March 11, 1995

Development of a Spelling List

M. D. McIlroy

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Word processing
Dictionaries
Hashing methods

ABSTRACT

The word list used by the UNIX spelling checker, *spell,* was developed from many sources over several years. As the spelling checker may be used on minicomputers, it is important to make the list as compact as possible. Stripping prefixes and suffixes reduces the list below one third of its original size, hashing discards 60% of the bits that remain, and data compression halves it once again. This paper tells how the spelling checker works, how the words were chosen, how the spelling checker was used to improve itself, and how the (reduced) list of 30,000 English words was squeezed into 26,000 16-bit machine words.

Origin

Some years ago, S. C. Johnson introduced the UNIX® spelling checker *spell.* His idea was simply to look up every word of a document in a standard desk dictionary and print a list of the words that were not found. A typical short technical paper yielded a residual list of a few dozen correctly spelled words in addition to misspellings and typographical errors. The idea was good, workable, and instantly popular. The only problems were slowness (the program had to scan the entire dictionary) and the coverage of the dictionary.

The vocabulary of the standard dictionary[1] fell short of covering the vocabulary of real documents in two ways: regularly inflected forms were omitted according to custom, and large classes of common words–proper nouns, abbreviations, and new technical terms–fell outside of the chosen domain of the dictionary. Johnson found that simple-minded stripping of plausible suffixes worked well to recognize inflected forms, in spite of the obvious pitfalls of accepting nonwords like *alumnuses* or *lossing.* Thus his program would, given *paraded,* look up *parade* and *parad* as well, and accept it as a word if any of the three forms appeared in the dictionary. Easy as it was to fool deliberately, the spelling checker accepted few misspellings that happened in real life.

The matter of coverage was more difficult. In hopes of broadening coverage, an unabridged dictionary[2] was tried as a standard instead of the desk dictionary–with unfortunate consequences. While the coverage of words that occurred in everyday documents scarcely improved, obscure entries from the unabridged dictionary, such as *yor* and *curn,* often matched mistyped short words.

The important lessons learned from Johnson's work were that false acceptances happened infrequently enough not to impair the usefulness of the idea, and that the spelling checker was sufficiently popular to justify some attention to the problems of speed and coverage. I decided to attack both at once by constructing a shorter list specialized to its purpose, with no unnecessary words. In particular, the derivation rules were to be augmented, and derivable forms were to be dropped from the list. (The desk dictionary contained over 5,000 trivially derived forms in *ly* alone.)

Today’s spelling checker

The modern spelling checker consists of a sequence of processes:

(1) Split out the words of the document, one per line. This isn’t quite as easy as it sounds, for documents in the computer typically contain a liberal admixture of formating instructions. The nasty job of sorting out the real text is done by a program (*deroff*) written by S. I. Feldman and L. L. Cherry.

(2) Cull the words for duplicates by sorting them, preserving case distinctions.

(3) Look the words up in the stop list. If a word, or a stem obtained by stripping prefixes and suffixes, is found on the stop list, attach a stop flag.

(4) Look the words up in the spelling list. If the word has a stop flag, accept (that is, discard) it only if it appears verbatim in the spelling list. Accept a word with no flag if it, or any stem obtained by stripping prefixes and suffixes, appears in the spelling list.

(5) Print all remaining words as potential spelling errors.

The spelling checker examines ‘‘words’’ set off by blanks or hyphens, which contain more than one letter, and from which which all leading and trailing punctuation has been stripped. Thus *homeowner, Hofstadter’s, 13th*, and *AT&T* get checked, but *N* and *1981a* do not. Embedded punctuation is included, in order to catch run-on typing, as in *Out,damnedspot.* Hyphenated words are split to avoid cluttering the spelling list with innumerable ephemeral combinations like *twelve-ton* and *left-footed.* As a mildly annoying consequence, words like *topsy-turvy* and *helter-skelter* can be accepted only by placing their unwanted components in the list.

In testing a word, any upper-case letters in the spelling list or in the stop list must be matched exactly. A wholly upper-case word will match any case in the lists; and a word with an initial capital will match either a capitalized or an uncapitalized word. Thus typos like *COmmittee* or *mcilroy* will be detected, even if the spelling list contains *committee* and *McIlroy.* No further attempt is made to assure proper use of capitals; in particular, the spelling checker cannot enforce sentence capitals.

The spelling checker is popular with almost everybody who prepares text on machines, good spellers and bad spellers, expert secretaries and hunt-and-peck programmers alike. It gets used about 100 times a week on our department UNIX machine and 50 times on the local computer center’s Honeywell 6000. An average run turns up about 15 unknown words. Most of these are actually good words–unusual proper names, abbreviations, mathematical symbols, etc.–but there is plenty of pay dirt: typically half the suspect words are genuinely bad the first time a document is checked; resubmissions of corrected documents bring the overall level of authentic misspellings down near 10% of all reported words.

The number of errors that slip by undetected is more difficult to assess. A new one comes to my attention every few weeks. Some may be blamed on an incautious word list. For example, *id* had to be removed; it often resulted from misfingering *is* on a *qwerty* keyboard and never appeared in its own right. Some errors arise from misuse of homonyms; a word-based spelling checker can hardly be expected to flag *adapt* used for *adept*, much less *Schwartz* for *Swartz.* Others, such as *represenation* and *widerer*, come from too blindly and vigorously stripping prefixes and suffixes. Such examples, however, happen rarely enough that nobody has lost faith in the program.

Choosing the word list

There was raw material at hand to get started on the new list: we had on line the Brown corpus[3] of a million words of running text, replete with misspellings and other dubious words. We also had the authoritative unabridged dictionary. The words in the two lists were compared, with the Brown corpus taken as a standard of currency, and the unabridged dictionary as a standard of validity. I included the 1,000 most common last names from a 1,000,000-name phone book that was readily available from my company, and all the given names that occurred in the Brown Corpus.† To these were added the countries of the world, states of the USA, provinces of Canada, their capitals, 100 largest cities of the world, 100 largest cities of

† Given names were recognized by comparing capitalized words in the Brown corpus with a published list of 1,300 boys’ and girls’ names. The latter list had too many obsolescent entries and variant spellings (e.g. *Dian, Diann, Diane, Dianne*) to be admitted on its own.

the USA, the chemical elements, and a few other such lists. Derivable words were culled: capitalized words that were spelled like uncapitalized words, words with selected suffixes (e.g., *-s, -ment, -ical*) whose stems were also present, and words with selected prefixes (*un-, dis-, non-*). Thus the name *Wells* was unnecessary, as was *Peters* and even *Peter* = *Pete* + *-er.*

The spelling list of some 24,000 words was by then quite effective. It had been made mostly by automatic means and had never been proofread by anybody. (It has been since.) The spelling checker ran twice as fast and questioned fewer good words than it had when the traditional dictionary three times as large was used as a standard. The new list was nevertheless incomplete; some of what it gained over the standard dictionary, mostly in proper names, it lost in various perfectly ordinary words. But the checker could be put to work to improve itself. It was tested against documents already in the computer, and the output was scrutinized for words to add to the list. Copies of the output of all real runs were squirreled away, too.

When the collected output of some 10,000 runs on several thousand modest-length documents had been surveyed for new words, glaring gaps remained. Many ordinary words like *carburetor, lackluster,* and *propound* cropped up very late in the game, and no end was in sight. A check against real dictionaries indicated that a few thousand more words would be needed to make a list thorough enough that obvious oversights would only turn up in a minority of documents; these words were added to the list. Catching all those words in the field, as it were, would clearly have taken hundreds of thousands of runs, on overtly as well as passively selected documents. The exercise drove home the realization of what an enormous amount of text must be absorbed to develop a normal vocabulary, let alone a comprehensive dictionary†. Nevertheless, the field work was indispensable, and produced a thousand unquestionably useful words that don’t occur in traditional lists.

With time more prefixes and suffixes were added, as much to cut the size of the list as to improve its coverage. At the same time a ‘‘stop list’’ of improper derivatives was developed–again mostly by automatic means–to catch errors such as *thier* = *thy* + *-er* or *presenation* = *pre-* + *senate* + *-tion.* Much of the stop list is given over to identifiable classes of words, notably irregular verbs and polysyllabic verbs that double the final consonant when adding *-ing, -er,* and *-ed*; thus the stop list prohibits *seeked, bited, transmiter, begining,* etc. Of course the correct spellings are in the main list. The stop list made possible the use of reckless derivation rules that would otherwise allow likely misspellings to slip by. For example, the prefix *fore-* could not be included in the checker’s vocabulary until the stop list had been strengthened to prohibit errors like *foregather* and *forego.*

Where does one stop in accepting words? Dictionary makers try to cover everything but proper nouns in a broad range of fields. Proper nouns, however, abound in real text; a spelling checker that ignores them will be weak indeed. But there are tens of thousands of proper nouns, especially last names, coming from all the languages of the world, with a bewildering variety of variant spellings. To avoid being inundated, I picked last names almost solely by frequency, which was easy to determine from telephone books. Of course one must admit *Einstein* and *Shakespeare,* but it is not at all clear how objectively to cast the net for such universally famous people.

Some proper names come in natural clusters. Big companies, like *IBM* and *Exxon,* come from the top of the Fortune 500. Book publishers, whose names appear ubiquitously in bibliographies, are another group. Place names have been admitted essentially by rule: oceans, countries, states, capitals and big cities get in, others don’t. Planets come in all at once, as do bright stars and major constellations. Mythological characters troop in together from the index to Bulfinch;[4] but other literary characters can’t be so neatly classified, and come in somewhat hit-or-miss. As a result of the original method of collection by intersecting the Brown corpus with the unabridged, *Falstaff, Copperfield,* and *Ahab* happen to be in; *d’Artagnan, Raskolnikov,* and *Fagin* are out.

Among variant spellings, I have admitted just one for each word, as it would be unwise to approve wayward mixtures of spelling conventions within any particular document. Thus the common *Johnson* is accepted, the rarer variant *Jonson* rejected. Unfortunately, there is no way to detect a substitution of the

† Each million words surveyed in preparing the spelling list is equivalent to 5 or 10 books, an amount a not particularly voracious reader might be expected to ingest in a year. Many more than the present few millions of words would have to be surveyed to capture a passable human vocabulary.

common form where a rare variant is really intended.

Aside from proper names, I have chosen between variants strictly by spelling rules. Thus, according to the rule for doubling final consonants in accented syllables, *formated* is to be preferred over *formatted* and *traveled* over *travelled,* even though all these spellings are generally acceptable. This policy also picks (by the rule that *-e* is dropped only before vowels) *judgement* and *acknowledgement* over the self-conscious alternatives *judgment* and *acknowledgment,* which are customarily preferred in American spelling. (Fowler[5] approves this choice.) For the same reason, *aging* and *movable* are preferred over *ageing* and *moveable.†*

Because it is so difficult to know where to stop, I have taken a hard line with the nearly limitless vocabularies of chemistry, medicine, biology, botany, and mineralogy. The chemical elements have been accepted, but very few compounds. Common anatomical names like *spleen* and *jaw* are in, but not latinate names like *ulna* and *villi;* and similarly with diseases: *measles,* but not *schistosomiasis.* From an early attempt to cover common, but not scientific, names of animal and plant species, taking Palmer[6] as an authority, I have retreated now to accepting only very familiar species, *lion* and *oak,* but not *aardvark* and *chinquapin.* Because laymen's knowledge of these vocabularies tends to be spotty, any reasonably thorough list will contain dozens, if not hundreds, of words unfamiliar to any given person. Adjectives peculiar to the chemical and life sciences, such as *obovate* and *cerebrospinal,* have also been omitted. The spelling checker would be more useful in a chemical or medical department if it knew more prefixes such as *methyl-, oxy-, cerebro-,* and *radio-.* But that would not be enough; hundreds of new words would be needed too.

To the spelling list used in my own department I have added some 550 parochial words, about 2% of the total vocabulary. Most of these are proper names: the people who use the machine, and authors like *Knuth* and *Wirth,* whom we, but not people from other fields, refer to frequently. Some are jargon of our trade: *kludge, opcode,* and *quicksort;* some are technical abbreviations: *CPU, EE;* some are important companies in our field: *Intel, BBN.*

The degree to which it covers a real dictionary, Webster's collegiate, was determined by running *spell* on the dictionary itself. Ignoring proper names, for which the dictionary's criteria differ markedly from *spell's,* the 25,500 noncapitalized words in *spell's* list covered 47,000 of Webster's 68,000 words. The spelling list contains some 1,000 words (*phonon, vii, zilch, defuse, backfill,* etc.) literally and covers by derivation many obvious compounds (*nondisclosure, preprocessor, unproved, deflatable, electrooptical,* etc.) that are not in the dictionary at all. Because it was constructed by reduction from an unabridged dictionary, it also contains some root words (*librate, figurate,* etc.) that are not in the collegiate, although their common derivatives (*libration, figurative*) are. As most nouns come in singular and plural, verbs in singular, plural, past, and progressive, and adjectives often in positive, comparative, and superlative, dictionary words typically cover two English tokens. I infer, therefore, that the noncapitalized words in the spelling list cover about four tokens of English apiece–well over 100,000 in all.

Affixes

The spelling checker knows approximately 30 suffixes and 40 prefixes. Words are looked up by successively stripping suffixes, and within the suffix-stripping loop, by successively stripping prefixes until an exact match is found. Prefixes (at least those handled here) may be stripped mindlessly, but suffixes are another matter. Fortunately, we are interested only in soundness of results, not of method. Silly derivations like *forest* = *fore* + *-est* are perfectly acceptable, and even welcome. Anything goes, provided it makes the list shorter without impairing its discriminatory power. Still, as the rules listed in Appendix A show, different suffixes interact with each other to make stripping them a task of some delicacy. See, for example the set of rules for words in *-ly,* where *possibly, correctly,* and *handily* all need different treatment.

To forestall false derivations like *maping*→*map,* a suffix that begins with a vowel is not stripped when that would leave a monosyllable ending with vowel-consonant. If, however, stripping such a suffix would leave a monosyllable with a final doubled consonant, one of the consonants may need to be stripped

† In the case of *programming,* I have strayed from the rules, only because I belong to the computer programming crowd rather than the programed learning fraternity.

too. Both the doubled and undoubled stems are tried, to insure finding derivations like *mapping*→*map* as well as *buzzed*→*buzz.* The pertinent monosyllables for this rule are sets of letters containing exactly one vowel and not ending in *x* or *w.* Thus *fad* and *ten* are pertinent monosyllables, *neat* and *mix* are not. This rule, as do all the rules, may cause the spelling checker to try nonsense, such as *buz* as well as *buzz* as a possible stem for *buzzed.* Usually harmless, such false tries have been put on the stop list when they give improper derivations that come out too close to real words (e.g., *fadded* instead of *faded*)

Only minimal care need be taken in stripping prefixes, of which a complete list appears in Appendix B. They may be recognized in any order provided only that *under-*, when present, must be stripped in preference to *un-*. Certain common prefixes, notably *in-* and *de-*, are missing from the list. These prefixes are just too unreliable; *in-* is confusable with *un-*, to say nothing of requiring special treatment when prefixed to words beginning with one of *l, m, p,* or *r,* as in *illicit, immeasurable, impossible,* and *irrepressible.* Many words beginning with *de-* are not derived by prefixing at all; this prefix is not as productive as it might seem.

Some of the prefixes invite false derivations for likely typos such as *enbed, forego, antiroom, unpossible* and *dispair.* A particularly insidious class of errors crops up from admitting prefixes that are also words, for then missing spaces yield acceptable words: *outof, underthe, overand.* A few such prefixes were adopted only after considerable soul-searching when actual counts showed that they would significantly shorten the list. A similar problem exists with the suffixes *-ship, -hood,* and *-like.* All of the examples in this paragraph, and many more words like them, have been placed on the stop list.

Improvement

UNIX software tools helped build the spelling checker. Whenever a new suffix was adopted, I would follow a standard drill to cull out redundant entries in the spelling list and to strengthen the stop list against new classes of errors caused by stripping that suffix.

As an example, consider the suffix *-er.* Using the UNIX utility *grep,* all words ending in *-er* were selected from the spelling list. With the text editor, two prospective stem words were constructed from these by stripping *-r* and *-er* in turn. *Spell* itself determined which of these stems were not in the spelling list. The utility *comm* (for finding the members that two lists do–or do not–have in common) complemented the list of bad stems with respect to the list of all prospective stems; the result was a list of stems whose *-er* derivatives had become superfluous. Those derivatives were reconstructed with the editor, and the final list was expunged from the spelling list by using *comm* once again. At various times in the process, the lists had to be sorted.

I printed many of the lists, which seldom ran to more than a few multicolumn pages, and scanned them in search of ideas for improvement. In this way were spotted derived words whose stems were missing from the spelling list; the stems would go in and the derivatives would be discarded. Some words turned up, for example *bookkeeper* and *homemaker,* with hypothetical stems that could be safely added to the spelling list. Thus *bookkeep,* an unlikely typo, covers two derivatives, *bookkeeper* and *bookkeeping,* for the price of one. Once the pattern had been noticed, dozens of similar words were searched out automatically.

Improving the stop list was a trickier matter, for the task here was to find attractive nuisances, nonexistent words that could plausibly surface as misspellings or typing errors and would be derivable from stems in the spelling list. For example, blind stripping of *-er* would accept *beginer, transmiter, armer, reflecter, grammer, lader,* and *baner,* which are typical examples of typos or spelling errors that happen in real life. Full-scale dictionaries, though not perfect as spelling lists, were an invaluable help. Having identified *grammer* as a plausible error, I asked whether the unabridged dictionary contained any other words in *-ar,* the *-er* analogue of which could be falsely derived, and turned up almost 2,000 others, including *agar, familiar, granular,* and *polestar.* These boiled down to fewer than 90 upon casting out derivatives like *subglobular* and words like *dinar* that have a legitimate *-er* analogue. Words of the *armer-reflecter* class were discovered similarly. Words of the *beginer-transmiter* class were identified in yet another way. A pronouncing dictionary was searched (again with the UNIX *grep* utility) for verbs ending in vowel-consonant and accented on the last syllable. The naive *-er* (also *-ed* and *-ing*) derivatives of these words joined the stop list. At the same time, versions with properly doubled final consonants, *beginner* and *transmitter,* were added to the spelling list if their stems were there.

The stop list saves far more space than it costs. For example, some 350 *-er* words occur in the stop list. But the suffix *-er* covers about 3,300 words of the collegiate dictionary and certainly more of the unabridged, so this one suffix saves about 3,000 words net. Though not all affixes are so productive, shifting the burden from the 30,000-word spelling list to the 1,300-word stop list generally helps to ease the pressure of limited address space.

The following list exemplifies most categories of words on the stop list. In general, one representative of each category had to be thought up, or discovered by a sharp-eyed user; the rest could usually be collected by automatic construction from the spelling list, correlation with dictionaries, and inspection of the resulting lists.

barly	*bar* + *-ly,* should be *barely* or *barley*
bursted	*burst* + *-ed,* should be *burst*
Floridan	*Florida* + *-an,* should be *Floridian*
dispair	*dis-* + *pair,* should be *despair*
duely	*due* + *-ly,* should be *duly*
enbattle	*en-* + *battle,* should be *embattle*
invertion	*invert* + *-tion,* should be *inversion*
foreswear	*fore-* + *swear,* should be *forswear*
telenet	*tele-* + *net,* should be *Telenet*
bes	*be* + *-s,* short compound, can result from mistyping *best, bed,* etc.
antiroom	*anti-* + *room,* should be *anteroom*
axises	*axis* + *-es,* should be *axes*
counterfit	*counter-* + *fit,* should be *counterfeit*
counterto	*counter-* + *to,* should be *counter to*
unoffensive	*un-* + *offensive, inoffensive* is preferred
transmiter	*transmit* + *-er,* should be *transmitter*
sensable	*sense* + *-able,* should be *sensible*
speciality	*special* + *-ity,* should be *specialty* in American usage

Hashing

The spelling list, whose clear text runs to about 250,000 bytes, is accessed randomly.† This in itself suggests hashing; but much more significant is the possibility that a hashed version of the dictionary may be small enough to fit comfortably in main memory. Moreover, if the hash space is sufficiently sparsely populated, it may be possible to discard the clear text entirely, at the price of occasionally accepting a bad word. If the probability of such an error is kept as low as 1 in 2000, then since typical rough drafts rarely have more than 20 bad words, a mistake will be accepted in about 1 in 100 documents, an error rate that is minuscule compared to the incidence of undetectable errors such as *an* for *and* or *form* for *from.*

Floyd and Gill explicitly proposed this use of hashing,[7] but the main idea had been hinted at long before in Morris's classic paper on scatter storage techniques, where he said of it, ‘‘No one, to the author's knowledge, has ever implemented this idea [throwing the original data away], and if anyone has he might well not admit it.’’[8]

Two different hashing methods have been implemented. The first, based on a simple superimposed code scheme first proposed by Bloom,[9, 10] was supplied by D. M. Ritchie and succeeded in encoding a 25,000-word list into 50,000 bytes. A more elaborate method, in which values of a conventional hash function are represented in a differential Huffman code, squeezed 30,000 words into 52,000 bytes. The stop list is handled by the same method in a different process.

Data compression of the spelling list itself may be an alternative to hashing, since relatively easy techniques suffice to compress an alphabetized list significantly.[11] A sophisticated compression scheme of R. H. Hardin's came within 20% of the size of the second hashing technique, and took only about 50%

† Even though the input words have been sorted to cull duplicates, the spelling list is not consulted in sequential order, because the alphabetic order of the input may be destroyed by stripping affixes.

longer to decode;[12] neither figure is enough to rule out such error-free encodings on a more capacious machine.

The next two sections discuss the two hashing methods; Appendix C reviews the infinite Huffman code used in the second method.

Superimposed codes

Superimposed coding spatters each entry randomly across a large bit table so thoroughly that the hash values cannot be reconstructed from the table. Let an N-bit table be initially clear. For each entry, compute k independent hash functions into the range 0 to $N-1$, and set the corresponding bits of the table to 1. To probe the table, ask whether all k designated bits are 1. Remainders modulo the k largest primes less than N constitute a convenient and effective set of hash functions. As the last few bits of the hash table do not pull full weight in this scheme, the table is effectively a little shorter than N bits.

To convey maximum information, i.e., to achieve a minimal error rate for a given number of entries, the bits of the table should take values 0 and 1 with equal probability. The probability q of a bit being 0 is

$$q = \left[1 - \frac{1}{N} \right]^{km} \approx e^{-km/N},$$

where m is the number of entries; q becomes ½ when

$$k = \frac{N}{m} \log 2 .$$

The probability of false acceptance in this instance is

$$(1 - q)^k = 2^{-k}.$$

For the then 25,000-word list hashed into 25,000 16-bit words, or 400,000 bits, the optimal number of hash functions was $k=11$, and the probability of false acceptance was less than 1 in 2,000 tries.† Experience showed, however, that 25,000 words wasn't enough; almost every run turned up some worthy new word. To go further, without increasing the hash table size or error rate, the table had to be compressed more tightly.

Compressing conventional hash

If each entry of a v-member vocabulary is hashed into b bits, and only the hash values are remembered, then the probability of false acceptance will be the probability of accidentally hashing into a good value, namely $v/2^b$. A desirable vocabulary size for a spelling list is about 30,000, or roughly 2^{15} words, a desirable error rate about 1 in 2^{12}; thus an appropriate hash value size is 27 bits.

How can $2^{15} \times 27$ bits of hash values be jammed into the PDP11's addressable memory of $2^{15} \times 16$ bits? First, let us ask how well we might expect to compress this information. There are

$$\left[\begin{matrix} 2^b \\ v \end{matrix} \right]$$

possible sets of v distinct hash values, so

$$\log_2 \left[\begin{matrix} 2^b \\ v \end{matrix} \right] \approx v[b - \log_2(v/e)]$$

bits should suffice to encode such sets. In our case where $v=30000$ and $b=27$, this works out to 13.57 bits per entry–small enough to fit!

When the values are listed in order, the mean first difference, a, is given by $a = 2^b/v$, so differences

† *Spell* makes about 1½ tries per word. In the rare worst case, a word may be tried first in all caps, then in two capitalizations–initial and none–times the number of prefixes times the number of suffixes.

should typically be representable in about $b - \log_2 v$ bits, in our case 12. Thus, by storing differences, we may expect to compress the list of hash values nearly to the information-theoretic limit. All we need is an appropriate variable length code.

To a very good approximation, we may regard the ordered sequence of hash values as having been chosen by a Poisson process with rate $v/2^b$. The differences, therefore, must be exponentially distributed with mean $a = 2^b/v$. With $a = 2^{27}/30000$ as before, the simple Huffman code given in Appendix C has an expected codeword length of 13.60 bits. To look up a hash value, one must sum the differences from the beginning until the value sought is reached or surpassed–a time-consuming method indeed. However, by partitioning the table of differences among M bins, the search can be speeded up by a factor of M without giving up much space. In our case, the expected extra space for breakage and for pointers to 512 bins brings the effective mean codeword length up to 14.01 bits: a tight, but workable fit.

Spelling at Bell Labs

The collected output of the spelling checker, gathered for self-improvement, would be a gold mine for students of human spelling habits. Although I have made no consistent study of errors, in scrutinizing the output of thousands of runs of *spell,* I have noticed a few interesting facts.

Most spelling errors are merely typos. The typical less-than-expert typists who use our system find it a godsend to have a quick plausibility check on their work. And you can almost hear the sighs of relief as you watch the gaffes disappear from the work of the few truly terrible spellers. Even accomplished secretaries hit pay dirt by running *spell;* I suspect that they type a little faster, secure in the knowledge that they can catch many fluffs before their clients see them.

Some self-assured spellers ignore the advice of *spell,* thinking that the elaborate words they use are beyond its ken. Their favorite, and most insistent, misspelling is *idiosyncracy.* Seeing it come up over and over again, I have been led to violate the anonymity of the collected misspellings, so I could tell the perpetrators that, really, the word is *idiosyncrasy.*

My candidate for the most misspelled word at Bell Labs has eight potential flavors, a majority of which I have actually seen used: *acomadate, acomodate, acommadate, acommodate, accomadate, accomodate, accommadate,* and *accommodate.* Scarcely a day passes without somebody misspelling it, but fortunately no one has courage in his convictions about this one; it always gets cleaned up in later revisions.

A goodly number of Bell Labs people from Britain or elsewhere in the Commonwealth write with British spelling, and some tell me that acclimation (or acclimatization) has shaken their certainty about both British and American spelling. In using the spelling checker they had no check on consistency: *spell* would report on all their intended British spellings and overlook the unwanted Americanisms that crept in. I was thus led to attempt a British version of the program. Most of the differences fall into recognizable classes such as *center-centre, color-colour, ether-aether, traveled-travelled,* and *realize-realise.* (In the last instance Fowler prefers *-ize,* but ordinary Britishers tend the other way.) The same automatic methods used in developing the original spelling list served to locate most of the necessary changes, and the spelling list was split into a common core–about 99% of the words–and small annexes for the British and American variants. The *-eled* words required special care. To keep a single stop list, the *-eled* forms had to be stopped for the benefit of British spelling, and then overridden in the American annex. In exactly the same way, *speciality* and *acclimatize* had to be stopped for American and then overridden for British. It takes almost no extra work to maintain the second dialect.

Plausible improvements

This section and the next examine some ways in which the spelling checker might be improved in speed or effectiveness. Peterson refers to further literature on the subject.[13]

Occasional errors due to incomplete control over affixes could, in principle, be suppressed by supplying a complete word list. Sheer bulk argues against carrying this idea through literally: plurals alone will nearly double the size of a spelling list. However the same effect can be obtained by annotating each word of a reduced list to show what affixes may be attached. The Scrabble dictionary[14] and some computer spelling checkers[13] do this. The scheme works reasonably well with suffixes, but not so well with prefixes and suffixes simultaneously. How, for example, can one compactly encode the fact that *deserve* may be modified to make *deserving* and *undeserving* but not *undeserve?* And how can one sensibly predict perfectly reasonable neologisms, say *nonforwarded, decelebrate, drainkeeper,* or *thermoporous?* (I believe that *undeserve* is too bizarre to happen unwittingly, and that it would be counterproductive deliberately to exclude plausible nonce words or new coinages, especially from technical writing, where the new is commonplace. This viewpoint has driven me, somewhat reluctantly, to accept deplorable words like *finalize* and the Bell System's supreme horror, *connectorization.*)

It may be argued that a completely controlled list will suffer only by questioning more words than it should. In my experience, though, users tend to ignore good advice from a spelling checker if spurious advice is too common.† There is a tradeoff between machine recklessness, which lets errors go unreported, and human inattention, which lets reported errors go unnoticed. Without claiming that *spell's* degree of control is ideal, I am nonetheless convinced that a perfectly controlled list would have to be much larger–100,000 to 200,000 words–to work as well.

One halfway approach to control would be to annotate each word with its part(s) of speech, and annotate the affixes with the parts of speech to which they apply and the parts of speech they signify. The words in Webster's collegiate fall into about 120 classes according to the set of parts of speech that a word may have. On the somewhat shaky assumption that a classification for a spelling list would be similar, I estimate that the scheme would increase the storage for the spelling list by about 1/6, as the entropy of the dictionary classification is 2.38 bits. Aside from the effort of classifying 30,000 words, that turns out to be just too much for our 16-bit address space, but would be cheap enough in a roomier environment.

A minor, but possibly useful, improvement would be to include final periods in the ‘‘words’’ recognized by *spell* in order to control the style of abbreviations. Period would then be required in contexts like *Mrs.* and *Mr.* and would be regarded as a suffix otherwise.

A filter for high-frequency words may be added at the first stage of the pipeline. This saves some time: discarding the top 200 words according to the Brown corpus–covering somewhat over half of all occurrences in that sample–with a fast finite state algorithm (the UNIX *fgrep* utility[15] for linear-time recognition of a set of strings) reduced the overall processing of a 5,500-word paper from 77 CPU-seconds to 65 on a PDP11/70.

The present spelling checker squeezes the most out of every bit of memory; with more memory, word lookup (about 40% of total processing) could run considerably faster. Huffman-coded hash would be speeded up by partitioning memory into more bins; another 1,000 bytes dedicated to binning overhead would double the speed of lookup. A base-16 Huffman code instead of base-2 would nearly double the speed of lookup by eliminating bit-picking; this stratagem, too, would cost about 1,000 bytes.

Superimposed coding could be speeded up by using a larger, more sparsely populated, bit array. Doubling the memory would accommodate the extra words that Huffman coding squeezes in and also double the lookup speed with no loss of discriminative power.

If lookup speed were doubled, and frequent words were filtered out, the presort (about 20% of total processing) might be dropped in favor of looking the residual words up at each occurrence. The net gain for all these changes would be something like 25%: not very exciting–except possibly in turnaround time–for a program that runs only 10 minutes a day.

If a quick speedup with no extra capability were desired, and memory were no object, I would choose the easier superimposed coding; but it uses storage cavalierly, and offers no opportunities for extra function. The Huffman method, while somewhat slower (10%-20% overall), leaves the door open for exploiting extra memory for word-class information. Of the two methods, it gets the nod for the long run, but one should not forget the incompletely evaluated schemes for compressing clear text without loss.

It has been suggested that data compression may not be necessary at all with larger memories than the minicomputers on which *spell* began, but I would disagree. The Huffman-coded hashed spelling list is only 1/5 the size of the original 250,000 bytes of clear text, an advantage that may well tell in response time when data transfers from secondary memory are taken into account.

† Anecdotal evidence: *spell* rejects more than 150 words in this paper, which is infested with trash. While I was writing it, I repeatedly overlooked genuine typos buried in the middle of that long list.

Appendix A: Suffix rules

Final *-s*. No other suffix can follow any of these.

-hes → *-he* → *-h*	*aches*→*ache, arches*→*arch*
-ses → *-se* → *-s*	*vases*→*vase, basses*→*bass*
-xes → *-xe* → *-x*	*axes*→*axe, fixes*→*fix*
-zes → *-ze* → *-z*	*hazes*→*haze, buzzes*→*buzz*
-ies, -es	[use the *-er* rule below]
-s →	[not preceded by *s*]
-'s →	[not preceded by *s*]

The suffix *-er* and others treated similarly.

-ater	[do not strip; to avoid *creater*→*create*, etc.]
-cter	[do not strip; to avoid *reacter*→*react*, etc.]
-ier → *-y*	*copier*→*copy*
-er → *-e* →	*baker*→*bake, smaller*→*small*
[the following suffixes are treated similarly]	
-ied, -ed	*curried*→*curry, bored*→*bore, seated*→*seat*
-ies, -es	*flies*→*fly, palates*→*palate, woos*→*woo*
-iest, -est	*liveliest*→*lively, wisest*→*wise, strongest*→*strong*

The suffix *-ing* and others treated similarly.

-ing → *-e* →	*living*→*live, laughing*→*laugh*
[the following suffixes are treated similarly]	
-ist	*stylist*→*style, dentist*→*dent*
-ism	*cubism*→*cube, socialism*→*social*
-ity	*scarcity*→*scarce, rapidity*→*rapid*
-ize	*lionize*→*lion*
-able	[except after *c* and *g*] *livable*→*live, portable*→*port*

Exceptions to the *-ing* rule.

-blity	[stop, to avoid *noblity*→*noble*]
-bility → *-ble*	*nobility*→*noble*
-fiable → *-fy*	*identifiable*→*identify*
-logist → *-logy*	*psychologist*→*psychology*

Suffixes that are simply replaced.

-graphic → *-graphy*	*photographic*→*photography*
-istic → *-ist*	*stylistic*→*stylist*
-itic → *-ite*	*martensitic*→*martensite, politic*→*polite*
-like →	*ladylike*→*lady*
-logic → *-logy*	*biologic*→*biology*
-ment →	*battlement*→*battle*
-mental → *-ment*	*supplemental*→*supplement*
-metry → *-meter*	*thermometry*→*thermometer*
-nce → *-nt*	*inadvertence*→*inadvertent*
-ncy → *-nt*	*potency*→*potent*
-ship →	*discipleship*→*disciple*
-ical → *-ic*	*mystical*→*mystic*
-ional → *-ion*	*regional*→*region, national*→*nation*

Suffixes *-ly*, *-ful*, etc., that are stripped except after *i*.

-bly → *-ble*	*horribly*→*horrible* [a unique case]
[fRthe usual cases]	
-ily → *-y*	*scantily*→*scanty*
-ly →	*partly*→*part*
[the following suffixes are treated similarly]	
-iful, -ful	*dutiful*→*duty, harmful*→*harm*
-ihood, -hood	*likelihood*→*likely, neighborhood*→*neighbor*
-iless, -less	*penniless*→*penny, listless*→*list*
-iness, -ness	*heartiness*→*hearty, coolness*→*cool*

The suffix *-tion* and relatives.

-ification → *-ify*	*specification*→*specify*
-ization → *-ize*	*rationalization*→*rationalize*
-ction → *-ct*	*detection*→*detect*
-rtion → *-rt*	*exertion*→*exert*
-ation → *-ate*	*creation*→*create*
-ator → *-ate*	*creator*→*create*
-ctor → *-ct*	*detector*→*detect* [see also *-ater* and *-cter* under *-er*]
-ive → *-ion*	*creative*→*creation, decisive*→*decision*

Suffixes pertinent only to capitalized words.

-an → *-a*	*American*→*America*
-onian → *-on*	*Jeffersonian*→*Jefferson*

Appendix B: Prefixes

anti-
auto-
bio-
counter-
dis-
electro-
en-
fore-
geo-
hyper-
intra-
inter-
iso-
kilo-
magneto-
meta-
micro-
mid-
milli-
mis-
mono-
multi-
non-
out-
over-
photo-
poly-
pre-
pseudo-
psycho-
re-
semi-
stereo-
sub-
super-
tele-
thermo-
ultra-
under-
un-

Appendix C: Infinite Huffman codes

It is easy to see that an infinite set of messages with independent probabilities 1/2, 1/4, 1/8, ... has an entropy of 2 bits and a perfect Huffman code 0, 10, 110, 1110, ... This geometric series, which is self-similar in that any tail is a scaled replica of the whole, has a self-similar code: any tail is a shifted and 1-padded replica of the whole. In general, a set of probabilities

$$P_i = (1-r)r^i, \quad i = 0, 1, 2, ...,$$

where $r^m = \tfrac{1}{2}$ for some integer *m*, will be self-similar in blocks of length *m*: removing an initial block scales the series by a factor of two. We expect an appropriate code also to be self-similar in blocks of *m*, with each block 1 bit longer than the block before. Such codes have been described by Golomb.[16] For general *r*, *m* must be taken to be the least integer such that $r^m \le \tfrac{1}{2}$.[17]

Without loss of generality, let the first codeword of length k end in zero and let succeeding codewords of that length count in binary. If the value of the first codeword is $2x$, then by self-similarity the *m*th succeeding codeword must be $2^k + 2x$. Furthermore, the *k*-bit prefix of this $k+1$-bit codeword must have value $2x+m$, lest there be a wasted k-bit pattern.

To illustrate the preceding reasoning, let us take $k=4$ and $x=3$. The code runs

$k=4$					
0	1	1	0		
0	1	1	1		
1	0	0	0		$m=5$
1	0	0	1		
1	0	1	0		
1	0	1	1	0	
1	0	1	1	1	
1	1	0	0	0	
1	1	0	0	1	
1	1	0	1	0	

etc.

The code is clearly self-similar by blocks of length $m=5$, and the overlapped extension of the first block shows how the k-bit codes count from $2x=6$ up to $2x+m=11$ in running through one block and into the next.

For a given m, the length of the codewords in the first full block of equal-length codewords is the least integer k that satisfies

$$2(2x + m) = 2^k + 2x$$

for some nonnegative integer x. Hence

$$k = \lceil \log_2 2m \rceil,$$
$$x = 2^{k-1} - m = 2^{\lceil \log_2 m \rceil} - m.$$

From the known mean a, we can compute $r = a/(1+a)$, then $m = -\log 2/\log r$, and finally k and x. When $x \neq 0$, the first x codewords of the entire code have length $k-1$ and values 0 through $x-1$. The full code for the illustration is

0	0	0			
0	0	1			$x=3$
0	1	0			
0	1	1	0		
0	1	1	1		
1	0	0	0		$m=5$
1	0	0	1		
1	0	1	0		
1	0	1	1	0	

etc.

Decoding is easy. If w, the first $k-1$ bits of cipher text, is less than x, then the first word of cipher text is w, as is the first word of clear text. Otherwise extend w one bit at a time until u, its least significant k bits, becomes less than $2x+m$. The first word of cipher text is again w, and the first word of clear is $x+u+(s-1)\,m$, where s is the number of bits of extension.

The expected codeword length is

$$\sum_{i=0}^{\infty} P_i L_i = k - 1 + \frac{r^x}{1-r^m},$$

where L_i is the length of the ith codeword. Recalling that $r^m = ½$, we finally find the mean length of a codeword to be $k - 1 + 2r^x$, which varies between k and $k+1$ as x varies between m and 0.

1. *Webster's Seventh New Collegiate Dictionary*, G. & C. Merriam, Springfield, Mass. (1970).
2. *Webster's New International Dictionary of the English Language*, Second edition, G. & C. Merriam, Springfield, Mass. (1961).
3. C. K. Kucera and W. N. Francis, *Computational Analysis of Present-Day American English*, Brown University Press, Providence (1967).
4. Thomas Bulfinch, *Stories of Gods and Heroes* (originally *The Age of Fable*), Crowell, New York.
5. H. W. Fowler and E. Gowers, *A Dictionary of Modern English Usage, Second Edition,* Oxford (1965).
6. E. L. Palmer, *Fieldbook of Natural History,* McGraw-Hill, New York (1949).
7. L. Carter, R. Floyd, J. Gill, G. Markovski, and M. Wegman, ‘‘Exact and approximate membership testers,’’ in *Proc. 10th Annual ACM Symposium on the Theory of Computation*, ACM, San Diego (1978). Floyd and Gill addressed spelling checkers specifically in an unpublished precursor of this paper.
8. R. Morris, ‘‘Scatter storage techniques,’’ *Comm. Assoc. Comp. Mach.* **11**, pp. 38-43 (1968).
9. B. H. Bloom, ‘‘Space/time trade-offs in hash coding with allowable errors,’’ *Comm. Assoc. Comp. Mach.* **13**, pp. 422-426.
10. D. E. Knuth, *The Art of Computer Programming, Vol. III: Sorting and Searching,* Addison Wesley, Reading, Mass. (1973).
11. R. Morris and K. Thompson, *Webster's Second on the Head of a Pin*, Bell Laboratories internal memorandum (1974).
12. R. H. Hardin,, *personal communication*, 1981.
13. J. L. Peterson, ‘‘Computer programs for detecting and correcting spelling errors,’’ *Comm. Assoc. Comp. Mach.* **23**, pp. 676-687 (1980).
14. Selchow & Righter Company, *The Official Scrabble Players Dictionary,* Pocket Books, New York (1979). Scrabble is a trademark of Selchow & Righter Company.
15. A. V. Aho and M. J. Corasick, ‘‘Efficient string matching: an aid to bibliographic search,’’ *Comm. Assoc. Comp. Mach.* **18**(6), pp. 333-340 (1975).
16. S. W. Golomb, ‘‘Run-length encodings,’’ *IEEE Trans. on Information Theory* **IT-12**, pp. 399-401 (1966).
17. R. C. Gallager and D. C. Van Voorhis, ‘‘Optimal source codes for geometrically distributed integer alphabets,’’ *IEEE Trans. on Information Theory* **IT-21**, pp. 228-230 (1975).