HUNTING THE TEN-SQUARE

REX GOOCH
Letchworth Garden City, Herts, England
rexgooch@ntlworld.com

A ten-square consists of ten (different) 10-letter words written one under another, so that the first column reads the same as the first word, the second column the same as the second word, and so on.

Knowing how many capable people, over so many years, had tried to make a ten-square, for a long time I ignored suggestions that I should try to find one. I suppose it was inevitable that, after years of watching from the sidelines, I should be seduced into this way of passing a lifetime uselessly. Eventually, however, I paid attention to one series of blandishments (inspired by perhaps not the purest of motives), and started to investigate, hoping that I had not mortgaged the remainder of my stay on earth. I knew I had a sizeable list of words, so wrote a program. It ran slowly, so as a test I used a much smaller word list sold to me by Ted Clarke. I knew that Ted had spent years with computers trying to find ten-squares in this list of half a million words of all lengths, so when my program finished in well under an hour, I concluded that my program was not so bad. It found a few "ten-squares", some of the type I called quarter squares (ie with repeated tautonyms), and some containing non-dictionary phrases. These were not what I was seeking.

Readers are cautioned not to relate numbers in this article to other numbers, because the list of words and the program were changing very rapidly at times. Throughout, "k" is used to mean thousand, and M means million.

Brief History of Ten-squares

As ten squares have been produced by many people over almost a century, it will not be feasible to give a full history here. A longer treatment of the subject was written by Ross Eckler for a tribute volume for Martin Gardner: this has yet to be published. As will be seen later, it is rather simple to produce squares consisting of tautonyms, especially if repetition is allowed. This idea was introduced in 1921 by Tunste [Paul Bryan], who published some squares in *Enigma*: in 1926 1000 very similar squares came from Arthur Holt (who had produced the first nine-square in 1897). All the words were given sources, often from the South Seas (eg *Aardrijkskundig en Statisch Woordenboek van Nederlandsch Indie)*, but many proved impossible to find in 2002, on the Internet or otherwise. In 1925 Dudeney said that he had never seen (even) a good eight-square. By 1965, little progress had been made: Dmitri Borgmann gave a number of tautonymic 10-squares in *Language on Vacation*; for example a square starting ORANGUTANG, which has only slight changes to Holt's 1926 effort. Such squares abound: although tautonyms account for a mere 0.2% of my enlarged word list, they account for 29% of the ten-squares I have found, and a further 18% of squares have 9 tautonyms of the 10 possible.

In February 1977 (*Word Ways*), Frank Rubin found eight good words, and wisely left the last two incomplete.

An even easier route than tautonyms is to use rows consisting of two or three shorter words to make a possibly plausible phrase such as ONE EYE ACHE or CRAP SQUARE, albeit not sanctioned by any dictionary. Over the last decade, Ted Clarke has always needed such phrases to make a square. (SOUR GRAPES, being in a dictionary, would be more acceptable!) For more, click on **About Ted Clarke** The interesting story behind the mastermind in http://www.dictionary-thesaurus.com. The use of two or more words in place of a 10-letter word is to be deprecated: it enables combinations of vowel-vowel and consonant-consonant to be placed in the middle of a word that otherwise would not be found there (though compound nouns may have the same effect, they are far scarcer).

More kosher squares have been produced over the last decade or so by Jeff Grant: the ASTRALISED square (W90-198) which uses two personal names, and probably the best square by 2002, DISTALISED (W2002-8). In the October 2002 issue of Word's Worth, Ted Clarke published a better square, sadly without asking the permission of the author (me!) nor of the publisher to whom it had been sent. Indeed, neither of us know which square was published: it is, after all, in breach of copyright. The following issue contained derogatory comments on the square, in accordance with his views on "American so-called logologists". My apologies to the many people such as Darryl Francis who did good work, but have not been mentioned: some are credited later in this article.

In addition to actual squares, much has been written about techniques for finding them, and what size of vocabulary would be needed.

Methodology

The problem
Finding the best ten-square requires a large list of words, and (realistically) a program or method. There is an obvious method: take a word for the top row, find a word starting with the second letter of the first word: this makes the second row. Find a word whose first two letters are the third letters of the first and second words, and so on. There have been discussions in the past about whether it is better to proceed top down or bottom up, but this is a secondary issue. The problem is that you need a large number of words (250,000+, see later). Suppose for a moment that you are able to access the set of suitable words for each row instantly (see later). There are 250000 possible top rows. The number of words suitable for the second row we can guess would be 1/26 of this; the number for the third row would be 1/26 the number in the second row, and so on. The fourth row would appear to have typically less than one suitable word on average. Nevertheless, just the first three rows have almost one trillion 3-row starts to a square.

The first problem is therefore the huge number of valid combinations for the first four rows. However, when placing the second word BK…

A	B	C	D	E	F	G	H	I	J
B	K	M	N	O	P	Q	R	S	T

it would seem sensible to ensure that there are words starting CM, DN etc. Sadly, with such a large word list, 90% of all possible 2-letter starts exist, so the time taken to carry out the eight checks may be largely wasted, and it might be faster to allow invalid starts to be caught only after placing the third or fourth words. Unfortunately, it takes a long time to run a representative test. However, this type of pruning must be carried out for subsequent rows.

This crude analysis shows that by row 5, there is fewer than one word to fit each of the 12 trillion combinations of the first four rows, and an ever decreasing chance of finding one word as we go deeper.

The result is that the number of partial squares now diminishes at each level until we have very few, if any, at the tenth level. Simple though this analysis be, it does depict qualitatively what happens, and shows where efficient pruning is required.

The general picture painted by the crude model above is correct. In practice, a quick sample while running 692k words revealed that, after pruning, for each top row word, 46,523 second row words will fit: and for each of these, 273 third row words will fit, making 12.7 million words to be handled at the next step. After that, the pruning begins to have more effect, so that for the next step only 1.2 words fit per word. After that the numbers decline (to almost zero after the tenth row!). If these averages held throughout the whole corpus, the number of fourth row words to be handled would be more than 1 followed by 13 zeroes. The problem grows explosively as the number of 10-letter words increases. These data apply when the program is running fast, ie pruning is successful, and lower rows tend not to be visited.

Accessing the right words
Because the whole job is liable to take a long time on PCs, it is important not to waste time searching through words that are not relevant, and virtually never to access a disk or diskette, whose timings are in milliseconds, perhaps a million times slower than main memory (indeed, sophisticates might attempt to keep many references within the cache, which is faster than main memory). Even worse is to write intermediate results on the screen, as Ted Clarke did. So if we want a word beginning ABC, we should only access such words. This is not a treatise on techniques, but here are the outlines of four methods. We suppose in all cases that there are 260000 10-letter words (occupying at least 2.6 million bytes), and that the programs run under an operating system such as Windows, which may take 100MB of memory. You will also need to produce fast code for the computer, by using a first rate compiler. Assume the word list is in alphabetic order.

1. Indexing. For example, construct a table so that the first entry points to the first word beginning AAAA in the word list, the second entry points to the first word beginning AAAB, and so on to ZZZZ. You will need 456976 entries (26^4), taking 1.8MB. To access the first word beginning ABCD, calculate (a-1)*26*26*26 +(b-1)*26*26 +(c-1)*26 +d, where a, b, c, and d are the numerical values of A, B, C, and D—respectively 1, 2, 3, and 4. The arithmetic will be fast, and this scheme will run through a word list of 60,000 (about the size of Ted Clarke's) in a very few minutes. One reason is that relatively few potential squares in this size of list get as far as word 5. However, as the word list increases in size the number of potential squares increase dramatically at each level, so on average you have to descend further before dismissing a particular square. Thus the few minutes become tens of years for word lists capable of producing ten-squares. Some relief may be obtained by using an index 26 times larger, giving instant access to the first five letters, but six letters demand over 1.2GB of memory for the index, and few, if any, PCs have this much. Of course, the idea of an index being far larger than the data indexed is a little unusual: and huge runs of the same pointer will occur. There are various ways of coping, such as using sparse index techniques.

2. Binary search (or binary chop). Look at the middle word in the list: see if the word (or word start) you want is nearer the start or the end: if so look at the word halfway between the centre and the start/end respectively. Continue like this until you have the word you want. You may have to take about 18 steps to find your word, which takes time.

3. Randomising. The traditional idea here is that you convert each word into a (almost unique) number, and store the word at that location. For efficiency, you normally choose a number of locations somewhat larger than the number of words: some will be therefore be empty. An obvious way of turning a word into a number is to use A = 1, B = 2 etc, and calculate eg 26*26*a + 26*b + c for a word ABC, and

similarly for a 10-letter word. The numerical results may need 14 decimal digits, and this poses a problem, for such large numbers are not handled by hardware, so you need to use the right software, and arithmetic will be slower. The next step is to randomise this number: this is often done by taking the remainder after dividing the randomised number by the number of locations reserved, or, much more sensibly, a prime number somewhat smaller than the number of locations. This remainder designates the location in which you store this word. The effect is that most words will find a location to themselves, a number of locations will be empty, and sometimes two or more words will try to occupy the same location. This last situation is dealt with by forming them into a chain.

Our application of this method is different. Go through the word list in alphabetic order. Whenever you have a new 6-letter (or 7-letter) start to a word, randomise those 6 letters, and place in the calculated location the position in the word list of the word. When you want words with a given 6-letter start (for row 7), or wish to check for the existence of words with such a start, randomise the six letters, and the number in the calculated location will give you the position in the word list of the first of such words. Some details of this method have been omitted, but there is no problem with arithmetic, and relatively little memory is needed. This therefore is a way of getting fast access to relevant words when the direct indexing method uses too much memory. A combination of both methods would be sensible. There are many variants of randomising, and trial-and-error is called for. My experiments with six and seven letters in the 692k word list gave an average of about 1.1 probes per word requested, though changing the detailed conditions would improve this figure. At some stage, perhaps eight letters, there are so few words with such a long start that it might be sufficient just to use the 7-letter search, then check every such word until words starting with the desired eight letters have been passed: the number of searches at these levels is comparatively small anyway.

4. Trees/DAWGs. This is based on Graph Theory, where a graph in the mathematical sense is a set of nodes connected by paths. Suppose your word list consists of just the words A to Z, CAT, CAM, and CATHODE. Imagine each word written down the page. You start by pointing to A, the first of 26 nodes containing the letters A to Z (since all are words), with the A-node having a pointer forwards to the B-node, the B-node pointing forwards to the C-node, and so on until Z has a null forward pointer. All these top-level nodes also have a downward pointer, which is null except for the C-node, which points down to another A-node. This A-node has a null forward pointer (no words like CB, CC...CZ), and a down pointer to M. The M-node has a null down pointer (no other words beginning CAM), an end-of-word indicator (marked as ' below), and a forward pointer to T. The T-node has a null forward pointer, an end-of-word indicator (CAT), and a down pointer to H. This in turn points down to O, which points down to D, which points down to E, which has an end-of-word indicator (CATHODE).

$$\begin{tikzcd}
C' \arrow[d, dash] & \\
A \arrow[d, dash] & \\
M' \arrow[r, dash] & T' \arrow[d, dash] \\
& H \arrow[d, dash] \\
& O \arrow[d, dash] \\
& D \arrow[d, dash] \\
& E'
\end{tikzcd}$$

This tree, as used in word programming, has a number of modifications. No node exists unnecessarily: for example, the node P below C will not exist if there are no words beginning CP. Forward pointers can be omitted if the nodes are physically adjacent. Words with common endings like -ING all use the same ING nodes (so it is no longer a tree, as the twigs join up). An important reference is Andrew Appel and Guy Jacobson: *The World's Fastest Scrabble Program*, CACM, Volume 31, Issue 5 (May 1988), which gave the name DAWG (Directed Acyclic Word Graph) to this structure. The number of nodes required is typically less than the number of words — far less for small vocabularies. Graham Toal tells me that 592361 words needed 428543 nodes.

You can get a word list in alphabetic order by traversing the whole graph. For example, you can find all words beginning CAT by following pointers to C, then A, then T, then traversing the sub-tree.

Each node will have the letter, an end-of-word indicator, a down pointer (address, index), and an indicator for the end of a horizontal chain (T in the above example). These all fit into 4 bytes, allowing about 16 million words; in that case the nodes would occupy 64MB, which is available on recent PCs. Slightly different formats may be faster for higher-level programming languages.

A more detailed description of the DAWG structure can be found at http://www.gtoal.com/wordgames/wutka/dawg.html. Graham Toal's demonstration ten-square code can be seen at http://www.gtoal.com/wordgames/wordsquare/traditional-stable.c.html.

The DAWG or tree approach stores individual letters with pointers to define the words. The other approaches store whole words, though groups of letters smaller than a word may be processed. Nothing forbids use of more than one approach.

Some Characteristics of 10-letter words

It is often possible to run programs faster if they make use of some knowledge of the data. One such characteristic is the fact that 10-letter English words are sufficiently long to have a two or more of a prefix, a body, and a suffix. This causes clumps of words within an alphabetic listing. Here is some information on starts and ends of words in my 10-letter word list: the dominance of place names is obvious. Without them, a different picture would emerge, though one also with clumpiness. The clumpiness is more evident as the number of words increases.

Most popular word starts:

len of start	max no grps	actual # grps	Most common		Second most common	
2	676	603	MA-	18269	BA-	18209
3	17576	7157	BAN-	7143	BAL-	1755
4	456976	53709	WADI-	3652	MONT-	2055
5	11.9M	186015	SUNGA-	1642	CERRO-	1496
6	3.1B	366818	SUNGAI-	1622	ARROYO-	368
7	80.3B	517374	SUNGAIA-	195	BANNONG-	184

Thus five letters have 11.9 million possible combinations, but the 693,000 words use only 186,000 of the combinations: there is an average of 7 words for each combination (and the 7 hides wild swings). SUNGAI is Malay for river.

Endings show a very similar picture:

len of end	max no grps	actual # grps	Most common		Second most common	
2	676	580	-NG	23161	-ES	21112
3	17576	6955	-ING	11426	-ANG	4203
4	456976	49463	-IVER	3713	-BERG	3041
5	11881376	169508	-RIVER	3633	-CREEK	2423

Advocates of the bottom-up approach have cited the frequent occurrence of certain ěndings as having the potential to avoid repeated tests; but for 10-letter words in my vocabulary, it can be seen that there are also many common starts.

Without NIMA place names, the most popular (second most popular) starts are: CA (BA), DIS (CON), OVER (INTE), INTER (BLACK), DOUBLE (CONTRA), COUNTER (CONTRAR). The most popular endings are: ES (ED), ING (ERS), TING (ABLE), ATION (ENSIS).

Suppose we place a word in the third row: we would only try those words beginning with the two letters forced by the start of column 3, say WA. The next letter will fit both the third row and the third column. Suppose we have worked through all the WA words until we come to the first WADI. The I is the third letter of a word in column 4: suppose no such word exists (no DII...). For the third row, we can now skip to the next word after the WADIs, saving 3651 attempted placements. This seems efficient, but the savings must be offset against any extra costs in the majority of cases in which there is no saving.

In general terms, we should try to avoid doing the same check multiple times, ie to remember failures; using, if necessary, ancillary indexes or tables. The advantage of the DAWG approach is that, for any partial word, you know quickly which letters come next.

Vocabulary

I started with almost 140,000 10-letter words (1.2M words in all). This fraction of 10-letter words was to remain between 11% and 12% throughout. My program found no quality square. This was sad, but not unexpected. What was needed was an estimate of the number of words required, so that I could judge whether the task might be possible. Defining the "support level" as the number of words needed to give a 50% chance of finding a square, Chris Long's formula (see W93-5) gives 247717 for a ten-square. His formula is simply p to the power -4.5, where p is calculated from the (square of the) frequency of each letter in the word list, and has an experimentally determined value of about 0.063291. It is a measure of unevenness of occurrence of the letters: if all letters appear with equal frequency, p will be 0.038: if one letter appears 5 times more often than any of the others, p will be 0.056. Worse news was that the value of p for my words was 0.063191 meaning that 249489 words would be required.

Chris has another formula, also using p, which calculates the expected number of squares from a given number of words. It is the number of words in the list raised to the 10th power, multiplied by p to the power 45. This is very sensitive to p, but also to the number of words. A 1% increase in p will increase the expected number of squares by about 60%, and a 10% increase in the number of words will multiply the expected number of squares by about 3. The values of p^{45} for the ps given above vary by 8%, at about 10^{54}. The other term, based on 250000 words, is about 10^{-54}, giving the expected single square.

What was therefore surprising was Ted Clarke's claim to have found a number of 10-squares from a population of about 66196 unsourced (and often unsourceable, so not part of my list) words, as sold to me. This number is about enough to produce one nine-square, and Chris Long's formula gives a

1-in-a-million chance of finding a 10-square in this number. Indeed, in a matter of minutes my program found all the possible 10-squares in the Clarke vocabulary, including some he had not reported. There were six, an example being one starting TOPROWWORD. Apart from some previously published squares, they all contained such invented phrases: two 5-letter words, or a 4- and a 6-letter word (possible example: CRAPSQUARE). Now if I used two 5-letter words for each row, I would have 6 followed by 44 zeroes of 10-squares, of which surely a few trillion trillion would be rather nice. Moreover, if I used a 6-letter word plus a 4-letter word for each row, there would be a further 7 followed by 42 zeroes of 10-squares, and so on. Ross Eckler was not guilty of exaggeration when he called the challenge of this approach "trivial".

I knew that my vocabulary was fairly comprehensive, containing as it did perhaps all the headwords in the big dictionaries, plus many derived forms; specialised medical, legal, and zoological terms; encyclopaedia entries; British place names; and many obsolete words. I had twice tried to add alleged headwords from very large dictionaries: in one case, I found one new word, and in the other just 8 words, having eliminated duplicates and typing errors. What I probably lacked were some derived forms of modern words, and conjugations and declensions of Anglo-Saxon words. I considered foreign dictionaries, although they would be very much smaller than my English stock, but that would break my usual rule of including only words that could normally appear in an English language text (so that allows many foreign words, but not a whole dictionary).

For comparison, Dr Johnson's dictionary had 43,000 words. The OED is said to have had 414,825 words in 1928, and 615,000 today. Alternatively, it has 250,000 distinct English words, which, with inflections, technical terms, and regional words, and including all senses (meanings) makes 750,000. These various figures were all given by OED staff! The Shorter OED has 500,000 definitions. My belief is that the OED has around 400k headwords, with around a million more words appearing in the text or as variants etc. In sharp contrast to this ten-square situation, note that the OED alone is sufficient to produce excellent nine-squares (Nine-Square Round-Up, WW August 2003).

I then added US place names, telecommunications, textile, and computer specialised terms, plant synonyms, the whole of the Bible and Shakespeare's works, and very large numbers of genus and species names. This brought the total to over 2M words, of which 249772 had ten letters. For this, Chris Long's formula gave a disappointing estimate of 1.01 10-squares, but there were none, although there were 8237 cases where 9 rows were complete. The program was now taking over two weeks: this vocabulary, 3.8 times the size of the previous, was taking 800 times as long.

I then added place names from a number of countries, including the same names in two languages (eg Irish and English, Polish and German), a complete pharmacopoeia, and (reluctantly) new personal names (mostly family names), bringing the total to 258863 out of 2.2M.

Because of the sensitivity of the support level to the value of *p*, I asked Chris if I could run his support program against my data, thus getting a more relevant estimate: the program collects the frequencies of letters in each position of the words, then multiplies the frequencies of a given letter together. The results were telling (not to mention disheartening): at 258681 words, the program predicted only 0.29 squares against an estimate from the formula of 1.54. Moreover, the program now predicted a support level of 292961. Indeed, I am summarising a more lengthy process, in which the vocabulary continually exceeded the support level given by his program, only to see the bar raised higher. The formula originally wanted about 250k: when I had 258k, the program wanted 293k: next it wanted 319k!!.

It was now clear that even a new single-volume dictionary of entirely new words would probably not suffice. Extra words would have to come from words not in dictionaries. I needed proper nouns additional to the many I already had (or multiple foreign dictionaries). The UK has 18k easily accessible place names (and 50k in a chargeable form), so the world might have 5000/60 times 18k = 1.8 million plus, say a few million (though many duplicate each other and ordinary English words). For comparison, my 1922 Times Index-Gazeteer has about 128,000, including duplicates. Fortunately, I found a very large source of world place names. This caused a huge increase in vocabulary, bringing the total to 692847 out of 6.2M. I therefore did not pursue chemical names or biological names (there are millions of species, though not millions of names for them); nor did I pursue personal names, except for the most common. The formula now predicted 29296 squares: there were actually 2065. That short statement ignores the problem I had to find them all! Note also that there were 434k 10-letter words which were place names: that should be sufficient for a few word squares consisting solely of place names. It was: two examples are to be found in W2003-79.

In May 1992, August 1993, and November 1993, Ross Eckler reported on work with Leonard Gordon concerning calculation of support levels from the vocabulary needed to produce a number of squares. The work is far too extensive to summarise here, but having produced a number of ten-squares, the scaling formula offered can be used thus:

I got 2065 squares from 692847 words
If f be the fraction of the vocabulary to produce one square, then $2065 \times (1/f)^{10} = 1$
so $1/f$ = 10th root of $(1/2065) = 0.466131$
$0.466131 \times 692847 = 322957$ = support, say 323k.

This means that, if we had numerous sets of 323k words, the number of squares we would find would average one per set. Compare this with the last prediction of the program (too low) of 319k, and the formula prediction of 248k. Of course, to find the support level for one square by this means, you first have to produce a significant number, which was the problem in the first place! The same calculation using 176 squares from 441767 place name words gives a support level of 263k.

Although Chris Long's program behaved like pieces of chocolate with 75% cocoa solids dangled forever just out of reach, the distance did decrease, and gave some encouragement to continue the hunt despite the continual frustration.

Progress towards the Goal

When I had 250,000 words, my program was taking a fortnight or so — 800 times as long as it did for 67,000 words. It was fast (less than 3 days) when handling squares that had to be abandoned after a word or two, but 100 times or more slower when the square failed at the ninth or tenth word. Although it found no 10-squares, there were 8237 cases where 9 rows were complete. I had an estimate of 37 years for the number of words I thought might be needed. Here are some examples of failure at the tenth word, but completed manually (O means OED):

No s t o c a c e a	part of the Oscillatoria*
o u t r e a s o n s	O
s t e e r l i n g s	O
t r e e- l i n t i e	(chaffinch), O tree 10, 1844q
o'e r l i s t e n s	fabricated, akin to overlisten, O
Ca l i s t e n e s	pupil of Plato, O scoleye, 1434q
a s i n t e n d e d	as intended, Bloomsbury Thesaurus
c o n t e n d e r e	O nolo contendere
en g i n e e r e d	O
a s s e s s d e d e	assess dede (obsolete dead or death, O)

*www.scipress.org/journals/forma/pdf/1401/14010035.pdf

We can replace the last two rows by:

E n g i n e e r s' s	Whittaker's Electr. Engineers's Pocket-Bk. O reactive 6a 1920q
a s s e s s d e s k	assess desk, et al

This one is single-source:

u o r e~ s p e c h e	O forespeech, 1340q
o v e r t o i l e d	O overtoil
r e c h a r g i n g	O recharge
e r h a l t e n d e	German, O mneme, R. Semon *Die Mneme als erhaltende Prinzip**
s t e l l a n g e r	O stellenger, 1597q
p o r t a n t i n a	= sedan chair, O
e i g e n t o n e s	O
c l i n g i n e s s	O
h e n d e n e s s e	O hendness, 1393q
e d g e r a s s e s	edge rasses = peaks, O
	im Wechsel des organischen Geschehens (1904)

At about this time, I exchanged some information with Steve Root. He was taking the most popular 5-letter forenames and family names from the US Census to produce squares with 10-letter names: a very fast way to millions of 10-letter entities! I told him of the predictions of Chris Long's formula for 333933 and 401040 names, as a result of which he increased his word list to 500366 (giving 84 squares) and beyond. His program ran very fast, which might be understood in terms of the very restricted numbers of 5-letter groups, either at the start or at the end (about 1000 5-letter groups at the start of words, compared to 186,000 in my case). His program run time also increased rapidly with the number of words — 20 hours for 500k, to 59 hours for 700k. Ross Eckler checked the resulting squares against real names before publication in W2002-10 and -133. The program was certainly specific to this task: Steve tried a slightly changed program on my 258856 words, and what was expected to take three hours was projected to take three months. There seemed to be a flurry of activity at about this time: Jeff Grant and Ted Clarke were also active.

To get more speed, I changed my program by indexing down to the fifth level, rather than the fourth, removed code which was instrumenting what happened, and did some other optimisation. It ran much faster, though I was still expecting some years. The indexing method could not be taken further: the next step would have taken over 1GB for the index.

Therefore, in parallel with gathering more words and running the program, I did some experiments with randomising. By translating, say, the first six or seven letters of a word, randomising the result would give, almost all the time, the position in the alphabetic word list for the first word beginning with those letters. To check the first three letters, one simply randomises those letters with As appended to them to make six or seven letters. The results were promising, but at that time, Chris Long offered a program using a DAWG. (He was working on 10-squares containing words misspelled by a single adjacent-letter transposition!) I tried this, and it worked quite fast, but was very slow at times, and threatened to destroy my hard disk by virtue of intensive use at the start of each new top word. The diagnosis was simple: his program wanted more than the 256MB I had, so Windows was swapping memory contents in and out. At this stage, my program was chugging away on the full vocabulary of 692,000 10-letter words, and had produced a few squares, some good, like the one published as My First Ten-square (WW Aug 2002, "the first such square with all authenticated sources" according to www.hyperorg.com/blogger/mtarchive/000238.html). It was somewhat inconvenient to run my program on my machine, which was heavily used for some important matters, and impossible to run Chris's program with the other work.

Thus I came, with great agony, to a decision: I would buy a new machine. The new machine had 512MB, enough for Chris's program, and was also 3.8 times as fast. I soon had the DAWG program running. For a few weeks, I continued to run one program on each machine, on different parts of the word list. Then I ceased to run my program, which had found some good squares, and let Chris's program continue for almost a year until it finished (even though his program had the advantage of using lower-case letters — they are so much smaller and lighter to move than my capitals!). As luck would have it, the many squares it produced never matched the early ones in quality — note that the first words of the best squares, published in Nov 2002, all begin with D. So I need never have spent my money on hardware; although in that case the thought there might somewhere exist an absolutely perfect square would still be haunting me. Chris's program did, however, find very respectable squares for all the remaining letters of the alphabet, including the unpromising letters F, J, and Q - see *An A to Z of Ten-Squares*, Nov 2003.

For comparison, a complete run for 8-squares (with words from the same vocabulary) would actually take many years, but that is because it produces typically one to five million squares per hour, resulting in one trillion squares. The time is mainly spent at the lower levels.

Of the 2065 10-squares, 487 were quarter squares, 190 squares had exactly 9 tautonyms, and 176 squares contained purely place names. The most common top row words were ALANGALANG and INGITINGIT (34 times each): the most common non-tautonymic top row words were ENCALENDAR, and INCALENDAR (21 times each, essentially the same OED headword).

There is some interesting reading on 10-square work at http://www.gtoal.com/wordgames/wordsquares.html, and http://www.gtoal.com/wordgames/wordsquare/ (click on README).

Valuing squares
Having finally got so many squares, it was necessary to use a crude automated measure of quality to select the best. People place different values on characteristics of words: though all agree that a solid word is better than a hyphenated word, which in turn is superior to a phrase, what if the phrase is everyday and the solid word was last used as a variant spelling in 1282? However, manual selection from thousands of squares is almost impossible, so I have used a scheme to take account of various factors in order to present a smaller list of squares for serious consideration. Different people may have chosen differently, and there are factors not taken into account, such as single sourcing of all the words, diagonal and broken diagonal words, and so on. This scheme approximately reflects the difficulty of finding the various entities.

Start with 5 points for a solid word from a reputable dictionary. Award 4 points if the word is a proper name, or hyphenated, or a variant spelling, or is not a main entry (eg occurs in a quotation). Award 3 points for a phrase, and 2 for a hyphenated phrase. This gives for a maximum of 50 points for a ten-square. Subtract 1 for each repeated word.

These conditions normally sets a maximum of 45 for tautonymic squares, and for squares consisting entirely of solid place names. On this basis, a score of 40 or more is a good ten-square. Of course there are problems: for example a word may be hyphenated or solid depending on the source. Foreign words or phrases may be very acceptable: for example *sine qua non* probably should be treated as an English phrase. Proper names which are also phrases, such as those in Ross Eckler and Steve Root's collection (WW May 2002), should probably earn 3 points each, for a maximum of 30; but the matter does not arise for me, as any such words in my list are of famous people, which partly mitigates the penalty. However, the approach is only a first pass at selecting quality squares.

Among many squares with a score of 44 or more, the top score was 48 points out of 50, so a perfect square remains to be discovered; but then the first perfect nine-square was only published in August 2003 (*Nine-Square Round-Up*). The previously best square, by Jeff Grant, I would rate at about 42.

Conclusion

In principle, finding a 10-square is simple: find enough words, and use a quick method of searching. This article has attempted to explain why such a simple task has taken so long. The results of my work have been published in five articles in *Word Ways* from August 2002 to November 2003. It seems unlikely that there will ever be a perfect English ten-square containing no proper nouns or hyphens. That I was able to find good examples in 2002/3 was due to the existence of words on the Internet, and the recent availability of fast, large, home computers. I actually now have enough new words that perhaps I could make a square without place names, but a major improvement is unlikely.

The helpfulness of Chris Long and Steve Root will have been evident, as will the recent references provided by Graham Toal.

Eleven squares?

When I found the 10-squares, I had 586,000 11-letter words (now over 590k). For 11-squares, the formula suggests about 1M 11-letter words, which probably implies over 11M words; and the program suggests 1.27 million. This may be possible, as I understand Google has about 14M search terms, though many are misspellings. Finding acceptable sources for them might be a problem. Another approach is to add many foreign languages to my list. I calculate that might work: it certainly should if personal names are added. The squares, however, would reflect this mélange. Clearly finding a perfect square would be extremely lucky. Note that the tautonymic trick will not work on words with an odd number of letters, though a near miss is a square starting:

a	b	c	d	e	X	a	b	c	d	e
b	f	g	h	i	Y	b	f	g	h	i
c	g	j	k	l	Z	c	g	j	k	l

Words of this form are quite rare: I have but 80. (The support level for a 5-square is about 250, but you also need XYZ... to make a word.)

Twelve-squares?

Sadly, my 400 plus tautonyms yielded no squares. Indeed, no square was found merely starting with a tautonym. This is predictable, as Long's formula gives 992 as the support level for 6x6 squares, and a chance of 1% with my number of tautonyms. The 3.9 million words required by the formula can surely only be obtained by combining words of lesser length, as with names.

Foreign 10-squares

When finalising this article on 13th March 2004, Graham Toal sent me, hot off the press, two ten squares that do not use place names — but no English words either! He took 10-letter words, to a total of 592361 from Scrabble-like lists, in about 20 languages. There are many more squares to come. The first square uses Dutch, Spanish, Czech, Norwegian, Spanish, Romanian, French, Spanish, French, and Finnish in that order: the second is similar, but includes Italian. They both begin "aa-".