# Spellchecking by computer

---

Roger Mitton is a lecturer in Computer Science at Birkbeck College, University of London. His research interest is the development of a computer spellchecker for people whose spelling is poor. This article, which is a review of the methods that have been applied in computer spellchecking, is based on a chapter from his book *English Spelling and the Computer*, published by Longman, 1996, and now available online at Birkbeck ePrints. The article was first published in the *Journal of the Simplified Spelling Society*, Vol 20, No 1, 1996, pp 4-11. It is reproduced here by kind permission of the editor.

---

By the standards of the computer industry, spelling correction has a long history; people have been writing programs to detect and correct spelling errors for over thirty years. Reviews of the literature are provided by Peterson (1980a, 1980b) and Kukich (1992a). In this article I sketch the main methods and some of the unsolved problems. I will simplify the descriptions so as not to get bogged down in the detail. First I discuss methods for detecting errors; then I discuss methods for suggesting corrections.

## 1. Detecting errors

The most popular method of detecting errors in a text is simply to look up every word in a dictionary; any words that are not there are taken to be errors. But before I describe variations on this method, I will mention two that do not use a dictionary in this way.

### 1.1 Spellcheckers without dictionaries

The first uses a dictionary indirectly (Riseman and Hanson 1974). It

begins by going right through the dictionary and tabulating all the trigrams (three-letter sequences) that occur; *abs*, for instance, will occur quite often (*absent*, *crabs*) whereas *pkx* won't occur at all. Armed with this table, the spelling checker divides up the text into trigrams and looks them up in the table; if it comes across a trigram that never occurred in the dictionary, the word that contains it must be a misspelling. It would detect *pkxie*, for example, which might have been mistyped for *pixie*.*1 For detecting people's misspellings, this technique is of limited value since a high proportion of errors do not contain any impossible trigrams, but it is of some use in detecting errors in the output of an optical character reader (a machine that scans a page of text and `reads' the letters).

The second does not use a dictionary at all (Morris and Cherry 1975). Like the previous method, it divides the text into trigrams, but it creates a table of these, noting how often each one occurs in this particular piece of text. It then goes through the text again calculating an index of peculiarity for each word on the basis of the trigrams it contains. Given *pkxie*, for instance, it would probably find that this was the only word in the text containing *pkx* and *kxi* (and possibly *xie* too), so it would rate it highly peculiar. The word *fairy*, by contrast, would get a low rating since *fai*, *air* and *iry* probably all occur elsewhere, perhaps quite often, in the passage being analysed. Having completed its analysis, it draws the user's attention to any words with a high peculiarity index. Like the previous method, it would fail to spot a high proportion of ordinary spelling errors, but it is quite good at spotting typing errors, which is what it was designed for. An advantage that it has over all dictionary-based methods is that it is not tied to English; it will work on passages of, say, French, German or Greek.

## 1.2 Saving space

The majority of spelling checkers, however, use a dictionary in some way. I say `in some way' because they do not necessarily hold a complete dictionary with all the words spelt out in full, though some

do. Some economize on storage space by holding only the stems of words (McIlroy 1982). For example, instead of holding *doubt*, *doubts*, *doubted* and *doubting*, they hold just *doubt* and use a set of rules to remove suffixes before looking words up; given *doubting*, the checker would remove the *ing* and look up the *doubt*. They may remove prefixes also (*undoubted*) and they may carry on removing suffixes (or prefixes) until they reach a stem (*undoubtedly*). The process is known as `affix-stripping'.

The rules have to be a bit more complicated than this in order to cope with such forms as *cutting* (to get *cut* rather than *cutt*), and *denied* (to get *deny* rather than *deni*). The rules have to have some ordering, so as to accept *undoubtedly* but not *undoubtlyed*, and they need to have some way of coping with words that look like inflected forms but aren't, such as *farthing*. The strength of this system is that the checker can accept freshly minted words that are acceptable but are possibly not in any dictionary, such as *unplaceable*. The weakness is that it will accept some words that don't exist, such as *undoubt*.

A second way to save storage space is to hold the dictionary as a bit map (McIlroy 1982, Nix 1981). Imagine the memory of a computer as a long row of lightbulbs, initially all switched off. You go through the dictionary and convert each word, somehow, into a number. For example, you might start by converting *a* to 1, *b* to 2, *c* to 3, and so on; the word *ace*, for example, would become 1,3,5. Then multiply the first number by 1, the second by 2 and so on, and add them up; 1,3,5 gives (1×1)+(3×2)+(5×3) = 22. Finally, multiply by 10 and add the number of letters in the word: (22×10) + 3 = 223. Now you go to the 223rd lightbulb and switch it on. After you've done this for every word in the dictionary, some of the lightbulbs are on and the rest are still off.

Now you are ready to do a spelling check. You take each word of the text and convert it to a number by the same process you used before; if you came across the word *ace*, you'd convert it to 223. You look at the appropriate lightbulb. If it's on, the word is acceptable; if it's off, it

isn't. So, *ace* (lightbulb 223) is accepted. *Ade,* by contrast, would be converted to 243; the 243rd lightbulb would be off, so *ade* would be rejected.

The long line of lightbulbs is the `bit map', an array of thousands of binary digits (0's and 1's). Converting a word to a number is known as hashing, and the method you use is a hashing function. The hashing function described above is too simple to do the job properly - *dcd, hdb* and various other non-words would all hash to 223 and be accepted - but it's possible to devise more complicated hashing functions so that hardly any non-words will be accepted. You may use more than one hashing function; you could derive, say, six numbers from the same word and check them all in the bit map (or in six separate bit maps), accepting the word only if all six bits were set.

If there is no need to save storage space, the dictionary can be held as a straightforward list of words, inflected forms included. The computer looks a word up in much the same way as a person looks one up in a printed dictionary. The words can be stored in alphabetical order, so the computer can go straight to the right place and check if it's there or not.

## 1.3 Real-word errors

There are two ways in which a spelling checker can fail: it may flag a word as an error when in fact it's correct, or it may let an error slip through. Obscure words and proper names are the cause of the first problem. The frequency of these false alarms can be reduced by having a larger dictionary or a specialized one for particular kinds of text, such as a dictionary with lots of medical terms, but there is no real solution to the problem of proper names - there are just too many of them. Many checkers have a facility whereby the user can build up a private supplement to the dictionary, to prevent the checker from constantly flagging names that crop up often in the user's documents.

False alarms, though irritating, may be acceptable in moderation since

the user can always ignore the checker's output, but the second problem - letting errors slip through - is more worrying since the user cannot be sure that a passage is error-free even when the checker has gone over it. The problem arises because some misspellings match words in the dictionary, as in `*Their* she goes,' `The *wether* was glorious,' or `The Continental restaurant company is developing a chain of French-style *brassieres.*'*2 I call these `real-word errors'.

Unfortunately the problem gets worse as the dictionary gets larger; including more obscure words in the dictionary, to reduce the number of false alarms, increases the risk of missing real-word errors. The word *wether* illustrates this. The word is, arguably, so obscure that any occurrence of *wether* in a passage is more likely to be a misspelling of *weather* or *whether* than a genuine occurrence of *wether,* so a checker that did not have the word in its dictionary would do better than one that did.

Drastic pruning of the dictionary, however, is not a solution; a checker with a small dictionary raises too many false alarms. A recent study has shown that, when an uncommon word occurs, it is far more likely to be a correct spelling of a rare word than a misspelling of some other word (Damerau and Mays 1989). This may not be true of some highly obscure words that resemble common words, such as *yor* and *stong*, so perhaps some judicious pruning is advisable. Nor is it true of certain medium-rare words that occur commonly as misspellings of other words, such as *cant* and *wont* which are often misspellings of *can't* and *won't;* these seem to require special treatment. But, with these provisos, big is beautiful for a checker's dictionary.

How serious is the problem of real-word errors? At first sight, it appears to be only marginal; the proportion of words that can be changed into another word by a small typing slip, such as *whether* into *wether*, is only about half of one per cent. However, the proportion is far higher among short, common words than among long, rare ones. Mistyping *sat*, for instance, is quite likely to produce

another word (*set, sit, sad* and so on), whereas mistyping *antirrhinum* is not. Taking this into account, the proportion of all typing errors that produce other words may be as high as sixteen per cent (Peterson 1986).

When spelling errors, as well as typing errors, are included, the problem becomes much more alarming. In a corpus of about four thousand errors taken from the free writing of secondary-school pupils of school-leaving age, forty per cent were real-word errors (Mitton 1987). In some cases the misspelling was based on pronunciation, and it was only by chance that it matched a dictionary word, such as *tort* for *taught*, but, more often, the misspelling was some other familiar word, as if the person writing it had two known spellings in mind and chose the wrong one. The wrong one was often a homophone of the right one, but not always. Errors of this kind were particularly likely to occur on function words (words like *of, and, be* and so on); in eighty per cent of the misspelt function words, the error consisted in writing some other function word in place of the one intended, such as `*He* name was Mrs Williams,' and `You *we* treated like babies.'

## 1.4 Detecting grammatical anomalies

Very few spelling checkers make any attempt to detect real-word errors, but at least three research projects have tried to tackle the problem. The first is a system called CRITIQUE (previously called EPISTLE) developed by IBM (Heidorn et al. 1982). This is a piece of software that will check the spelling, grammar and style of business correspondence. Armed with a complicated set of grammar rules for English, it attempts to parse each sentence of the text, i.e. to analyse a sentence into its syntactic parts - Noun (Subject of sentence), Adjective (qualifying Subject), Main Verb, Prepositional clause, and so on. If it fails, because the sentence is grammatically incorrect, it tries again, this time relaxing some of its grammar rules, and it carries on doing this until it achieves a successful parse. Since it knows which rule or rules it had to relax, it can work out what was

grammatically wrong with the sentence. A real-word error quite often produces a grammatically incorrect sentence (such as `I might of done'), so CRITIQUE can detect such errors and can sometimes suggest a correction, since the syntactic context gives a lot of clues to what the word should have been.*3

The second project also depends on syntax as a way of spotting real-word errors. It is a modification of a system, developed at Lancaster University, for tagging words in a text with their parts-of-speech (Marshall 1983, Garside et al. 1987). Given a sentence such as `The fly bit the goat,' it first consults a dictionary to find out which tags (parts-of-speech) each of the words can have; it will find that *the* is a definite article, and that *fly* (likewise *bit*) can be a noun or a verb. It also has a table, derived from a large corpus of English text, showing the probability of a given tag being followed by another in a sentence; the table will show, for example, that a definite article is very likely to be followed by a noun, but not likely to be followed by a verb. It then works out, purely on the basis of probability, that *fly bit* in this sentence is likely to be Noun-Verb, rather than Verb-Noun (or Noun-Noun or Verb-Verb).

The system can be applied to looking for real-word errors by modifying it to report when it finds a sequence of tags that is very unlikely. For example, it would query `Please complete the *from* in capitals,' since the sequence *the from in* (Definite article, Preposition, Preposition) has only a low probability (Atwell 1983, Garside et al. 1987).

Both these systems have some success in spotting real-word errors, but both tend to give too many false alarms because of sentences which are grammatically out of the ordinary but not ungrammatical (Richardson 1985, Leech et al. 1986). Neither, of course, can do anything about real-word errors that are not syntactically anomalous, such as `We had thirty *minuets* for lunch,' `We used to *pant* on Thursdays and hang up the *pitchers* on the walls,' `There was a *fate* every summer.'

## 1.5 Detecting unlikely combinations of words

The third assault on real-word errors, again by researchers at IBM, resembles the Lancaster work somewhat in using probabilities derived from a very large corpus of text, but the probabilities are not of the co-occurrence of tags but of the co-occurrence of actual words (Mays et al. 1991). Given any two words from their 20,000-word dictionary, they can say what the probability is of any other of their dictionary words occurring next. Given, for instance, `I think' as the first two words, they could say what the probability was of the word *that* occurring next. Or of the word *slowly* or *big* or *therefore* or *teapot*. (Presumably the probability of *that* after `I think' is relatively high whereas the probability of *teapot* after `I think' must be close to zero.)

In an experiment, they took sentences containing a single real-word error, such as `The thief licked the lock,' (for *picked*). The misspellings were all of the simple typing-slip kind, i.e. differing by just one mistype from the correct spelling. (I explain below what I mean by that.) When considering a word as a possible error, the system first generated all the words that might have been changed into this word through a typing slip. For example, from *licked* it would have generated *kicked, ticked, locked, liked* and so on, including *picked*. For each of these alternatives, it calculated the probability of the whole sentence from its table of three-word probabilities, i.e. one value for `The thief *kicked* the lock,' another for `The thief *ticked* the lock,' and so on. It also calculated the probability of the original sentence, `The thief *licked* the lock.' If `The thief *picked* the lock' came out as more probable than `The thief *licked* the lock,' it would conclude that *licked* was a real-word error that should be corrected to *picked*.

It could be wrong either by leaving the original error uncorrected or by preferring the wrong alternative or by `correcting' some other word in the sentence. It had no way of knowing in advance that *licked* was the misspelling here. It would go through the same procedure with all the other words. It would generate *dock, rock,*

*sock, look* and so on for *lock* and might possibly prefer `The thief licked the *rock*.'

There was a further factor in its calculations, namely a general level of expectation of errors in the text. This was set by the experimenters at levels between 0.1 and 0.0001. Essentially, if it was told to expect a lot of errors, it tended to make a lot of corrections, i.e. to rate the alternatives as more probable than the original, though many of its `corrections' were wrong. If it was told that errors were rare, it was more respectful of the original text; when it did make a correction, it was nearly always right, but it left a lot of the misspellings uncorrected.

It is not clear what application this method could have to ordinary spelling checkers in the near future because of its considerable demands on memory and computing power, but it is the only method I know of that has been capable of detecting (and correcting) syntactically acceptable real-word errors in unrestricted text.

## 2. Spelling correction

Many people find that a spelling checker is all they need; they know how to spell and they just want their occasional slips to be pointed out to them. People who have trouble with spelling, however, need something more. Suppose you have written *neumonia* and the checker has told you this is wrong. If you don't know how to spell *pneumonia*, you're stuck. The dictionary is no help. You want the computer to tell you the correct spelling.

To correct someone's spelling errors, you have to be able to guess what words the person meant and you have to be able to spell them correctly. People generally find the first part easy but the second part harder; most people would understand `She was excused swimming because of her verouka,' but they would not be able to correct it. For computers, it's the other way round. Producing a correct spelling is easy - they can store a complete dictionary and retrieve any word as

required; the hard part is deciding which word was intended.

It is for this reason, incidentally, that one cannot say in general whether computers are better or worse than people at spelling correction. Given a minor misspelling of a long word, such as *innoculation,* a computer will detect it and correct it better than most people would, because it is easy to guess what word was intended but not easy to spell it. By contrast, with a misspelling of a common word, such as *cort* (`We got cort in the rain'), a computer might have difficulty deciding that *caught* was the word intended, whereas most people would correct it easily.

Given a dictionary of realistic size - say 30,000 to 80,000 words - it is not practical to go through the entire dictionary for each misspelling, considering every word as a possible candidate; a corrector has to select a section of the dictionary, of some tens or hundreds of words, and search through these in the hope of finding the correct word.

## 2.1 Simple errors

Analyses of errors - mainly typing errors - in very large text files (Damerau 1964, Pollock and Zamora 1984) have found that the great majority of wrong spellings (eighty per cent to ninety-five per cent) differ from the correct spellings in just one of the following four ways:

```
one letter wrong (peaple)
one letter omitted (peple)
one letter inserted (peopple)
two adjacent letters transposed (pepole)
```

It has also been found (Yannakoudakis and Fawthrop 1983a) that the first letter is usually correct. Given a mistyped word, therefore, there is a good chance that the correct spelling will begin with the same letter and will be either the same length or just one letter longer or shorter. If the words are held in order of first letter and length, it is easy for the corrector to restrict its search to the appropriate section of

the dictionary (Turba 1982).

Words that are misspelt, as opposed to mistyped, tend to differ from the correct spellings in more than just the simple ways listed above (Mitton 1987). For example, *disapont* - a misspelling of *disappoint* - is two letters shorter than the correct word; looking through the dictionary at words beginning with *d* and of seven to nine letters long would fail to find *disappoint.* You could simply increase the number of words to be considered, perhaps taking in words that are two letters longer or shorter than the misspelling, but this would increase substantially the number of words the corrector had to look at, so it would take longer to produce its correction. It would also be inefficient since a large proportion of the words it looked at would be nothing like the misspelling; for *disapont,* it would take in *donkey* and *diabolical,* which are obviously not what *disapont* was meant to be. What is needed is some way of retrieving those words that have some resemblance to the misspelling.

## 2.2 Soundex

This problem has been around for a long time in the context of retrieving names from a list of names. Suppose you are working at an enquiry desk of a large organization, with a terminal connecting your office to the central computer. A customer comes in with a query about her account. She says her name is *Zbygniewski.* You don't want to ask her to spell it - perhaps her English is poor and other customers are waiting. To make matters worse, the name may be misspelt in the computer file. You want to be able to key in something that sounds like what she just said and have the system find a name that resembles it.

The Soundex system was devised to help with this problem (Knuth 1973, Davidson 1962). It dates, in fact, from the days of card-indexes - the name stands for `Indexing on sound' - but has been transferred to computer systems. A Soundex code is created for every name in the file. The idea of the code is to preserve, in a rough-and-ready

way, the salient features of the pronunciation. Vowel letters are discarded and consonant letters are grouped if they are likely to be substituted for each other - an *s* may be written for a *c*, for instance, but an *x* for an *m* is unlikely. The details are presented in Figure 1, with some examples.

```
1) Keep the first letter (in upper case).
2) Replace these letters with hyphens: a,e,i,o,u,y,h,w.
3) Replace the other letters by numbers as follows:
     b,f,p,v : 1
     c,g,j,k,q,s,x,z : 2
     d,t : 3
     l : 4
     m,n : 5
     r : 6
4) Delete adjacent repeats of a number.
5) Delete the hyphens.
6) Keep the first three numbers or pad out with zeros.

For example:

   Birkbeck   Zbygniewski   toy    car    lorry   bicycle

   B-621-22   Z1-25---22-   T--    C-6    L-66-   B-2-24-

   B-621-2    Z1-25---2-    T--    C-6    L-6-    B-2-24-

   B621       Z125          T000   C600   L600    B224
```

Figure 1 The Soundex code, with some examples

So, every name in the file has one of these codes associated with it. The name *Zbygniewski* has code Z125, meaning that it starts with a *Z*, then has a consonant in group 1 (the *b*), then one in group 2 (the *g*) and then one in group 5 (the *n*), the remainder being ignored. Let's say you key in *Zbignyefsky*. The computer works out the Soundex code for this and retrieves the account details of a customer with the same code - *Zbygniewski* - or perhaps the accounts of several customers with somewhat similar names.

It is fairly obvious how this system can be applied to spelling correction. Every word in the dictionary is given a Soundex code. A Soundex code is computed from the misspelling, and those words that have the same code are retrieved from the dictionary. Take as an example the misspelling *disapont.* A corrector would compute the code D215 from *disapont* and then retrieve all the words with code D215: *disband, disbands, disbanded, disbanding, disbandment, disbandments, dispense, dispenses, dispensed, dispensing, dispenser, dispensers, dispensary, dispensaries, dispensable, dispensation, dispensations, deceiving, deceivingly, despondent, despondency, despondently, disobeying, disappoint, disappoints, disappointed, disappointing, disappointedly, disappointingly, disappointment, disappointments, disavowing.*

## 2.3 The SPEEDCOP system

The purpose of the SPEEDCOP project was to devise a way of automatically correcting spelling errors - predominantly typing errors - in a very large database of scientific abstracts (Pollock and Zamora 1984). A key was computed for each word in the dictionary. This consisted of the first letter, followed by the consonant letters of the word, in the order of their occurrence in the word, followed by the vowel letters, also in the order of their occurrence, with each letter recorded only once; for example, the word *xenon* would produce the key *XNEO* and *inoculation* would produce *INCLTOUA.* The words in the dictionary were held in key order, as illustrated by the small section shown in Figure 2.

```
PLTDOE      plotted
PLTE        pellet
PLTEI       pelite
PLTIO       pilot
PLTNGAI     plating
PLTNSUO     plutons
PLTNUO      pluton
PLTOU       poult
```

Figure 2 A section of the SPEEDCOP dictionary

When the system was given a misspelling, such as *platin,* it computed the key of the misspelling and found its place in the dictionary. In this example, the key of *platin* would be *PLTNAI*, which would come between *PLTIO* and *PLTNGAI*. Moving alternately forwards and backwards from that point, it compared the misspelling with each of the words to see if the misspelling could be a single-error variation on that word, until either it had found a possible correction or had moved more than fifty words away from its starting point. The SPEEDCOP researchers found that, if the required word was in the dictionary, it was generally within a few words of the starting point. In the example, the corrector would quickly find the word *plating* as a possible correction (*platin* being an omission-error variant of *plating*).

The Soundex code and the SPEEDCOP key are ways of reducing to a manageable size the portion of the dictionary that has to be considered. Confining the search to words of the same length (plus or minus one) restricts the search even further. The price to be paid is that, if the required word is outside the set of those considered, the corrector is not going to find it.*4

## 2.4 String-matching

The next task facing the corrector is to make a selection from the words it looks at - a best guess, or at least a shortlist. If the corrector is intended mainly to handle typing errors, this task is not difficult. Given that the great majority of mistyped words fall into one of the four classes listed above, the corrector compares the misspelling with each candidate word from the dictionary to see if they differ in one of these four ways. If they do, then that candidate joins the shortlist. Given the misspelling *brun,* for instance, the corrector would produce the list *brunt* (omitting one letter gives *brun*), *bran* (changing one letter), *bun* (inserting one letter) and *burn* (transposing adjacent letters).

Another way of selecting candidates is to calculate, in some way, how

closely each word resembles the misspelling and to shortlist those that have the best scores. This process is called `string-matching', and there are many ways of doing it. One way is to see how many chunks of the shorter string are present in the longer string (Joseph and Wong 1979). For instance, given *medsin* and *medicine,* you could say that *medsin* has the *med* and the *in* of *medicine,* a total of five letters out of the eight in *medicine,* a score of sixty-three per cent. Another method considers the number of trigrams (three-letter sequences) that the two strings have in common (Angell et al. 1983). *Medicine* and *medsin* would be divided up as follows (the # symbol marks the beginning or end of a word):

```
medicine      #me med edi dic ici cin ine ne#
medsin        #me med eds dsi sin in#
```

The more trigrams the two have in common, the better match they are considered to be. Some methods give more weight to letters near the front; others rate letters near the end more highly than those in the middle; some rate certain letters more highly than others, such as consonants over vowels.*5 Some hand-held spellcheckers make use of a special-purpose chip which implements string comparisons at high speed (Yianilos 1983).

## 2.5 Feature vectors

A project at Bellcore is investigating the use of spelling correction in an unusual setting, namely to assist deaf or speech-impaired people to use the telephone (Kukich 1992b). Deaf people can communicate with each other over a telephone line by using a screen and keyboard. When they want to converse with a user of a voice telephone, they go via a relay centre. The voice user speaks to the relay person who types the message to the deaf person; the deaf person types back and the relay person speaks it. Bellcore would like to automate this process and part of this involves the generation of computer speech from the keyed text. But this text typically contains typing errors which upset the speech generator, hence the need for spelling correction. The corrector is allowed to make only one correction for

each misspelling, not a list of possible corrections such as a spellchecker would produce.

Experiments have found that one of the simpler methods is the most effective. A `feature vector' of about five hundred bits (think of a line of lightbulbs again) is computed for each word in the dictionary. If the word contains an *a*, the first bit is set (the first lightbulb is turned on); if it contains a *b*, the second is set, and so on. If it contains *aa*, the 27th is set; if it contains *ab*, the 28th is set. (There is no place in the line for letter-pairs that don't occur in English, such as *yy*.) A corresponding feature vector is computed for the misspelling and this is compared with the vectors of the dictionary words. The word whose vector is most like the misspelling's vector (most nearly has its lightbulbs on and off in the same places) is chosen as the correction.

## Error probabilities

Some methods of string-matching make use of tables showing the likelihood of this or that letter being involved in an error. I describe one of these methods (Wagner and Fischer 1974) in more detail in my book. It was developed for correcting the output of an optical character reader. These machines are prone to make certain errors more than others; for example, they are likely to read an *e* as an *o*, but not likely to read a *t* as an *m*. The corrector has a table showing the probability of one letter being mistaken for another, and it uses these figures in deciding what the word ought to be. Given *gom*, it would guess that the word was *gem* rather than *got*.

Probability is also the basis of an approach developed at Bell Labs for correcting typing errors (Kernighan et al. 1990,Church and Gale 1991). This system has tables of error probabilities derived from a corpus of millions of words of typewritten text. The tables give the probability of an *a* being substituted for a *b*, a *p* being inserted after an *m*, and so on. It also has an estimate of the probability of any particular word occurring in the text.

When it detects a misspelling (which it does by dictionary look-up), it first retrieves from the dictionary all the words that could have given rise to this misspelling by a single mistype. (It doesn't handle more complicated errors.) For example, from the misspelling *acress,* it retrieves *actress, cress, caress, access, across* and *acres.* Taking *actress,* it consults its table for the probability of having a *t* omitted after a *c* and combines this with the probability of meeting the word *actress.* In this way it produces a probability estimate for each of the candidates and it then puts the candidates in order of probability for presentation to the user.

The errors that poor spellers make are more complicated than those of an optical character reader or a typist, but a similar approach can still be used. One system (Yannakoudakis and Fawthrop 1983b) has a table of error-patterns, derived from the analysis of a corpus of spelling errors; the table might show, for instance, that *au* is sometimes written as *or*, or *ch* as *tch*. It compares the misspelling with each of the words in the section of the dictionary that it's looking at to see if the difference follows the patterns in its table. For example, given *lorntch,* it would find that *launch* differs from it in two of these ways. The table also contains information about the frequency with which each of these error-patterns occurs, so the corrector can put the shortlisted candidates into order. When trying to correct *lorntch,* it would also find *lounge* but it would rate this as less likely than *launch* because the table contains the information that *or* for *ou* and *ge* for *ch* are less likely than *or* for *au* and *tch* for *ch.*

## 2.7 Phonetic similarity

Some of the more advanced commercial correctors also retrieve candidates on a `phonetic' basis. Their dictionaries presumably contain information about pronunciation, and the correctors use this to offer words that might sound like the misspelling, even though they don't look much like it; for *newmoanya,* for example, their list would include *pneumonia.*

Commercial companies tend not to publish details of how their spellcheckers work, but there is one pronunciation-based spellchecker described in the research literature; it was developed in the Netherlands for the correction of Dutch, though the principles would apply to English also (VanBerkel and DeSmedt 1988). It uses a variation on the trigram system mentioned earlier, but with pronunciations rather than spellings. Given the misspelling *indissceat*, for example, it would begin by making a guess at the pronunciation - perhaps /IndIski:t/ - then break this up into `triphones' and then compare this with the pronunciations of various words in its dictionary, also broken up into triphones. The comparison with *indiscreet* would look like this:

```
indissceat     #In Ind ndI dIs Isk ski: ki:t i:t#
indiscreet     #In Ind ndI dIs Isk skr kri: ri:t i:t#
```

The more triphones a dictionary word has in common with the misspelling, the better match it is considered to be.*6 Homophones, of course, match perfectly.

## 2.8 Ordering the list

Most correctors simply offer a small selection of possible corrections, generally about six, for the user to choose from, though some correctors offer dozens of suggestions if the user wants them. This shortlist, however, is often a curious rag-bag of words. When asked to make suggestions for *perpose,* Microsoft Word Version 6.0 produced the list (in this order) *preppies, propose, papoose, prepuce, preps* and *props,* but not *purpose.* The lists often contain obscure words with no indication of their level of obscurity; many of the offerings are wildly inappropriate for the context and perhaps not even syntactically possible. When asked for suggestions for *cort* in `I've cort a cold,' Wordperfect 5.1 produced - take a deep breath - *cart, cert, coat, colt, cont, coot, copt, cor, cord, core, corf, cork, corm, corn, corp, corr, cors, corti, cost, cot, court, crt, curt, carat, carate, card, cared, caret, carried, carrot, carte, cerate, cered, ceroid, chaired, charade, chard, chariot, charred, chart, cheered,*

*cheroot, chert, chirred, chord, choreoid, chorioid, choroid, cirrate, cored, corrade, corrode, corrupt, coward, cowered, curate, curd, cured, curet, curette, curried, karate, kart, keyword, scared, scarred, scirrhoid, sciuroid, scored, scoured* and *scurried,* but not, alas, *caught* (perhaps because *caught* and *cort* are not homophones in American speech). One can't help feeling that the corrector ought to be able to do better - to restrict its list to plausible suggestions and to order them so that its best guess is generally the one required. Given `You shud know,' it ought to offer *should* ahead of *shad* and *shed.*

Word frequency can help; *shad* could be removed from the above list, or at least relegated to the bottom, purely because of its rarity. But it doesn't help much; candidates in the shortlist are often of similar frequency, such as *there* and *their* for *ther,* and a rare word will occasionally be the one required.

Syntax can also help. I described earlier how some correctors do a syntactic analysis in order to spot real-word errors; they can use the same analysis to rule out some of the candidates. Quite often, as in *shad, shed, should,* there will be only one candidate left.

## 2.9 Using semantics

A semantic analysis is much more difficult for a computer to attempt, but it may be possible when the subject matter of the text is restricted (Morgan 1970, Teitelman 1972). For example, a corrector that checked the commands that people typed into an electronic mail system would be able to correct *Snd* to *Send* (rather than *Sand* or *Sound*) in `Snd message to Jim,' because *Send* is one of the few words that could occur at that point in this sentence (Durham et al. 1983). Similarly, a system that handled enquiries to British Rail would be able to use its interpretation of the meaning to correct `Is there an erlier conexson?' (Hendrix et al. 1978) A system of this kind might be able to detect some real-word errors. A computerized tourist guide might detect that a query about *gold courses* was really about *golf courses.* More ambitiously, a system that conducted a dialogue with a

user might be able to build up a representation of what the user had in mind and use this for spellchecking (Ramshaw 1994). If a user of the computerized tourist guide had been asking about holidaying in the west country and then asked `Are there trains to *Swinton*?' the system might guess that he meant *Swindon*, since Swindon is on the main line from London to the west whereas the places called *Swinton* are all in the north. In general, however, spellcheckers that handle unrestricted text do not have enough information about the words in their dictionaries or about the topics people write about to enable them to make any use of the semantic context.

## 2.10 The state of the art

At present, then, checkers and correctors play a small but useful role in helping people to remove minor errors from their written work. Some systems are just checkers - they flag errors but make no attempt to offer suggestions - and this is often all that is required; if you've typed *adn* for *and*, you can correct it easily. Most systems, however, do both checking and correcting, so that the word *spellchecker* usually means a piece of software that both checks the text and offers suggestions for misspelt words. A list of suggestions can occasionally be helpful, especially for people whose spelling is a little weak; not everyone would know, if a checker queried *occurence,* that it ought to be *occurrence.* But spellcheckers are still some way short of offering the help that a poor speller wants - the kind of job that a good typist would do.

They miss a fairly high proportion of errors; real-word errors form a substantial minority of spelling errors and most spellcheckers ignore them completely. Their suggestions are often irritatingly inappropriate, frequently including words that are obscure or syntactically out of place. If the misspelling differs from the correct word in certain ways, such as having a different first letter (*nowledge, wrankle, eny*), or being more than one letter longer or shorter (*probly, cort, pollitishion*), or having several letters different (*payshents, powertree, highdrawlick*), the required word may not be in the list of

suggestions at all.

## Notes

1. Riseman and Hanson's trigrams are actually more complicated than my short description suggests. The letters in a Riseman and Hanson trigram are not necessarily consecutive, and the position in the word is important. For example, from the word *trunk* they would store the information that all of the following trigrams were possible:

```
t-r-u at positions 1-2-3
t-r-n at positions 1-2-4
t-r-k at positions 1-2-5
t-u-n at positions 1-3-4
t-u-k at positions 1-3-5
t-n-k at positions 1-4-5
r-u-n at positions 2-3-4
r-u-k at positions 2-3-5
r-n-k at positions 2-4-5
u-n-k at positions 3-4-5
```

2. The *brassieres* (for *brasseries*) error was in a report quoted in a short piece about spellcheckers in The Times of 16 February 1995.
3. A similar system has been implemented in a language-sensitive text editor for Dutch (Kempen and Vosse 1992). It is capable, for example, of detecting the misspelling *word* in *Peter word bedankt* (English *Peter am thanked*) and correcting it to *Peter wordt bedankt* (*Peter is thanked*).
4. The SPEEDCOP researchers found that the most frequent cause of failure with their system was the omission of consonant letters near the beginning of a word (Pollock and Zamora 1984). For example, the misspelling *pating* would produce the key PTNGAI, which might be some distance away from PLTNGAI, the key of *plating*. They therefore computed a second key, called the `omission key'. They knew from their analysis of a large corpus of spelling errors that consonant letters were omitted in the following order of increasing

frequency - the letter *j* was omitted the least and the letter *r* the most:

```
J K Q X Z V W Y B F M G P D H C L N T S R
```

The omission key consisted of the consonant letters of the word sorted in this order, followed by the vowel letters in their order of occurrence in the word. The omission key for *pating* would be GPNTAI, which would probably be close to the omission key for *plating* - GPLNTAI.

5. Some of these variations are described in Hall and Dowling (1980), Alberga (1967), Blair (1960) and Cornew (1968).
6. For some misspellings it is possible that more than one variant pronunciation might be generated, though many details of the pronunciation, such as stress pattern, can be ignored since this application is less demanding than, say, speech generation. The dictionary also stores the frequency with which each triphone occurs and these frequency values are taken into account; if two pronunciations share an unusual triphone in common, this will be considered more significant than if they share a run-of-the-mill one.

---

# References


1. Alberga, Cyril N., "String similarity and misspellings," Communications of the A.C.M., vol. 10, no. 5, pp. 302-313, May 1967.
2. Angell, Richard C., Freund, George E, and Willett, Peter, "Automatic spelling correction using a trigram similarity measure," Information Processing and Management, vol. 19, no. 4, pp. 255-261, 1983.
3. Atwell, Eric, "Constituent-likelihood grammar," ICAME News: newsletter of the International Computer Archive of Modern English, no. 7, pp. 34-67, May 1983.
4. Blair, Charles R., "A program for correcting spelling errors," Information and Control, vol. 3, pp. 60-67, 1960.

5. Church, Kenneth W and Gale, William A, "Probability scoring for spelling correction," Statistics and Computing, vol. 1, pp. 93-103, 1991.
6. Cornew, Ronald W., "A statistical method of spelling correction," Information and Control, vol. 12, pp. 79-93, 1968.
7. Damerau, Fred J and Mays, Eric, "An examination of undetected typing errors," Information Processing and Management, vol. 25, no. 6, pp. 659-664, 1989.
8. Damerau, Fred J., "A technique for computer detection and correction of spelling errors," Communications of the A.C.M., vol. 7, pp. 171-176, March 1964.
9. Davidson, Leon, "Retrieval of misspelled names in an airlines passenger record system," Communications of the A.C.M., vol. 5, pp. 169-171, March 1962.
10. Durham, I., Lamb, D.A., and Saxe, J.B., "Spelling correction in user interfaces," Communications of the A.C.M., vol. 26, no. 10, pp. 764-773, October 1983.
11. Garside, Roger, Leech, Geoffrey, and Sampson, Geoffrey, *The Computational Analysis of English: a corpus-based approach*, Longman, 1987.
12. Hall, Patrick A.V. and Dowling, Geoff R., "Approximate string matching," Computing Surveys, vol. 12, no. 4, pp. 381-402, Dec 1980.
13. Heidorn, G.E., Jensen, K., Miller, L.A., Byrd, R.J., and Chodorow, M.S., "The EPISTLE text-critiquing system," IBM Systems Journal, vol. 21, no. 3, pp. 305-326, 1982.
14. Hendrix, Gary G., Sacerdoti, Earl D., Sagalowicz, Daniel, and Slocum, Jonathan, "Developing a natural language interface to complex data," A.C.M. Transactions on Database Systems, vol. 3, no. 2, pp. 105-147, June 1978.
15. Joseph, D.M. and Wong, Ruth L., "Correction of misspellings and typographical errors in a free-text medical English information storage and retrieval system," Methods of Information in Medicine, vol. 18, no. 4, pp. 228-234, 1979.
16. Kempen, Gerard and Vosse, Theo, "A language-sensitive text editor for Dutch," in *Computers and Writing* , ed. Noel

Williams, pp. 68-77, Intellect Books, 1992.
17. Kernighan, Mark D, Church, Kenneth W, and Gale, William A, "A spelling correction program based on a noisy channel model," in *COLING-90 13th International Conference on Computational Linguistics* , ed. Hans Karlgren, pp. 205-210, Helsinki University, 1990. Volume 2
18. Knuth, Donald E., *The Art of Computer Programming: Volume 3 Sorting and Searching*, Addison-Wesley, 1973.
19. Kukich, Karen, "Spelling correction for the Telecommunications Network for the Deaf," Communications of the A.C.M., vol. 35, no. 5, pp. 80-90, May 1992.
20. Kukich, Karen, "Techniques for automatically correcting words in text," Computing Surveys, vol. 24, no. 4, pp. 377-439, 1992.
21. Leech, G.N., Garside, R.G., and Elliott, S.J., *Development of a Context-sensitive Textual Error Detector and Corrector: Final project report submitted to International Computers Limited*, Unit for Computer Research on the English Language, Lancaster University, November 1986.
22. Marshall, Ian, "Choice of grammatical word-class without global syntactic analysis: tagging words in the LOB Corpus," Computers and the Humanities, vol. 17, pp. 139-50, 1983.
23. Mays, Eric, Damerau, Fred J, and Mercer, Robert L, "Context based spelling correction," Information Processing and Management, vol. 27, no. 5, pp. 517-522, 1991.
24. McIlroy, M. Douglas, "Development of a spelling list," IEEE Transactions on Communications, vol. COM-30, no. 1, pp. 91-99, January 1982.
25. Mitton, Roger, "Spelling checkers, spelling correctors and the misspellings of poor spellers," Information Processing and Management, vol. 23, no. 5, pp. 495-505, 1987.
26. Morgan, Howard L., "Spelling correction in systems programs," Communications of the A.C.M., vol. 13, no. 2, pp. 90-94, February 1970.
27. Morris, Robert and Cherry, Lorinda L., "Computer detection of typographical errors," IEEE Trans Professional Communication, vol. PC-18, no. 1, pp. 54-64, March 1975.

28. Nix, Robert, "Experience with a space efficient way to store a dictionary," Communications of the A.C.M., vol. 24, no. 5, pp. 297-298, May 1981.
29. Peterson, James L., "Computer programs for detecting and correcting spelling errors," Communications of the A.C.M., vol. 23, no. 12, pp. 676-687, December 1980.
30. Peterson, James L., *Computer Programs for Spelling Correction: an experiment in program design*, Springer-Verlag, 1980. Lecture Notes in Computer Science: 96
31. Peterson, James L., "A note on undetected typing errors," Communications of the A.C.M., vol. 29, no. 7, pp. 633-637, July 1986.
32. Pollock, Joseph J. and Zamora, Antonio, "Automatic spelling correction in scientific and scholarly text," Communications of the A.C.M., vol. 27, no. 4, pp. 358-368, April 1984.
33. Ramshaw, Lance A., "Correcting real-word spelling errors using a model of the problem-solving context," Computational Intelligence, vol. 10, no. 2, pp. 185-211, 1994.
34. Richardson, Stephen D., *Enhanced Text Critiquing using a Natural Language Parser*, IBM, 1985. Research Report RC 11332 (#51041)
35. Riseman, Edward M. and Hanson, Allen R., "A contextual post-processing system for error correction using binary n-grams," IEEE Trans Computers, vol. C-23, no. 5, pp. 480-493, May 1974.
36. Teitelman, Warren, "Do What I Mean": the programmer's assistant," Computers and Automation, pp. 8-11, April 1972.
37. Turba, Thomas N., "Length-segmented lists," Communications of the A.C.M., vol. 25, no. 8, pp. 522-526, August 1982.
38. VanBerkel, Brigitte and DeSmedt, Koenraad, "Triphone analysis: a combined method for the correction of orthographical and typographical errors," in *Second Conference in Applied Natural Language Processing, Austin, Texas* , pp. 77-83, American Association of Computational Linguistics, 1988.
39. Wagner, Robert A. and Fischer, Michael J., "The string-to-

string correction problem," Journal of the A.C.M., vol. 21, no. 1, pp. 168-173, Jan 1974.

40. Yannakoudakis, E.J. and Fawthrop, D., "The rules of spelling errors," Information Processing and Management, vol. 19, no. 2, pp. 87-99, 1983.
41. Yannakoudakis, E.J. and Fawthrop, D., "An intelligent spelling error corrector," Information Processing and Management, vol. 19, no. 2, pp. 101-108, 1983.
42. Yianilos, Peter N., "A dedicated comparator matches symbol strings fast and intelligently," Electronics, pp. 113-117, 1 December 1983.

---

*webmaster@dcs.bbk.ac.uk*