *Standby* by pressing Navi [the phone now shows *Phone book*], then pressing Navi [shows *Search*], then pressing ∨ [shows *Service Nos*] followed by a final press of Navi to get the function to work. All menu items have a numeric code (diplayed on the Nokia's LCD panel); for example, *Service nos* can also be accessed by pressing Navi 1 2 (no final press of Navi is required).

Phone book, *Navi*-1,
    Search, *Navi*-1-1,
    Service nos, *Navi*-1-2,
    Add entry, *Navi*-1-3,
    Erase, *Navi*-1-4,
    Edit, *Navi*-1-5,
    Send entry, *Navi*-1-6,
    Options, *Navi*-1-7,
        Type of view, *Navi*-1-7-1,
        Memory status, *Navi*-1-7-2,
    Speed dials, *Navi*-1-8,
Messages, *Navi*-2,
    Inbox, *Navi*-2-1,
    Outbox, *Navi*-2-2,
    Write messages, *Navi*-2-3,
    Message settings, *Navi*-2-4,
        Set 1, *Navi*-2-4-1
etc

There are some complications I shall ignore; for example C does not work with short cuts (e.g., Navi 2 C 1 is equivalent to Navi 2 1, as if C was not pressed). It is important to note that there is no fixed relation between short cuts and the function's position in the menu, since some functions may not be supported (e.g., by particular phone operators): if *Service Nos* is not available, pressing ∨ would move from *Search* directly to *Add entry*, but the shortcut for *Add entry* would still be Navi 1 3 and trying Navi 1 2 would get an error.

We can visualise how a user chooses functions by tracing routes through the following diagram:



The black dots in the diagram represent menu functions. *Standby* is at the top and each press of a button (∧, ∨ or Navi) moves downwards (or diagonally downwards) — if you make a mistake, not taking an optimal route to a black dot, you go upwards. The diagram has been carefully drawn so that this is so. When you move through the diagram and land on a black dot, this is a menu function. As can be seen, some functions are easy to activate (they are near the top), and some require quite a lot of pressing — these are the ones lower down in the diagram. For clarity, I've deleted all lines out of black dots: they all simply go back to *Standby*.

If this isn't clear, or you haven't got a Nokia 5110 to play with, I have put a Javascript simulation of the phone at www.uclic.ucl.ac.uk/harold/ansim — where all other details are available too.

In order to use a compression scheme like Huffman's, we need to know the probability that users do individual functions on the phone. We could do some experiments, but let's suppose Nokia has done them, and they chose the menu structure accordingly to make it as easy to use as possible. If so, then Nokia consider the function that takes 18 presses to get to to be the least likely to be used (how often do you change your phone to use Suomi?), and for instance, less likely to be used than either of the two functions taking 17 presses. For the sake of argument, I take the probabilities of functions to be proportional to the reciprocal of the number of button presses the Nokia takes. If a function takes 17 presses, then its probability is 1/17 compared to the others — this gives us a Zipf distribution. In particular, the second most frequently used functions are assumed to have a probability of one half the most frequent. Obviously all the probabilities add up to 1, since the user is certain to do something. Working out the details, we get the following table:

Function	**Rank**	**Presses**	**Assumed probability**
Search	1	3	0.0613
Incoming call	2	4	0.0306
Inbox	2	4	0.0306
Speed dials	2	4	0.0306
Service nos	2	4	0.0306
…	…	…	…
Português	14	16	0.00438
Svenska	14	16	0.00438
Español	15	17	0.00408
Norsk	15	17	0.00408
Suomi	16	18	0.00383

On the Nokia, we are using 4 keys (∧, ∨, Navi and C) to select from 84 functions. Getting to a function takes anything from 3 to 18 presses; the average is 8.83, or if we weight by the probabilities (so harder things are done less often) the average is 7.15 key presses.

We can create a Huffman code, retaining the C key for corrections, so we only use 3 keys (∧, ∨ and Navi) for the actual code. It then takes an average cost of 4.14 presses to access any function, or 4.04 weighted with the probabilities. This is a considerable improvement, almost twice as good.

But Nokia have also provided a 10 digit code for short cuts. A user can either use ∧, ∨, Navi and C to search the list of functions, or if they know the short cut number, they simply press Navi then the short cut code. The average cost of a short cut is 3.39 key presses. This is better than our Huffman code because it is using ten digits, rather than three menu selection buttons, and so it can be coded more efficiently. But if we use a ten digit Huffman code, the average cost is better still, down to 2.98.

It's possible that Nokia chose their short cuts very carefully for some special reason. For example, if they bring out a new phone, perhaps they want Navi-2-5 to do the same thing on the new phone too. So they can't just use a Huffman code to allocate the codes; there are constraints. (My guess is that the overriding 'constraint' is that it is easier to program the short cut codes the way Nokia did; I can't think of any genuine usability reason for them to be as they are.) However, the Nokia short cut codes don't use all the codes available, and we can squeeze a Huffman code into the remaining short cuts that aren't used by Nokia. This does worse than an unrestricted Huffman code, but surprisingly at an average of 3.09 it does better than Nokia's original codes! What's interesting is that we can now have a phone with Nokia's original short cuts, to preserve whatever benefits they have, *and* we can have a faster Huffman code. The two codes can co-exist, and users can use the Huffman code whenever they want to be faster — which is presumably the point of short cuts. For some functions the Nokia codes are better than the squeezed-in Huffman codes, and if a sensible user uses the shorter of the two (if the new Huffman codes are worse, we don't need to say what they are: just provide one shorter code for the function if the Nokia short cut is better), the average drops to a mere 2.69.

This is a most interesting result, for two reasons. First, we've got an improved user interface design for the Nokia mobile phone without losing any of the (purported) advantages or features of the existing design. Secondly, there is a theoretical intrigue. Just how can using Nokia codes have got us a better result than a Huffman code, which is supposed to be optimal? The answer lies in the fact that Nokia's short cuts aren't prefix free — they rely on the user pausing between codes. In fact, the pause amounts to using an extra key `[PAUSE]` as it were. A Huffman code with an extra 'key' could do even better — though of course a pause takes extra time.

Design	Best case	Worst case	Average
Original Nokia	3	18	7.15
Huffman, 3 key	3	5	4.04
Nokia shortcuts	2	5	3.39
Unallocated Huffman	2	4	3.09
10 digit Huffman	2	4	2.98
Shortest	2	3	2.69

More usable techniques

Huffman codes maybe aren't very memorable, but if you want to be quick they are ideal — they would suit users who frequently do certain phone functions and therefore easily learn the functions' short cuts out of habitual use. More intriguingly, the dramatic improvement of Huffman codes in terms of keystrokes needed to tell a mobile phone what function to select shows us that there is potential room for improvement, even if you don't think much of Huffman codes themselves. We would like more memorable codes. Given that it is possible to do much better than the original design, can we now find a compromise that is faster *and* easier?

The digits on a mobile phone can also be used for writing text with. So, for instance, the digit `[2]` has letters A, B and C on it; the digit `[3]` has D, E,F on it, and so on. Text messages are sent by pressing digits to choose letters; so you could send the message ACE by pressing `[2]` once, pausing, pressing `[2]` three times, then pressing `[3]` twice — a total of 6 presses and a pause. This is an unambiguous, but slow way of typing. We can do better, since `[2][2][3]` can only spell out one of the following 27 words: AAD, AAE, AAF, ABD, ABE, ABF, ACD, **ACE**, ACF, **BAD**, BAE, BAF, BBD, BBE, BBF, BCD, BCE, BCF, **CAD**, CAE, CAF, CBD, CBE, CBF, CCD, CCE or CCF. Only three of these are real English words, those that I've shown in **bold**. So if we arranged that we could type this way, we might be able to type ACE by pressing the three presses `[2][2][3]` then taking two more presses to choose ACE from the properly-spelt alternatives. This is a small saving of about one press for this example; but if we were trying to type a longer word, not only would the saving be better, but there would be fewer properly spelt plausible words to choose from.

There are various schemes for typing on mobile phones like this; Tegic's (www.tegic.com) T9 system is probably most familiar. Seasoned text users devise their own codes that are even more

efficient, and a sensible text system would allow users to add their own words (and proper names, *etc*) to the dictionary.

We can use the same idea to type not words from an English dictionary, but words from the relatively short list of functions the phone supports. So, our dictionary will have words like Search, Incoming call, Inbox and so on. We press Navi first — just as we would to start a standard Nokia short cut. Now, however, we spell out the function using the letters on the digit keys. If we want the *Inbox* function, we type [4] [6] [2] [6] [9]. In fact, *Inbox* is found before we've needed to type all these digits.

We can do even better. Suppose we cannot remember whether the Nokia function is called *Speed dials*, or *Dial speed.* Our dictionary can have two (or more) entries for each function: in this case, one for *Speed* and one for *Dials*. So long as the user can remember at least one word, they will be able to find the function.

We built a phone using a variation of this idea, and we evaluated it comparing it with Nokia's original design. It did better. In one of our experiments, the Nokia design took subjects 16.52 presses against the redesign of 9.54, a strongly significant result ($p<0.001$). Almost every subject preferred our modified design. Our simulation has been exhibited at the Science Museum in London; see people.cs.uct.ac.za/~gaz/res/gadget/index.htm

One important way our new design excels is when a user does not know where the function they want is. In the original Nokia, if you have to search the menu hierarchy for a function (you know it's there, but you can't remember it's name or what subsection it is in), you are setting out on an exploration that will take over 100 presses — if you don't make a mistake (e.g., it's easily possible to cycle around submenus endlessly unless you recognise when you are going around in circles). In contrast, the redesign presents all function names that have not yet been resolved; in particular when the user has only pressed Navi, there is a list of every function to choose from. (If you pressed, say, [2] next, you'd get a much shorter list, of every function starting with A, B or C only.) This means that in the same situation, you press Navi and then simply scroll down the list till you recognise which function you want. There is no chance of going around in circles, and you don't have to do special things to 'get into' submenus and get out of them when you've finished exploring them.

Another improvement to the design was to make the short cut feature "modeless." In a normal phone, the user is either dialling or doing menu selection (or whatever they want to do after getting to a phone function). In our modified design, the phone did *both* as the user entered numbers; thus the user could decide what they were doing when they had finished entering numbers, rather than having to decide first. This flexible rearrangement in order avoids a classic, and common, user error (a so-called post-completion error). On our modified phone, you can't make the easy mistake of typing in what you think is a short cut only to notice you forgot to press Navi first.

We'll give details in the lecture, or you can read up in our paper "Data Structures in the Design of Interfaces," in *Personal and Ubiquitous Computing.*

Adapting to the particular user

In our original design, the list of possible functions that match what the user has typed so far is given in alphabetical order. We saw earlier that an adaptive system is more efficient, and we can make the mobile phone adaptive here too. The phone counts which functions the user uses. The lists of choices are then displayed in most-frequent-first order. Now, the most frequently chosen function *whatever it is* can be accessed by the user in just 2 keystrokes: namely, Navi — it's then shown at the top of the list — then Navi to select it. Most of the next 8 most popular functions can be selected in 3 presses, namely Navi, then the digit corresponding to their first letter, then Navi to select them from the list. A nice feature is that the functions the user never uses, and therefore is unfamiliar with, remain in alphabetical order for easy searching. An alternative design, which reduces the possibly surprising adaptiveness, is to display the first 4 matching functions in frequency order, then all matching functions in alphabetical order regardless of frequency.

We haven't done the experiments that might help us decide whether 4 ought to be 5 or 3 or something else. If I was rushing a new phone to market, I would make the number user-selectable. Some users probably want 3 choices, some want more, and preferences would let them be in charge of their own user interface.

Driver performance

All of the mobile phone schemes we have considered require the user to look at the phone's screen to see what is happening, or the user has to be *very* skilled to know all the arcane key sequence that activates the required functions. Older drivers with reduced accommodation (i.e., weaker eyesight) will find focusing on the phone's small screen difficult: a head up display could be used — which would have the advantage of making the driver at least look in the right direction — but it would not avoid the long process of selecting functions. It is possible to use speech (input or output) to help select functions, but this is demanding and requires considerable concentration. When driving a car, the driver has more important things to do than select one out of many functions on a mobile

phone. The only safe solution is for the phone to disable menu selection (and text messaging) when the car has the ignition switched on.

No phones on the market do this. If it seems too radical to lock out all menu functions from the driver, a weaker but important improvement to make would be to disable all time-outs in the phone. On my Nokia, if the driver does nothing for 10 seconds while trying to find some function, the phone changes mode to explain what the current function does. Now the user interface is subtly different — to continue searching for the function, the driver either has to wait about 30 more seconds (to let the phone get to the end of its help), or the driver has to press buttons to get out of the help and back to where they were. This is exactly the sort of unpredictable problem drivers do not need.

On the plus side, a recent survey has shown that mobile phones reduce smoking amongst teenagers, probably because it is just too tedious to hold a phone, dial, speak (or use texting) and smoke all at the same time. So, perhaps, there are good reasons not to make using mobile phones any easier.

Conclusions

We have covered the history of long distance communication, covering a period of three millennia, from hilltop fires to digital mobile phones. The earliest example covered 500 miles; the last example covered the inches between your hand and your mobile phone. The general theories of communication, which we illustrated with Huffman codes, applies equally to conventional communication problems as well as to modern user interface problems. We did experiments with real users and found that the theoretically-motivated design ideas did better than established market-leading commercial designs. Our conclusions are that (*i*) user interface design can always be much improved by a little mathematical, abstract, thinking; (*ii*) success in design depends on usability — from issues of the balance of training versus usability of the earliest telegraphs, to present-day mobile phones, where the right design balance depends on the task the user is doing, and whether they are driving, for instance.

Further reading

The best book on compression is *Managing Gigabytes*, IH Witten, A Moffat & TC Bell, 2nd ed, Morgan Kaufmann, 1999.

Full details of the Huffman code applied to Nokia mobile phones is in H Thimbleby, "Analysis and Simulation of User Interfaces," *Proceedings BCS Conference on Human-Computer Interaction*, edited by S McDonald, Y Waern & G Cockton, **XIV**, pp221–237, 2000, which is also available at www.ucl.ucl.ac.uk/harold/srf

G Marsden, H Thimbleby M Jones, P Gillary, "Data Structures in the Design of Interfaces," *Personal and Ubiquitous Computing*, **6**(2), pp132–140, 2002.

The information about Wheatstone draws on G Hubbard, *Cooke and Wheatstone*, Routledge & Kegan Paul, 1965.

Claude Shannon's epochal work is still in print: C E Shannon & W Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, 1998.

Acknowledgements

Paul Gillary, Matt Jones and Gary Marsden worked with me on the design ideas described in this paper, and they did all of the usability experiments. Orange donated some of the handsets we used.