Automatic detecting/correcting errors in Chinese text by an approximate word-matching algorithm

Lei Zhang
Dept. of Computer Science and Technology
Tsinghua University
Beijing, China, 100084
zhl@s1000e.cs.tsinghua.edu.cn

Ming Zhou
Microsoft Research China
Beijing, China, 100080
mingzhou@microsoft.com

Changning Huang
Microsoft Research China
Beijing, China, 100080
cnhuang@microsoft.com

Haihua Pan
Dept. of Chinese, Translation and Linguistics
City University of Hong Kong
Kowlong, Hong Kong, China
cthpan@cityu.edu.hk

Abstract

An approximate word-matching algorithm for Chinese is presented. Based on this algorithm, an effective approach to Chinese spelling error detection and correction is implemented. With a word tri-gram language model, the optimal string is searched from all possible derivation of the input sentence using operations of character substitution, insertion, and deletion. Comparing the original sentence with the optimal string, spelling error detection and correction is realized simultaneously.

Introduction

No system aiming at automatic detecting and correcting errors in Chinese text achieves satisfying result today. One representative approach is confusing character substitution method (Chang, 1994), where confusing characters are used to replace every character in the input sentence, and a "correct" result with highest evaluation score is searched from all paths. While achieving relatively good result, it has obvious weakness: only character substitution errors can be detected and corrected, other kinds of errors can not be handled, including character deletion, character insertion, and string substitution errors.

There is a clear two-level structure in English spelling error detection and correction, "non-word" error and "real-word" error (Kukich, 1992). Things are different in Chinese. Although many approaches find that most errors in Chinese cause segmentation abnormal (Sun, 1997 and Zhang, 1998), no one stress on such "non-word" error and the two-level structure is not adopted in Chinese. Following are possible reasons for this situation. There is no obvious word boundary in Chinese text, so automatic word segmentation must be introduced. If error exists, segmentation result could be weird. For example, the segmentation of 忠耿耿, which is a character deletion error for the word 忠心耿耿, may appear like 忠/耿/耿, including three one-character words. It's easy for human to judge that 忠耿耿 is some kind of "non-word" error, but it's difficult for computer to make such a decision because most Chinese characters could be used as a one-character word.

A fast approximate Chinese word-matching algorithm is presented. Based on this algorithm, a new automatic error detection and correction approach using confusing word substitution is implemented. Compared with the approach of (Chang, 94), its distinguished feature is that not only character substitution error, but also character insertion or deletion error and string substitution error could be handled.

1 Fast approximate Chinese word-matching algorithm

1.1 Error types in Chinese text

When classifying errors in Chinese text, most papers prefer to categorize errors on the character level. In our opinion, this kind of classification contributes little to improve the performance for Chinese text error detection and correction. Referring to what's common in English spelling error detection and correction, we classify errors as followed:

⑴Non-word errors: The string mapping to a word in the correct text can't be treated as a word in the corresponding error text. This kind of error can be further classified into: ① Character substitution error. A correct character is replaced by another character. Such as 牺牲品 → 牺牲吕, 一鸣惊人 → 一呜惊人. ② String substitution error. A correct string is replaced by another string, and at least one of the two strings consists of more than one character. Such as 实事求是 → 实施求是. ③ Character insertion error. Such as 惊天动地 → 惊天天动地. ④Character deletion error. Such as 忠心耿耿 → 忠耿耿.

⑵Real-word errors. This kind of error could be further classified into: ① Word substitution error. Such as 统治 → 通知. ② Word insertion error. Such as 基于 → 基于基于. ③Word-Deletion error. Such as 一条龙 → 一龙.

1.2 Fast precise Chinese word-matching algorithm

Dictionary organized in (deterministic) *finite state recognizer* (*FSR*) format is used to implement the fast precise Chinese word-matching algorithm. *FSR* = (Q, A, δ, F) with Q denoting the set of states, A denoting the input alphabet, which are all Chinese characters for a Chinese dictionary, $\delta : Q\times A\rightarrow Q$ denoting the state transition function and $F \subseteq Q$ denoting the final states. Besides, let *stStart* denote the starting state of a *FSR*, *stError* denote the state when input character cannot be accepted.

Precise Chinese word match is to find all the strings, starting from a specific position in a sentence, which are items in a dictionary. For example, with the dictionary including words: 中、中国、中国人、国人、人、人民, at the beginning of the sentence "中国人民站起来了！", three words of different length "中", "中国", "中国人" should be matched. Figure 1 shows the algorithm of fast precise Chinese word match. Where *dict*=(Q, A, δ, F) is a Chinese dictionary, *state* is its current state, *str* is the string currently read by *dict*, *sentence* is the input sentence, *idx* is the subscript in the sentence of the character that should be read immediately. All matched words are put into *result*. The initial values when calling this recursive function should assure: *str=nil, state=stStart, result*=∅. This function is used by approximate word-matching algorithm later.

1.3 Approximate Chinese word-matching algorithm

Approximate word match based-on *FSR* is used in English spelling error detection and correction to find all words in a dictionary whose minimum edit distance to a given string is less than a threshold. However, there are great differences between the approximate word match of Chinese and English. First, the length of the English string to be matched is determinate. Because there are no obvious word boundaries in Chinese text, the length of the Chinese string to be matched is unknown. So approximate Chinese word match must find words that are similar to all strings of different length. For example, when approximately matching the sentence "展览会举办得很成功" at its beginning, the algorithm should give out all Chinese words that are similar to "展", "展览", "展览会", "展览会举", "展览会举办"…

```
Procedure CnPreciseMatch(dfa, state, str, sentence, idx, Var result )
begin
  if ( idx is not the end of sentence) and ((state'←δ(state,sentence[idx])) ≠ stError) then
  begin
      if ( state'∈ F ) then
          result←result ∪ { str + sentence[idx] };
      CnPreciseMatch( dfa, state', str+sentence[idx], sentence, idx+1, result );
  end;
end;
```

Figure 1. Fast precise Chinese word-matching algorithm

Second, character is the basic unit in the English, and its approximate word-matching algorithm can adopt a cut-off method (Oflazer, 1996) depending on the string currently read and only one character to be read next time. This decreases the search space greatly. As mentioned before, errors in Chinese text may be caused by string replacement where the lengths of the correct string and its corresponding error string may be different. This means, no matter how far the *FSR* has read from the beginning position, how dissimilar the string currently read is to all possible words, we can't determine whether or not words that are similar to the target string could be found if more characters are read later. For example, the error 逃之夭夭 → 逃之夭乘风破浪, which is caused because the Five-stroke input code are similar between 夭(tdi) and 乘风破浪(tmdi). Although 逃之夭乘风破 is dissimilar to all words in dictionary, 逃之夭乘风破浪 is similar to word 逃之夭夭.

The direct way to implement the approximate Chinese word match is: for each substring with length 1, 2, 3, …, N, starting at specified position in sentence, browse the dictionary and count their similarity or edit distance to every word in it. This approach will face two problems: ① High computational complexity. A common Chinese dictionary with normal size contains about 60,000 words. Besides, the evaluation of the similarity between two strings is also time-consuming. Its computing cost is far from the real-time requirement of error detection and correction. ②What is the maximum substring length N? One may think that 4 or 5 is enough. Let's look at an extraordinary example. Imaging an inexperienced Five-Stroke typewriter is supposed to input a four-character word and he makes mistakes on every character. More unfortunately, his carelessness causes every single character transformed to a four-character word. Finally he gets a 16-character string instead of the expected four-character word. Here, N should be at least 16 to get the proper match.

1.3.1 Definition of the distance

To implement approximate match, distance of two strings should be defined. Some predefined distances of two strings is called meta-distance, denoted with *MetaD*. We define *MetaD*(*X*,*Y*) between two Chinese strings *X* and *Y* according to their grapheme, pronunciation and input code: ①The similarity of their Pinyin input code. For characters *X* and *Y* with same pronunciation, for example 同(tong) and 通(tong), *MetaD*(*X*,*Y*)=30. For characters *X* and *Y* different only on surd/sonant, for example 删(shan) and 三(san), *MetaD*(*X*,*Y*)=40. ② The similarity of their Five-Stroke input code. For strings *X* and *Y* that one input code could be transformed to another at most by one substitution operation, for example牙刷(ahnm) and 工具书(ahnn), *MetaD*(*X*,*Y*)=30. For strings *X* and *Y* that one input code could be transformed to another only by one insertion, deletion or transposition operation, for example 式(aa) and 工(aaa), 阶(bwj) and 创(wbj), *MetaD*(*X*,*Y*)=40. ③ Some rules learned from text errors reflecting common human confusion on Chinese characters. For example, *MetaD*(菅, 管)=26. In addition, let ε mean null string, for each character z, let *MetaD*(ε,ε) = *MetaD*(z, z) = 0, *MetaD*(z, ε) = *Metad*(ε, z) = 50. For those string pairs *X* and *Y* that there is no meta-distance definition between them, just let *MetaD*(*X*,*Y*)=+ ∞. Define the set of meta-strings *M* as:

$$M = \{ X \mid X \neq \varepsilon, \exists Y \neq X, MetaD(X, Y) < +\infty \}$$

For each meta-string $X \in M$, define its confusing string set *cfs*(*X*) as:

$$cfs(X) = \{ Y \mid Y \in M, Y \neq \varepsilon, MetaD(X, Y) < +\infty \}$$

And each $Y \in cfs(X)$ is called confusing string of *X*. When character transposition error[*] is not considered, the distance of two strings X_1^m and Y_1^n, where m and n is the length of X and Y respectively, can be defined as:

$$d(X_1^m, Y_1^n) = \begin{cases} MetaD(X_1^m, Y_1^n) \ldots\ldots\ldots\ldots MetaD(X_1^m, Y_1^n) < +\infty \\ \underset{1 \le i \le m, 1 \le j \le n}{MIN} \{ d(X_1^i, Y_1^j) + d(X_{i+1}^m, Y_{j+1}^n) \} \ldots\ldots other \end{cases}$$

This recursive definition is not easy to compute. Fortunately, our approximate match algorithm use a heuristic expanding method and avoid the computing cost.

[*] Character transposition error is rare in Chinese text.

1.3.2 Fast approximate Chinese word-matching algorithm

Chinese word approximate match is to find, from a specific position in the target sentence, all words that has a distance less than a threshold t_w to substrings beginning at the position with different length 1, 2, 3, …. At the same time, the beginning position of the match next time and the distance *d* between the matched word and its corresponding original string should also be given. In our implementation, the set of meta-string *M* is also organized in a *FSR* format. In *M*'s every final state representing a meta-string *X*, all strings in *cfs*(*X*) and their distances to *X* is recorded. Figure 2 shows the approximate Chinese word-matching algorithm. Where *cdfa*=(Q', A', δ', F') is *M*. *dict*=(Q, A, δ, F) is Chinese dictionary. *state* is the current state of *dict*. *str* is the string currently read by *dict*. *sentence* is the target sentence. *idx* is the subscript in the sentence of the character to be read next time. *diff* is the distance of two partial strings already matched. *result* is a set of 3-tuple elements like (*word, next, d*), where *word* is the approximate matched word, *next* is the position where the approximate match should start next time. *d* is the distance between the matched word and its corresponding original string. The initial values when calling this algorithm should assure: *str*=*nil*, *state*=*stStart*, *diff*=0, *result*=∅. The algorithm could restore character insertion, deletion, substitution errors as well as string substitution errors. The threshold t_w decreases the search space.

For example, when to approximate match at the beginning of "展临会举办得很成功。", where 临 is a character substitution error of 览, the result may be contains (展, 2, 0), (站, 2, 30), (盏, 2, 30), (民航, 2, 40), …, (展览, 3, 0), (展览会, 4, 0), (展望, 2, 50), (展出, 2, 50),…, (招展, 2, 50), (参展, 2, 50)…, etc. The correct word 展览会 is in the list.

2 Confusing word substitution approach

Confusing character substitution approach (Chang, 1994) got a relatively good result, but can not deal with errors of character insertion, character deletion and string substitution. Our confusing word substitution approach is an improvement on the confusing character substitution approach by mending such disability. It is based on the fast approximate Chinese word-matching algorithm. In this approach, a given sentence is approximately segmented from all possible derivation of the input sentence using operations of substitution,

```
Procedure CnFussyMatch( dict, state, str, sentence, idx, diff, $t_w$, cdfa, Var result )
begin
    if ( idx is the end of sentence) then return;
    if ( diff+MetaD(sentence[idx],$\varepsilon$) $\le t_w$) then        //try to delete a Chinese character
        CnFussyMatch(dict, state, str, sentence, idx+1, diff+MetaD(sentence[idx], $\varepsilon$), $t_w$, cdfa, result);
    Foreach { x | (state'$\leftarrow\delta$(state, x)) $\neq$ stError } do        //try to insert a Chinese character
        if ( diff+MetaD(x, $\varepsilon$) $\le t_w$)  then
        begin
            if ( state'$\in$ F)then
                result  += { (str+x, idx, diff+MetaD(x, $\varepsilon$)) };
            CnFussyMatch(dict, state', str+x, sentence, idx, Diff+MetaD(x, $\varepsilon$), $t_w$, cdfa, result);
        end;
    CnPreciseMatch( cdfa, stStart, nil, sentence, idx, set=∅ );        //get all possible meta-strings into set
    Foreach { X | X$\in$ set } do
        Foreach { Y | Y$\in$ cfs(X) } do        //try to replace X with its similar string Y
            If ( diff+MetaD(X,Y)$\le t_w$) and ( (state'$\leftarrow\delta$(state, Y)) $\neq$ stError ) then
            begin
                if ( state'$\in$ F ) then
                    result += { (str+Y, idx+|X|, diff+MetaD(X,Y)) };
                CnFussyMatch(dict, state', str+ Y, sentence, idx+|X|, diff+MetaD(X, Y), $t_w$, cdfa, result);
            end;
end;
```

Figure 2. Fast approximate Chinese word-matching algorithm

```
Procedure CnFussySeg( dict, path, sentence, idx, diff, cdfa, Var result )
begin
    if（idx is the end of sentence）then
        result += path;
    else begin
        CnFussyMatch( dict, stStart, nil, sentence, idx, 0, min{ $t_s$-diff, $t_c$ }, cdfa, fussyWords=∅);
        foreach ( (word, next, d) in fussyWords ) do
            CnFussySeg( dict, path+{(word, next, d)}, sentence, next, diff+d, cdfa, result);
    end;
end;
```

Figure 3. Complete approximate segmentation algorithm

insertion, and deletion. Paths are then evaluated using base language model and distance discount. The optimal path with highest score is searched and treated as the correction of the original sentence.

2.1 Approximate segmentation

As what happens in precise segmentation, approximate segmentation is to give a segmentation path for a input sentence, but with error tolerant ability. On the path, each word is similar to its corresponding original string in the input sentence. That's why we call this confusing word substitution approach. For a input sentence, let threshold $t_s \geq t_w$. It's required that the sum of distances between all the words *W'* on an approximate segmentation path and their corresponding original string *W* can not be greater than t_s:

$$\sum_{W' \in path} d(W', W) \leq t_s$$

The reason for using t_s is that there are always little errors in one sentence and it could decrease the space of the approximate segmentation paths. Figure 3 shows the algorithm of the approximate segmentation listing all possible paths. Where the *path* is an approximate segmentation for input sentence, it's an array of elements like (*word, next, d*). *result* is the set of all possible approximate segmentation paths. Other symbols, such as *dict* and *cdfa,* are of same meaning as in section 1.3.2.

2.2 Path evaluation

The evaluation of paths consists basic language model evaluation and distance discount. For a given approximate segmentation path, basic language model evaluation can adopt N-Gram models of character, word, POS tag or word class. Denote the score with *ModelScore*(*path*). The distance discount multiplies the *ModelScore* with a discount, valued from 0 to 1, according to the distance between the *path* and the input sentence. The final score of a path *FS* is:

$$FS(path, sentence) = ModelScore(path) * discount(path, sentence)$$

In this paper, we use:

$$discount(path, sentence) = \prod_{W' \in path} f(l, n, d(W', W))$$

Where *l* is the length of *W'*, *n* is the number of the segment units when its corresponding original string *W* is segmented. *d*(*W', W*) is the distance between *W'* and *W*. *f* is the discount function. Generally, the value of *f* should be closer to 1.0 if *d* is less, *l* is longer or *n* is greater.

Because the number of possible paths in an approximate segmentation is very large, to avoid the computational complexity, dynamic programming is adopted and some changes are made to the segmentation algorithm.

3 Experiment and results

First define some evaluating indicator for the automatic error detection and correction task, let:

A=number of errors in target text.
B=number of warnings the proofreading approach given
C=number of errors the proofreading approach detected
D=number of errors the proofreading approach corrected

*	Character substitution error	String substitution error	Character deletion error	Character insertion error	Total	Number of warnings	Recall rate	Precise rate	Correction rate
Test text	395	76	29	35	535				
(Chang 1994)	315 / 297	17 / 0	3 / 0	6 / 0	341 / 297	622	64%	55%	56%
This approach	317 / 298	66 / 63	19 / 18	27 / 24	420 / 403	656	79%	64%	75%

*(number of error warned/ number of error corrected)

Figure 4. Error distribution and experiment result

Then, recall rate = C/A*100%, precise rate = C/B*100%, correction rate = D/A*100%.

In the experiment, a text containing 535 errors is to be detected and corrected. The corpus to train the word tri-gram language model is about 200M bytes, including people's daily 93 and 94, 10 years reader, BaiJiaBao'94, ShiChangBao'94. Two thresholds *ts* and *tw* are set 59 and 50 respectively. Zhang Zhaohuang's approach is also applied on the test text as a comparison. The distribution of different kinds of errors and the experiment results are shown in Figure 4.

From the result, we can see that our approach using similar word substitution has the same ability to detect and correct the character substitution errors as (Zhang 1994) approach. But its ability to detect and correct character insertion error, character deletion error and string substitution error are highly enhanced. The result shows that our approach has great practicability.

The reason for those incorrect warnings and undetected or uncorrected errors mainly on: ① Insufficient of the similar string set. When a correct string is not included in the similar string set, some errors will not be detected and corrected. For example, the error "大家同心协力"→"大学同心协力" did not detected in our experiment because "家" does not in the similar string set of "学". More complete similar string set will detect and correct more errors, yet they may also cause more incorrect warnings, and increase the computing cost. ②Language Mode deficiency. Tri-gram only has local linguistic constraints. It's necessary to adopt long-distance constraints. ③Incomplete of the dictionary. ④ Data sparseness. Larger corpus needed in training.

4 Conclusion

A fast approximate Chinese word-matching algorithm is presented in this paper. Based on the algorithm, an automatic Chinese Spelling Error Correction approach using similar word

Error sentences	Detection and Correction results
巨大的**后座力**震得人…	巨大的**后坐力**震得人…
一座可**按装**1 5 万门程控电话…	一座可**安装**1 5 万门程控电话
而外面的风经常是由**穿流不息**而过的车流或楼房林立的狭窄街道所形成。	而外面的风经常是由**川流不息**而过的车流或楼房林立的狭窄街道所形成。
你可别把它们**混为**一啊。	你可别把它们**混为一谈**啊。
认真查办了**骗取得**国家出口退税款犯罪案件。	认真查办了**骗取**国家出口退税款犯罪案件
动物饲养员须找出猴群之中的**顶点群之马**。	动物饲养员须找出猴群之中的**害群之马**。

Figure 5. Examples of the test result using our error detection and correction approach

substitution and language model evaluation is designed. Compared with Zhang Chao-Huang's confusing character substitution method, this new approach can deal with not only character substitution errors but also insertion, deletion and string substitution errors. Because no word boundaries in Chinese text, there is not a two-level structure of "non-word" and "real-word" errors in Chinese spelling correction task like that in English spelling correction. The fast approximate Chinese word-matching algorithm can handle Chinese "non-word" error efficiently, making it easy to establish a two-level structure in Chinese spelling correction.

The future research may include: ①Pruning the approximate word matching result before they take part in the approximate segmentation. This will decrease the computing cost. ② Introducing long distance constraints. What's need to point out is that dynamic programming may dislike this kind of long distance constrains. So they are more suitable in the pruning and discounting.

References

Chang Chao-Huang (1994) *A Pilot Study on Automatic Chinese Spelling Error Correction*. Communication of COLIPS, 4/2, pp. 143—149

Karen Kukich (1992) *Techniques for automatically correcting words in text*. ACM Computing Surveys, 24/4, pp. 377—439

Kemal Oflazer (1996) *Error-tolerant Finite-state Recognition with Applications to Morphological Analysis and Spelling Correction*. Computational Linguistics, Computational Linguistics, 22/1, pp. 73—89

Sun Cai (1997) *Research on Lexical Error Detection and Correction of Chinese Text: [Master Dissertation]*. Tsinghua University, Beijing. 96p

Zhang Yansen, Ding Bingqing (1998). *Research and Practice on the Lexical Error Detecting System Based on "Banding and Filtering" in Chinese Text Automatic Proofread*. In Proc. of International Conference on Chinese Information Processing, Tsinghua University Publishers, Beijing, China, pp. 392—437