Talisman: A Prototype Expert System for Spelling Correction

Hal Berghel
Cecily Andreu

Department of Computer Science
University of Arkansas

Abstract:

This paper reports on the status of **TALISMAN**, a logic-based spelling assistance package for MS-DOS microcomputers which is currently being refined and tested in our laboratory. The essence of the package is described, and is contrasted with current products. The uniqueness of the approach lies in the fact that formal definitions of spelling errors are directly encoded into the program. Some recent benchmark results indicate that **TALISMAN** may actually out-perform competitive products as well as exceed their accuracy and overall effectiveness.

Introduction

Spelling assistance programs are expected to do at least three things: determine whether each and every word-token is correctly spelled (verification), provide some help for the user toward correcting any errors (correction), and standardize the document so as to facilitate the above (normalization). We will discuss each of these stages of document processing, insofar as it relates to **TALISMAN**, below.

The most challenging part of spelling assistance software is the correction stage. In our view, there are two basic classes of spelling errors to be detected: typing errors and knowledge errors. Typing errors are simply data entry errors. They are easy to define, there are considerable empirical data on their frequency and distribution (see references [5] [6] [[9] [17] [18]), and they fall into a small number of categories. Exactly the opposite is true of knowledge errors: their descriptions are much more complex which makes them more difficult to classify, and there are many more types of them. As we explain, below, **TALISMAN** is currently implemented for typing errors and some closely related knowledge errors. It is restricted to these errors primarily for testing and benchmarking purposes since most current products focus on them. After the testing cycle is completed, the next state of development of the product will attempt to deal with knowledge errors, as such and in general.

We approach spelling correction from the point of view of 'generic error types'. Generic error types are orthographical transformations of words. For example, character transposition is a generic error type, including the special cases of transposition of adjacent characters, transposition of two characters around a third, and so forth. The three additional generic errors which are of present interest are insertion, deletion and substitution, in any number of forms. It should be mentioned that some current products define spelling corruption graphemically. (A grapheme is an orthographical approximation of a phonetic representation of a word). We avoid this type of representation because of the inherent imprecision.

Table 1 illustrates the distinction between the two classes of spelling errors. This reveals several error types from both sources with which we are currently experimenting. As is evident from the table, our four generic error types account for seven types of typing errors and at least four types of knowledge errors. We have previously referred to these four generic error types as Damerau errors [3], after the author of the first published report of their prominence among typing errors [5]. Of course, our ultimate goal will be to identify and prioritize a multitude of generic error types, so that the greatest range of spelling errors of all types may be recognized.

In our view there are at least three levels of sophistication for spelling correction software. At the lowest level, the software may determine and prioritize correctly spelled words which are orthographically and/or phonologically similar to the word token. This is the level at which most products operate, although with varying levels of success. With **TALISMAN** we achieve, in terms of information theory [16], 100% precision, complete recall and zero fallout (with respect to the generic error types) because the logical description of the error type is directly encoded into the program (i.e., no approximations or metrics are used). For further discussion of a logic-based approach to approximate string matching, see [3].

TABLE 1
SPELLING ERRORS

I. Typographical Errors (decreasing frequency of occurrence [peterson])

error	generic error type
1. character transposition	transposition
2. extra character	insertion
3. missing character	omission
4. wrong character	substitution
5. 2 extra characters	insertion
6. 2 missing characters	omission
7. 2 characters transposed around a third	transposition

II. Knowledge Errors (miscellaneous - frequency unknown)

error	generic error type
1. orthographical	
1. doubled consonants	
a. unnecessary doubling (e.g., gallactic)	insertion
b. failure to double (e.g., mispelling)	omission
2. mistaken vowel order (e.g., heirarchy)	transposition
3. mistaken consonant order (e.g., nmemonic)	transposition
B. phonological	
1. errors from mispronunciation (e.g., numonia)	?
2. homonymal errors (e.g., boar for bore)	?
C. syntactic	
1. prefix confusion (e.g., priempt)	?
2. suffix confusion (number,case)	?
3. gender confusion re: borrowed words	?
D. semantic	
1. confusing unrelated words with similar spellings (e.g., diary and dairy)	?

The middle level would require that the software identify the correctly spelled words which are grammatically similar to the token as well as being orthographically or phonologically similar. This objective, which requires the inclusion of a parser into the spelling corrector, is currently being investigated by the more innovative software houses. We see this objective as attainable (to a certain degree, at least) with current technology within the very near future. For a brief discussion of the connection between parsing and our approach to spelling correction, see [2].

The highest level would require the identification of correctly spelled words which are grammatically and semantically appropriate given the context. This would require some form of natural language 'understanding', which is, as of this writing, beyond current technological limits.

We see **TALISMAN** as a paradigm for rigorous and effective, low-level, spelling correction with considerable potential for the middle level as well. What we describe, below, are the major characteristics of this product as far as it relates to the general problem of spelling assistance in text editing environments. Admittedly, we use the term 'expert system' loosely. The 'expert' component of **TALISMAN** amounts to a direct encoding of our knowledge of certain categories of spelling errors, as expressed in first order logic.

Document Normalization

The first phase of spelling checking is document normalization. This refers to a procedure, or set of procedures, which convert the electronic documents into a canonical form. This entails at least the following sorts of operations:

a) standardization of character encodings with regard to case, font, character set, etc.
b) recognition of word boundaries (by spacing, punctuation, hyphenation and quotation devices, etc.,
c) removal of formatting symbols (so-called 'control sequences'), and
d) transformation of modified ASCII to standard ASCII notation.

We may best illustrate these operations by example.

The **TALISMAN** prototype was specifically written for WORDSTAR documents because of our familiarity with the software, and its enormous user-population. Since WORDSTAR only supports one font and character set, a) was accomplished easily: only case conversion was involved. b) was accomplished with similar ease. We define words to be character strings bounded by spaces (hex 20), punctuation symbols ('.', '?', '!', ',', ';', ':') or other grammatical devices (single and double quotation marks, parentheses, brackets and braces, special characters, etc.). Thus, the removal of these characters is straightforward for the most part. The exceptions involve single quotation marks which identify contractions, and hyphens which signify compound words. These are handled with slightly greater finesse.

One type of formatting symbol is the embedded toggle. Examples include boldface (hex 02), underscore (hex 13), sub- and super-scripts (hex 16 and 14, respectively), and so forth. Another type is the 'dot' command which is essentially a 'command line' a la' JCL. A typical example is '.LH <ARG>' (hex 2E 4C 48 <ARG>), for control of line height. Similar controls are available for paper length, character width, page offset, margins, heading/footing identifiers, and page breaks.

Another category of format symbol is a fill-character. To illustrate, multiple-line spacing is accomplished by padding the end-of-line marker with multiple <CR>+<LF> (hex OD OA). Text centering is achieved by inserting the appropriate number of spaces (hex 20) before the first text character. Left margin control involves padding the first n spaces of each line with character hex AO, and so forth.

The last category of format symbol involves the transformation of a modified ASCII string to its alpha-numeric equivalent. Like many word processing packages, Wordstar uses a high-order bit shift for control. For example, soft carriage returns are identified by hex 8D OA, rather than hex OD OA for hard carriage returns. Similarly, a 'soft line' (e.g., one which is alterable under formatting) is distinguished from a 'hard line' by setting the high order bit high for the last byte of every word in the line.

Spelling Verification

The spelling verification component of a spelling assistance package attempts to verify that a target word in the electronic document is correct. If this is confirmed, attention is directed to the next word. Only upon failure is the correction component invoked.

Spelling verification can be conducted in two basic ways: deterministically and probabilistically. In a deterministic approach, the lexicon is consulted directly for each target. This involves standard techniques for lexical searching, and is usually performed with sublinear algorithms [4][7]. An alternative is to estimate the likelihood that the target is misspelled. If the probability exceeds some threshold, the target is turned over to the correction procedures. If not, it is assumed to be correct. A paradigm case of this approach is constituent analysis [18]. Naturally, both approaches have strengths and weaknesses: the deterministic approach is generally more accurate while the probabilistic approach is potentially faster.

Currently, **TALISMAN** uses the deterministic approach. As is common for spelling assistance software, the searches are defined over a hierarchy of lexicons: a common-word lexicon retained in primary memory, a main lexicon which resides on secondary storage and a user-defined dictionary which may be modified at any time. At this time, the common-word lexicon consists of the first 100 words of the "Brown Corpus" [8], and the main lexicon consists of a 10,000 word list developed in our lab. Our verification routine, written in C, consults the primary-resident dictionary first, and only upon failure consults the main and then user-defined dictionaries.

Spelling Correction

The 'business part' of **TALISMAN** and the part which distinguishes it from more conventional spelling assistance packages is the correction routine. This procedure, written in Prolog, directly encodes set-theoretical descriptions of error types into the program. Since no similarity measures are used, the method results in the generation of a set of alternative words which are related to the target in exactly the way intended.

While a thorough description of the corrector is beyond the scope of this paper, and has been dealt with elsewhere [3], a brief overview will provide a fuller understanding of the uniqueness of **TALISMAN**. Assume, for the moment, that one were interested in including in the list of suggested alternative words for a purported misspelling those words which were different by having one more character. Let's define these words extensionally by the set O (for Omission of a character). Let s be an arbitrary string and y be a character. We extensionally define the set, O, as the set of character strings similar to $c_1c_2c_3...c_{n-1}c_n$ such that

$$s \in O \leftrightarrow (\exists y)(s=((yc_1c_2c_3...c_{n-1}c_n) \vee (c_1yc_2c_3...c_{n-1}c_n) \vee ... \vee (c_1c_2c_3...c_{n-1}c_ny).$$

Given this set-theoretical description, the conversion into a logic programming language (in our case, Prolog) is trivial. The resulting clause set is

```
is_an_element_of_set_O(X):-
    adding_a_character(X,Y),
    legitimate_word(Y).
where
    adding_a_character(X,[Y|X]).
    adding_a_character([U|V],[U|W]):-
        adding_a_character(V,W).
```

This clause set reads as follows: a string, X, is an element of the set in question (namely O) if the result of adding a character to the string is itself a legitimate word. Adding a character is then defined recursively, using Prolog list notation. The result is that a string is an element of the set just in case it is both one character longer and a legitimate word, which is the literal description of character omission.

The mechanism by which the corrector works is the resolution/unification inference engine internal to Prolog. In this way, a suggested alternative to a misspelled word is found to be a theorem of the union of the lexicon with a set of 'corruption rules' like the one above. This is the most direct and effective way to approach the problem that we know of.

As we mentioned above, the current interest is with typing errors of the Damerau type. This enables us to make direct performance comparisons between **TALISMAN** and other current products, for most such products emphasize typing errors in their design. However, the only limitation on the nature of the errors to be detected is that they be expressible in first order logic. This, in effect, accommodates all error types which can be unambiguously expressed.

Lexical Organization:

As we reported in an earlier article [3], perhaps the greatest single difficulty in using Prolog in approximate string matching is the inefficiency with which it accesses the clause database. Frequently, the clauses are organized by predicate name and arity. Occasionally, indexing takes into account first argument as well. Thus, we suggested that the lexical database be organized in a length-segmented fashion, with words stored alphabetically within each segment (compilers typically perform this automatically when organizing internal databases). In the worst case, each segment must be searched serially for each match. In the best case, each subset of the segment with the desired character in the first position must be searched serially. In either case, several hundred patterns may be compared prior to a match. As we noted, this approach is not viable.

At first glance, one might consider representing the words themselves as predicates of zero arity. In this case, the entire lexical database would be completely indexed, and amenable to sub-linear search procedures. Since the logic of our spelling correction requires that each word-token be treated as a list containing characters, a simple re-conversion by means of the 'name' predicate would be necessary prior to search. The problem is that with the exception of the transformation test, the search targets contain uninstantiated character variables, and a predicate containing uninstantiated variables is undefined in Prolog. Thus, the solution must lie elsewhere, if it exists.

Two file access techniques which are available on some compilers are hashing and B-trees. Since search targets contain uninstantiated variables, hashing seems unrealistic. For it to work, one would need hash tables for all character positions, individually. We felt that greater promise would be offered by B-trees. Since lexical databases are essentially static in this application, we might take advantage of the search characteristics of B-trees while avoiding the update overhead.

B-trees of order m are multiway search trees which satisfy the following properties:

1. Every node has $\leq m$ children,
2. Every node but the root and terminal have $\geq \lceil m/2 \rceil$ children,
3. a non-terminal root has at least 2 children,
4. Terminal nodes appear on the same level, and have no information, and
5. An internal node with k children contains k-1 key values.

The efficiency of searching B-trees is directly related to their order. The maximum number of nodes which must be traversed is $K \leq 1+\log \lceil m/2 \rceil((n+1)/2)$, where n is the number of key values in the tree. Thus, for a B-tree of order 256 defined over a 100,000 word lexicon, the maximum search length is 4.

The Arity Prolog compiler supports a modified form of B-trees (condition 4, above, is relaxed to allow up to nine data elements on leaves), but regrettably restricts the order to three. Despite this limitation, a minor increase in performance was expected over the length-segmented list approach. This was not what we found.

In order to test the B-tree approach, we created a text file of 95 misspelled words, corrupted by transposition of two adjacent characters, insertion of an unwanted character, omission of a character, and substitution of a wrong character (see Appendix 1). The errors were distributed according to the ratio 8:20:33:34, respectively, which represents the average of the distributions of the two error sources reported by Peterson [11]. This text file was then 'corrected' by two versions of our Prolog spelling checker against our 9,734-word test lexicon. The first version organized the lexical database by word length first, and then alphabetically. The second version used sixteen B-trees (one for each word length), each with order 3.

The results were as follows: the source code and database for the length-segmented database compiled to 590kb and executed in 34.8 minutes,while the B-tree and program compiled 790kb and executed in 39.6 minutes, on an IBM PC/AT with a 6 MHZ clock, 640kb of TPA and a 3.5mb RAM disk for virtual memory. Thus, it became clear that the overhead associated with B-tree organization outweighs the minor advantage in search efficiency. This could be due to any number of factors. First, it is quite likely that the presence of uninstantiated variables in keys may undermine the indexing advantage by forcing a large number of serial searches. This might be overcome by using multiple indices. However, we felt that the additional space overhead would have been prohibitive, and therefore this direction was not explored. Second, the terms stored in the B-tree were lists of characters rather than atoms or unit clauses, which may introduce a complicated pointer structure with corresponding performance degradation. Third, the increase in the size of the database may cause thrashing due to a lack of spatial locality. Or, perhaps some combination of the above becomes pathological in approximate string matching contexts. Since we do not know how B-trees are handled in the compiler, there is no way to determine the precise cause of the performance degradation. For whatever reason, B-trees proved to be ineffective in lexical organization, at least as far as this product is concerned.

Having exhausted the lexical organization options within Prolog, it was decided that we would explore the possibility of interfacing the Prolog program with a high-level language which would handle the lexical organization. For this purpose, we chose Microsoft C, version 4.0. We thought that by judicious use of filters which would reduce the number of searches per target, we might increase the efficiency of **TALISMAN** by an order of magnitude. The task of developing the interface was undertaken by one of our associates, and the results are reported in [13].

The general approach was to use n-grams analysis on the lexicon to restrict searches to only those strings which were likely to be found. That is, if a transformation on a word target resulted in word token which contained a n-gram which does not appear in the lexicon, no search would be conducted to determine whether it was a legitimate word. In this application we felt that trigrams would offer the best balance between precision and efficiency (cf.,[1] [18]).

We represented the trigram data from the lexicon by means of 'Boolean Cubes'. A Boolean Cube is a three dimensional bit array where each axis corresponds to a position in the trigrams to be tested. The bit is '1' or '0' according to whether the corresponding n-gram appears, or fails to appear, anywhere in the lexicon. Thus, when one of the logical transformations (i.e., transposition, omission, etc.) of the word target is tested, the Boolean cube is consulted. If the 'massaged' word target is not inconsistent with the values within the cube, one or more searches is conducted, else no search results.

Perhaps it is easiest to explain the operation of the n-gram filter by way of an example. Suppose that the current test is for the accidental omission of a character within the string $c_1...c_k$. The search will be for instances of $Xc_1...c_k$ through $c_1...c_kX$, where X is a currently uninstantiated variable. Obviously, were one to search for all instantiations of X in all positions a great deal of time would be wasted, for only 15% of all possible trigrams actually occur in the lexicon. We overcome this inefficiency by first consulting the Boolean Cube to determine the instantiations which have any chance of success. One may think of the uninstantiated variable as designating up to three vectors in the cube as it changes from position one to three in the corresponding trigrams.

For testing purposes, we organized the lexicon as length-segmented lists, with lexical data stored alphabetically within each segment. Database access was a simple binary search within the appropriate segments. Our goal of improving the performance of **TALISMAN** by an order of magnitude was realized at least with respect to the sample data discussed above. The 95-word text file (Appendix 1) was processed in under 2 minutes. To place this in perspective, version 1.4 of The Word required 5 minutes 49 seconds, and version 5.0 of Office Speller (using the new Proximity Technology PF474 algorithm [14][15]) took 4 minutes and 11 seconds. Thus even at the early prototype stage, the performance of **TALISMAN** was promising. It is important to mention that the current implementation **TALISMAN** uses a considerably smaller lexicon that either of the two other products (The Word and Office Speller both use lexicons with more that 50,000 words). However, since the number of seeks is constant, irrespective of database size, and since the accessing is performed in sublinear time, even a ten-fold increase in the size of the lexicon will only contribute a 25% decrease in performance. This may eventually be offset by additional tuning of our trigram filters.

Accuracy

In order to demonstrate the accuracy of **TALISMAN**, we call the reader's attention to the list of alternatives which appear in the Appendix. In all cases, when an alternative is suggested by The Word but not by **TALISMAN**, the missing word was not in the latter's dictionary. The converse was not the case. The entries for 'XHAIRS', 'HTE', 'ERBS' and 'XEBRA clearly show the The Word relies upon the correctness of the first few characters of the target for entry into the database. **TALISMAN** presents no such limitation. It should also be mentioned that the current, test-version of **TALISMAN** is specifically designed with the Damerau conditions in mind so that its performance may be compared with other products. Logic-based approaches to spelling correction are easily and precisely extended to non-Damerau relations, while it isn't at all clear how this would be handled in products which define similarity by similarity measures or metrics (see ref. [3] for further discussion).

In addition, since TALISMAN directly encodes the definitions of the spelling corruptions to be considered into the program, there is precise control over the nature of the alternatives. Listed below are typical alternative spellings, in the order in which they are suggested, for our word list provided by Office Speller:

1. massachusets	{massachusetts,masochists,masochistic, musicologist,musicologists, misallocation,mastications, methodistic}
2. oxegen	{oxygen,hexagon,exiting,exuding, exigent,exogamy,hexagons,oxygenic}
3. hezitate	{hesitate,acetate,esthetic,aesthetic, atheletic,elucidate,housecoat,hesitated}
4. xebra	{zebra,subway,sabre,sober,siberia, super,soberly}.

A brief reading will show that the approximation metrics in use are far too broad. Specifically, it is hard to imagine a possible world in which 'oxegen' is a more likely misspelling of 'exciting' than 'oxygenic' is. Similarly, 'massachusets' is not likely to be a misspelling of 'misallocation' or 'musicologist'; nor is 'hezitate' a likely misspelling of 'housecoat', nor 'xebra' of 'subway' or 'siberia'. This indicates the inherent weaknesses of using approximations. In the above cases, at least, the first alternative is a reasonable one. However, the latter alternatives should make us suspicious that the approximation technique is a bit crude. Our suspicion is confirmed for the following two words:

5.xhairs {theirs,Thurs.,sailers,sailors,thirst,saris}

6.nieghbor {newsboy,newsboys}

In these cases, the breadth of the metric is not the only problem: the metric simply doesn't work at all. The spelling corruptions are among the most common of all errors: basic Damerau typing errors resulting from the substitution of a character by an adjacent one on the keyboard and a simple transposition, and yet they go undetected. Similar problems occur with Turbo Lightning: 'xhairs' suggests 'x-axis', 'x-rays' and 'Xmas', while 'sychological' generates the alternatives 'schismatically', 'scholastic', 'schoolmarm', 'schoolmistress', 'schoolmate' and 'schoolteachers', none of which are orthographically close to the corrupted target. The main problem of approximation techniques, that they must be continually 'tuned' and are never exactly 'right', is entirely avoided by our approach.

Operational Characteristics

TALISMAN is currently designed for the IBM/PC family of microcomputers, running under PC-DOS 2.X or above, which have at least 512kb or primary. The operational characteristics are as follows.

The main screen is partitioned into eight windows which contain the status line, the target window which presents the purported misspelling, the context window which offers the context for the current token of the target which is taken from the document, the alternatives window with candidates for the intended words, the help window and three windows for location pointers. The functional description is easiest to understand in the context of Figure 1.

The status line contains four sub-windows for toggle status. SCOPE refers to the extent to which any correction of the target will affect occurrences (or tokens) of the target within the document. The options are ALL and ONE. Since there are 2 occurrences of 'AETS' in the document, a correction will affect both in our illustration.

DICT:OFF indicated that the dictionary will only be consulted upon request (<SHIFT>-<F2>). The alternative is to have alternatives automatically generated for all targets.

The UPDATE toggle determines whether the contents of the target window will be inserted into the user's dictionary. INSERT has the same effect as in a word processor, allowing the user to edit the target (e.g., in the case of an accidental omission of a character).

The three location pointers relate the screen display to the electronic document. The target pointer tells us that there were 34 putative misspellings found, of which 'AETS' is number 1. The context pointer indicates that this is the first of two occurrences of this token. The alternatives pointer shows 'NETS' to be the fifth of nine alternatives suggested.

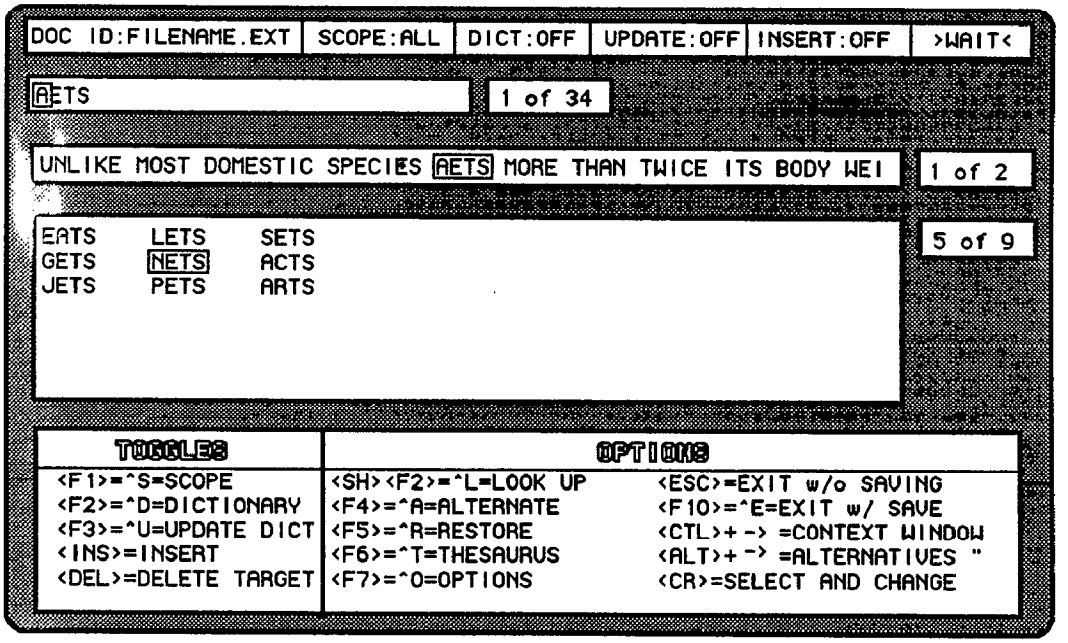

FIGURE 1

Conclusion

We have described in general terms the basic operation of TALISMAN, and contrasted it with more conventional approaches. A basic understanding of both the nature of the spelling correction system and of spelling errors illustrates the effectiveness of this approach. However, the efficiency of the approach remained in doubt until recently. We are now able to forecast, with considerable confidence, that the logic-based system will provide at least the level of performance enjoyed by conventional approaches, while at the same time affording a far more accurate solution to the problem.

APPENDIX

Misspellings	Alternatives (TALISMAN)	Alternatives (THE WORD
Rekon	Reckon	Reckon
Cotninent	Continent	Continent
Dreamd	Dreamed,Dread, Dream,Dreams,Dreamy	Dread,Dream,Dreamed, Dreams,Dreamt,Dreamy
Enjoyible	Enjoyable	Enjoyable
Dictater	Dictate,Dictator	Dictate,Dictated,Dictates, Dictator
Absolutly	Absolutely	Absolutely
Flameing	Flaming	Flaming
Drible	Dribble	Dribble,Driblet
Estimats	Estimate,Estimates	Estimate,Estimates
Absense	Absence	Absence
Responsability	Responsibility	Responsibility
Carpentur	Carpenter	Carpenter
Vareful	Careful	no similar word found
Oiley	Oily	Oiled,Oiler,Oily
Jeepp	Jeep	Jeep
Poum	Plum,Poem,Pour	Plum,Poem,Pour,Pout
Xhairs	Hairs,Chairs	no similar word found
Ect	Eat,Etc,Act	Eat
Norhtern	Northern	Northern
Ebing	Being,Ebbing	Ebbing,Eying
Minig	Mini,Mining	Mini,Mining
Quarrul	Quarrel	Quarrel
Nonsence	Nonsense	Nonsense
Pited	Fitted,Fated,Fired,Fixed	Fated,Feted,Filed,Fined Fired,Fisted,Fitted,Fixed
Potatoe	Potato	Potato,Potatoes
Emploied	Employed	Employed
Skillul	Skillful	Skillful
Herold	Herald	Harold,Herald
Jelous	Jealous	Jealous
Shoare	Share,Shore	Share,Shore
Gutso	Gusto	Gusto,Guts
Prarie	Prairie	Prairie
Mardhes	Marshes	Marches,Marshes
Fragrent	Fragrant	Fragment,Fragrant
Sychological	Psychological	no similar word found
Plundged	Plunged	Plunged
Evaseion	Evasion	Evasion
Providance	Providence	Providence
Millons	Millions	Millions
Hte	The,Hate,He,Ate,Hoe,Hue	Hate,He,Hie,Hoe,Hue
Absorbe	Absorbed,Absorb Absorb,Absorbed,Absorbs	
Onored	Honored	no similar word found
Eateable	Eatable	Eatable
Basicaly	Basically	Basically
Oxegen	Oxygen	Oxygen
Gluist	Gluiest	no similar word found
Dyiing	Dying	Dying
Cheecked	Checked	Checked
Practise	Practice	Practice
Triped	Striped,Tripped,Tried, Tripod	Tried,Tripe,Tripled, Tripod,Tripp
Massachusets	Massachusetts	Massachusetts
Abundence	Abundance	Abundance
Fixeable	Fixable	Fixable
Hezitate	Hesitate	Hesitate
Godess	Goddess	Goddess,Godless
Pivit	Pivot	Pivot
Mous	Mouse,Moss	Mobs,Modus,Moos, Mops,Moss,Mouse, Mousy
Diagramm	Diagram,Diagrams	Diagram,Diagrams
Erbs	Herbs,Verbs	Ebbs,Eras,Eros,Errs
Filamint	Filament	Filament

REFERENCES

[1] Angell, R., G. Freund and P. Willett, "Automatic Spelling Correction using a Trigram Similarity Measure", Information Processing and Management, Vol 19, pp. 255-261 (1983).

[2] Berghel, H., "Extending the Capabilities of Word Processing Software through Horn Clause Lexical Databases", Proceedings of the 1986 National Computing Conference, AFIPS Press, Reston, pp. 251-257 (1986).

[3] Berghel, H., "A Logical Framework for the Correction of Spelling Errors in Electronic Documents", Information Processing and Management, Vol. 23, No. 5, pp. 477-494 (1987).

[4] Boyer, R. and J. Moore, "A Fast String searching Algorithm", Communications of the ACM, Vol. 20, pp. 762-772 (1977).

[5] Damerau, F., "A Technique for Computer Detection and Correction of Spelling Errors", Communications of the ACM, Vol. 7, pp. 171-176 (164).

[6] Durham, I., D. Lamb and J. Saxe, "Spelling Correction in Use Interfaces", Communications of the ACM, Vol. 26, pp. 764-773 (1983).

[7] Knuth, D., J. Morris and V. Pratt, "Fast Pattern Matching in Strings", SIAM Journal on Computing, Vol. 6, pp. 323-350 (1977).

[8] Kucera, H. and W. Francis, Computational Analysis of Present-Day American English, Brown University Press, Providence (1967).

[9] Morgan, H., "Spelling Correction in Systems Programs", Communications of the ACM, Vol. 13, pp. 90-94 (1970).

[10] Peterson, J., "Computer Programs for Detecting and Correcting Spelling Errors", Communications of the ACM, Vol. 23, pp. 676-687 (1980).

[11] Peterson, J., "A Note on Undetected Typing Errors", Communications of the ACM, Vol. 29, pp. 633-637 (1986).

[12] Pollock, J. and A. Zamora, "Automatic Spelling Correction in Scientific and Scholarly Text", Communications of the ACM, Vol. 27, pp. 358-368 (1984).

[13] Rankin, R. "Increasing the Efficiency of Prolog Lexical Databases with N-Gram Boolean Cubes", Technical Report # TR-87-14, Department of Computer Science University of Arkansas (1987).

[14] Rosenthal, S., "The PF474", Byte, pp. 247-256 (November, 1984).

[15] Taylor, D., "Wordz that Almost Match", Computer Language, pp. 47-59, (November, 1986).

[16] Salton, G., Introduction to Modern Information Retrieval, McGraw-Hill, New York (1983).

[17] Shaffer, L. and J. Hardwick, "Typing Performance as a Function of Text", Quarterly Journal of Experimental Psychology, Vol. 20, pp. 360-369 (1968).

[18] Zamora, E., J. Pollock, and A. Zamora, "The Use of Trigram Analysis for Spelling Error Detection", Information Processing and Management, Vol. 17, pp. 305-316 (1981).

Skelaton	Skeleton	Skeleton
Abor	Labor,Abhor	Abort,Abhor,Arbor
Reefe	Reef	Reef,Reefs,Reeve
Nervey	Nerve,Nerves,Nervy	Nerve,Nerves,Nervy
Ofen	Often,Open,Oven	Often,Omen,Open,Oren, Oven,Owen,Oxen
Nieghbor	Neighbor	Neighbor
Ecco	Echo	Echo
Inword	Inward	Inward
Wenesday	Wednesday	Wednesday
Eaternal	Eternal	Eternal,External
Gipsy	Gypsy	Gypsy
Maried	Married,Varied,Marked	Marie,Marked,Marred, Married
Nomanal	Nominal	Nominal
Overwelm	Overwhelm	Overwhelm
Rustleing	Rustling	Rustling
Blueish	Bluish	Bluefish,Bluish
Hunerd	Hundred	Hundred
Eroll	Enroll	Enroll
Kerasene	Kerosene	Kerosene
Gymnasium	Gymnasium	Gymnasium
Hony	Horny,Honey,Bony,Pony, Holy	Holy,Hone,Honey,Hong, Honk,Horney,Hoy
Basaar	Bazaar	Bazaar
Herosm	Heroism	Heroism
Xebra	Zebra	no similar word found
Eah	Each,Ear	Each,Ear,Eat,Eh
Weazel	Weasel	Weasel
Memit	Emit,Remit	Merit
Libary	Library	Library
Aiets	Eats,Gets,Jets,Lets,Nets, Pets,Sets,Acts,Arts	Acts,Ants,Arts
Alabi	Alibi	Alibi
Dinasy	Dynasty	Dynasty
Forths	Fourths,Forth,Forts	Forth,Forts
Trouzers	Trousers	Trouper,Trouser
Artic	Arctic	Arctic
Allyes	Allies	Alleys,Allies