Corpus-based Automatic Rule Selection in Designing a Grammar Checker

†Yuan-Ling Liu, †Shih-ping Wang
‡Keh-Yih Su

† Behavior Design Corporation
2F, 28, R&D Rd II
Science-based Industrial Park
Hsinchu, Taiwan 300, R.O.C.

‡ Department of Electrical Engineering
National Tsing-Hua University
Hsinchu, Taiwan 30043, R.O.C.

†wsp@bdc.com.tw

‡kysu@bdc.com.tw

Topic Area: Grammar Checker; Key Words: Corpus & Sequential Forward Selection (SFS)

ABSTRACT

In designing grammar-checking systems, the pattern matching algorithm, although failing to handle complex errors, is still widely adopted today. This is because when compared with the method of employing full scale parsing, pattern matching is efficient in detecting local errors with much less computer time and memory. However, the patterns used in the pattern matching approach are usually hand-tuned, and thus suffer from inadequacy in handling correlations among patterns. These error patterns may conflict or overlap with each other. Therefore, an automatic rule selection method, called ***Sequential Forward Selection (SFS)***, is proposed in this paper to tackle these problems. SFS uses objective performance measures to automatically search the suboptimal rule-set from all the possible combinations of rules. With SFS, the effectiveness of each rule can be measured, and problematic patterns can be identified systematically and efficiently for the linguist to fine-tune. Therefore, the error patterns can be revised efficiently. In our tests based on a corpus of 1956 sentences, the false rate decreases by 11.8% (from 26.4% to 14.8%) if the suboptimal rule set (81) selected by SFS is adopted, instead of the whole rule set (127). With this suboptimal rule set, the recognition rate decreases only by 3.9% (from 38.9% to 35%).

I. Introduction

The research of grammar checking has long been an attractive area in computational linguistics. Its main function is to detect grammatical mistakes. To detect grammatical errors, various algorithms have been proposed in the past decade, such as pattern matching (e.g., Kay, 1987), partial match

(Pfaltz et al., 1980), short range grammar verifications, syntactic parsing (cf. Vergne et al., 1986), etc. All these approaches can be generally classified into (1) ***pattern matching*** and (2) ***syntactic parsing***. Although the parsing method can solve the problem of long distance dependency at the syntactic level, it is time-consuming and occupies too much disk space. Moreover, parsing does not always yield the correct results. False alarms and missing error rates still exist in parsing. On the other hand, pattern matching is used to detect the grammatical errors without parsing (cf. Atwell, 1987). It can search the "desired shape" with local distance dependency. Although it usually fails to catch complex errors, many local grammatical errors still can be detected effectively. It can also save time and operating costs. Therefore, this approach is still largely adopted and currently used in our system, i.e., **B**ehavior **D**esign **C**orporation-Grammar Checker (hereafter, BDC-GC).

However, the patterns used in the pattern matching approach are usually hand-tuned, which suffer the following problems:

(1) It is not easy to manage the correlation among a bunch of rules. That is, we are not sure whether rules conflict or overlap with each other or not.

(2) As different applications might have various requirements and characteristics, the best rule sets for different applications are usually different. For example, grammatical errors made by Chinese are different from those by native speakers of English. However, there is no systematic approach to revise the patterns for various implementations.

(3) It is difficult to identify the effectiveness of each rule and to pinpoint the problematic rules systematically and effectively.

Therefore, an automatic rule selection method, i.e., ***Sequential Forward Selection (SFS)***, is proposed in this paper to tackle those problems. Given this SFS, we can (i) automatically find the suboptimal rule sets for different applications, (ii) objectively measure the effectiveness of each rule and (iii) systematically identify the problematic rules to let linguists revise them. Thus, the goal of a smaller set of patterns but better performance can be achieved. As a result, it takes 195 seconds (originally 269 seconds) to check 1956 sentences. If the suboptimal rule set (81) selected by SFS is adopted, not the whole rule set (127), the false rate decreases by 11.8% (from 26.4% to 14.8%) . However, the recognition rate decreases only by 3.9% (from 38.9% to 35%).

II. The Framework of BDC-Grammar Checker

A. The Construction of Error Patterns

Our error patterns of current version consist of 127 rules/patterns which are constructed by a linguist. They are based on common mistakes found from Chinese students' compositions and references (cf. C-L Su, 1991, Strunk et al., 1979, among others). These patterns have been

encoded in terms of Arabic numerals (about 600 code numbers). An example of such a pattern is represented in the second line of Table 1.

/*2.1.2.1 Fragment*/	(error code & error type)
% (#) (,) although (!-1 % (#) (,) * [v]|be * , * [v]|be	(error pattern/condition)
/* [Advice]: This sentence needs a main clause. */	(advice)
/* [Example]: Although the weather was bad. */	(example)
/* [Correction]: Although the weather was bad, he went hunting. */	(correction)

Table 1 An Example of ERROR PATTERN

A close look at the error pattern shows that our pattern also includes the part of speech (Lin et al., 1992; cf. Church, 1988). With the aid of the part of speech, many different words can be clustered into various equivalent classes to formulate more concise rules. The number of rules/patterns, thus, can be significantly reduced to save time and space. Therefore, when compared with other systems such as RightWriter's large (±6,500) rule base (Brace, 1992), our rule size seems small. However, in our pilot test, it performs even better than several other tools (please see Section III). Additionally, it is allowable for linguists to write patterns which include some special symbols such as {#, %, *, (), |, [ ]} ('#': one token;'%': syntactic boundary; '*': zero to many tokens; '()': optional; '|': or; '[ ]': categorical brackets).

B. The Construction of Finite State Automata & the Operating Flow

To put these error patterns to real use, they must be converted into Finite State Automata (FSA, cf. Hopcroft et al., 1979). First, patterns with special symbols are converted into regular expressions which are then converted into FSA (Karttunen, L. et al., 1992 among others). This FSA includes a finite set of both states and transitions from state to state at input symbols, as shown in Figure 1a.

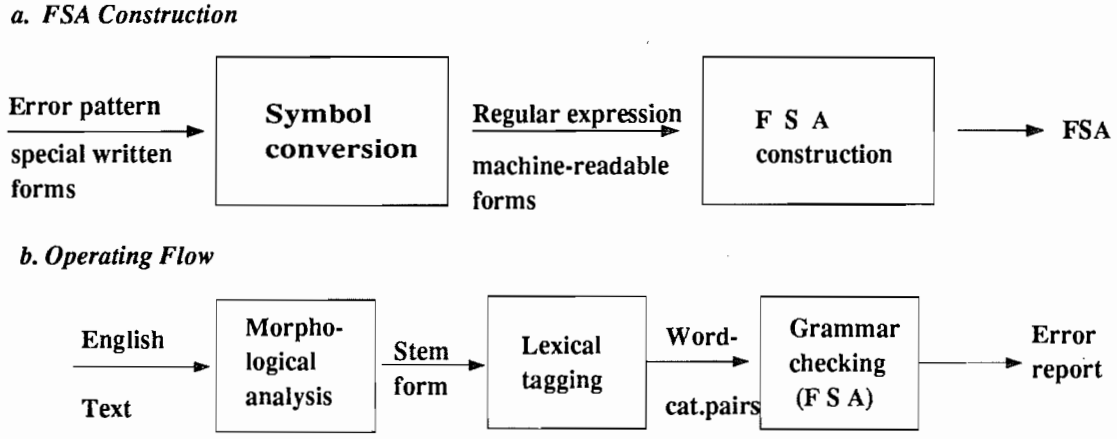

Figure 1 The flowchart of BDC-GC

A close look at the top flow in Figure 1a reveals that the special written forms such as {#, %, *, (), |, [ ]} are first converted to regular expressions. They are then transferred into machine-readable forms for the realization of FSA, which is the kernel of BDC-GC.

To illustrate the basic concept of our method, the operating flow of GC is shown in Figure 1b. Three stages of operation used to operate grammar-checking are stipulated as follows: (i) morphological analysis, (ii) lexical tagging and (iii) grammar checking. Let's take the sentence in Table 2 as an example. The surface form of a singular verb *influenced* is decomposed to the stem form '*influence*' with suffix ***ed*** through the morphological analysis. After the categorical tagging, it becomes a *word-category pair* with category ***v***. Afterwards, it turns out to be ***v/ed*** which is needed for grammar-checking.

(1)	Science has influenced our life	***Surface form***
(2)	science have influence our life	***Stem form***
(3)	n /- v /es v/ed poss/- n /-	***Category/Suffix***

Table 2 The morphological analysis & lexical tagging of BDC-GC

Currently, this system operates on Sun Sparc & IBM RS 6000. It takes about 269 seconds to check 1956 sentences (= 24069 words) on Sparc station ELC.

III. The Baseline System & Comparison with Other GC-Systems

To show the superiority of the proposed method, the original 127 rule set as a baseline system is used for comparison. Additionally, to give readers a general feeling about the performance of our baseline system, it is also compared with several other popular commercial products, e.g., Grammatik IV, RightWriter, The Writer's Toolkit, PowerEdit, etc. The comparison of performance evaluation for five different writer's tools is based on five pieces of student essays related to SCIENCE. There are 72 sentences (869 words) in total. Fifty-two errors are checked and hand-labeled by a linguist.

Table 3.1 illustrates the performance of different systems, where the recognition rate is calculated by the formula *recognition rate = (number of detected errors/number of total errors)%*; and the false rate is computed by *false rate = (number of false errors/number of total detected errors) %*. The best recognition rate is 65%, which is performed by BDC-GC. Likewise, the highest false rate (33%) also goes to our system. On the other hand, Writer's Toolkit performs at a high recognition rate (53%), but hits the lowest false rate (15%).

Unfortunately, PowerEdit, for example, does not perform as well here as it did in two previous tests listed in Table 3.2 & Table 3.3, where the recognition rate was approximately 51% (Rabinovitz, 1991) or 72% (Smith 1992).

% / Tools	BDC-GC	Writer's Toolkit	RightWriter	PowerEdit	Grammatik IV
Recognition	65%	53%	30%	38%	44%
False	33%	16%	15%	31%	24%

Table 3.1 Performance evaluation for five GC-tools (52 mistakes made by Chinese student)

The large performance variation among different tests might suggest that a large testing corpus is required for the sake of fair comparison. However, this kind of test is very time-consuming without modifying the error reporting program of other products. Besides, the great decrease of performance for PowerEdit (recognition rate: only 38%) in our test might suggest that it is not suitable for correcting essays written by Chinese students. This phenomenon also implies that different rule sets might be required for various applications.

% / Tools	RighrWriter	PowerEdit	Grammatik IV
Recognition	13%	51%	30%
False	8%	11%	3%

Table 3.2 Performance evaluation (grammar & style) based on Rabinovitz's report (150 test sentences) (Rabinovitz, 1991)

% / Tools	Writer's Toolkit	RightWriter	PowerEdit	Grammatik V
Recognition	56%	54%	72%	48%
False	2%	2%	2%	26%

Table 3.3 Performance evaluation based on Smith's report (50 errors) (Smith, 1992)

IV. How To Select Better Rules Based on Corpus

A. The Construction of Corpus Annotated with Error Patterns

To select rules automatically, an annotated corpus is required. Various archives of written tests and compositions from 2 universities and high schools were first collected. To make the students' essays easy to deal with, the raw material was first hand-labeled with the corresponding error codes, and then put together to form the corpus. For instance, the following case in Table 4 is an example of our training set. The double question mark "??" (e.g., in "**?? can not –> 7.3**") represents *error mark* (where an error is stipulated). After the question mark, the string "**can not**" indicates *error scope* and "**7.3**" illustrate *error code* (or error type).

Sentence	*"For example, people can travel around ... that can not be seen ... "*			
Hand-labled	??	can not	->	7.3
Meaning	error symbol	error scope	->	error code

Table 4 An example of hand-labeled correction

There are 30 grammatical categories employed in the system. (Examples are shown in Table 5.) Each code number represents a specific error type. The error scope as shown in the given example is indicated by a pair of brackets.

Code	Type	Example	Explanation
1.1	Dangling participle	*Seeing her teacher, [her face] turns red.*	Logical subjects in two clauses should be consistent.
2.1.1.1	Fragment	*[Although the weather is bad.]*	Missing main clause
18.1.2.1	Agreement	*[He get] up early every day.*	Subject-verb agreement

Table 5 Examples of 30 grammatical types to detect errors in GC-system

B. Automatic Rule Selection with Sequential Forward Selection (SFS)

To find the best rule set, it is necessary to check all the possible subsets of the original rule set. Different search algorithms (optimal & suboptimal) have been proposed (cf. Devijver et al., 1982) to do so. Among those, Sequential Forward Selection (SFS, also cf. Devijver) is adopted in our grammar checker. SFS is a simple bottom up search procedure which can be used to take care of the correlation among rules. Compared with other approaches, the SFS algorithm is faster and less complex. Thus, it is preferable in our system.

To implement the SFS algorithm, we first initialize two groups of rules (i.e., error patterns):

i). Group 1 (G1_rule) with all the rules (127); (original rule set)

ii). Group 2 (G2_rule) used to include the selected rules. (It is empty in the very beginning.)

The basic concept of SFS is to activate grammar-checking and to take the best rule of performance from G1_rule to G2_rule. Then, we choose the best one again from the remaining rules in G1_rule until it is empty. This algorithm is shown as follows.

The Sequential Forward Selection (SFS) Algorithm

```
SFS (n rules) {
     G1_rule = n rules; /*initialization*/
     G2_rule = empty;

     loop (while there are still rules in G1_rule){
```

```
        max_score = -9999; /*initialize the maximum score as -9999 here*/
        loop (for each rulei in G1_rule) do {
            build_fsa (rulei + G2_rule);  /*Combine rulei with those*/
                                /*already selected rules, then construct FSA*/
            check_grammar (input_text);
                                /*Operate grammar checker with new FSA.*/

            score = W1 * number of detected error - W2 * number of false alarm
                        /*Evaluate the performance;*/
                        /*where W1 and W2 are reference weight.*/
            if (score > max_score) then {
               max_score = score;     /*Replace the max_score */
                                      /*with the current one.*/
               best_rule = rulei;     /*rulei as the best rule*/
            }
        } /* end of for-loop */
        move the best_rule from G1_rule to G2_rule
    }    /* end of while-loop */
    output G2_rule;

}
```

The program includes two loops. The first one (while-loop) implies that while there are still rules/patterns in the G1_rule, it keeps working. The second loop (for-loop) indicates that for each remaining rule in the G1_rule, it builds up FSA for the rule set which is the union of the original G2_rule and the rule just picked up. Because all the rules in G2_rule are applied jointly, not disjunctively, the correlation among rules has been considered. Afterwards, it performs the grammar-checking and evaluation in terms of the following formula:

Score = [W1 * (number of detected errors)] — [W2 * (number of false alarms)], where W1 and W2 are the weighting factors which give different degrees of preference for *(number of detected errors)* and *(number of false alarms)*. Different weighting factors may be used for different applications. In summary, SFS uses the score function as the criteria for selecting the suboptimal rule set, which is a subset of the original rule set in many cases.

If any new score is bigger than the current max_score, then this score replaces the current one and becomes the max_score. Again, the best one is chosen and put into the G2_rule until the last one is done. That is, the G1_rule becomes empty in the long run.

V. Performance Evaluation

After executing SFS and operating 127 rules on the corpus of 1965 sentences, the results in Table 6 are generated. The rules are ranked according to their performance. For example, the most powerful rule, which is ranked Rule 1 here, finds 56 errors with 13 false alarms. Thus, it scores 43 for its performance by means of the formula (i.e., *Score = (number of errors detected) — (number of false alarms)*). The total detected mistakes throughout these 127 rules are 450 errors with 163 false alarms.

Rule	Detected errors	False alarms	Accumulative score	Rule	Detected errors	False alarms	Accumulative score
1	56	13	43	115	418	80	338
2	91	13	78	116	435	98	337
:	:	:	:	:	:	:	:
81	409	71	338				
82	416	78	338	126	448	129	319
:	:	:	:	127	450	163	287

Table 6 Statistical table with number of detected errors, false alarms & score

However, Table 6 shows that the rule set including the first 81 rules performs best. To capture a clear picture, Figure 2 is provided below.

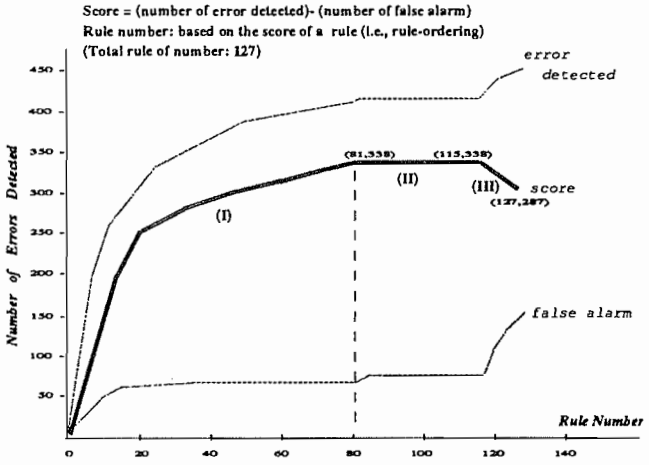

Figure 2 The Results of SFS with 127 Rules

The vertical axis shows the number of detected errors, 450 in total. The horizontal axis represents the ordered rule number (i.e., Rules 1–127) based on their performance as indicated by the given *formula*. The top curve shows the number of detected errors; the middle one illustrates the score of their performance, and the bottom one represents the number of false alarms. (The left Arabic numeral in each pair on the middle curve represents a rule number with its score at the right side of this pair, e.g., (81,338).)

The results demonstrate the following: .

(I). *The first 81 rules (scoring 338) perform better than the others.*

(II). *The performance of rules from 81 to 115 (also scoring 338) remain unchanged. This implies that these rules may be covered by the previous rules. Therefore, they can be deleted without any effect in our experiments.*

(III). *Afterwards, it goes down and finally degrades to 287. This suggests that Rules (116–127)*

should be revised or thrown away because they cause more false alarms than detected errors.

As mentioned above, our tests are based on a corpus of 1956 sentences. After the application of SFS and the subsequent rule-revision, the false rate decreases by 11.8% (from 26.4% to 14.8%) if the suboptimal rule set (81) selected by SFS is adopted, not the whole rule set (127). However, the recognition rate decreases only by 3.9% (from 38.9% to 35%), without ruining the merit of its better performance. That is, the best 81 rules (scoring *338*) perform better than that of the total 127 rules (scoring *282*).

VI. Conclusions

This paper stipulates that the pattern matching approach is still widely used in the area of grammar checking. The reason is that when compared with the method of employing full scale parsing, pattern matching is efficient in detecting local errors with much less computer time and memory. However, the error patterns used in the pattern matching algorithm are usually hand-tuned, and thus suffer problems such as the problem of correlation among patterns. These patterns may conflict or overlap with other patterns. The purpose of this paper, therefore, provides an automatic rule selection method, i.e., ***Sequential Forward Selection (SFS)***, to handle these problems. This algorithm uses objective performance measures and then automatically searches the suboptimal rule-set among all possible combinations. With the help of SFS, the effectiveness of each rule can be measured and the problematic patterns can be identified systematically by linguists in order to fine-tune them effectively. Moreover, the error patterns can be revised efficiently. The above tests based on a corpus of 1956 sentences display that the false rate decreases by 11.8% (from 26.4% to 14.8%) if the suboptimal rule set (81) selected by SFS is adopted, instead of the whole rule set (127). However, the recognition rate decreases only by 3.9% (from 38.9% to 35%) by using this proposed algorithm. The implementation of SFS in BDC-GC, therefore, is strongly recommended to improve the performance of grammar-checking.

ACKNOWLEDGEMENTS

This project is supported by *The Science-based Industrial Park* (Project Numbers: 81–1155–111–189 & 82–1230–111–267). We would like to thank Dr. Hsian-chin Liu (NTHU), Dr. Wen-yu Chiang (NTU) and English instructors Shu-ying Chang, Chiu-chu Li, Yung-hsian Ting, among others from Hsinchu Girl's Senior High School, who all have provided their students' essays to construct the error pattems used in the prototype of our grammar checker. We would also like to express sincere thanks to our consultants, Dr. Ting-chi Tang & Dr. Samuel H. Wang (NTHU), for their kind assistance. Many thanks also to our colleagues, Jing-shin Chang, Ting-yu Cheng, Pen-hwa Lin, Cheng-li Tien, Renee Ting, Jong-nai Wang, and Ming-wen Wu for their comments and help. Last but not least, we appreciate two anonymous reviewers' valuable suggestions.

References

Atwell, E.S. 1987. "How to Detect Grammatical Errors in a Text without Parsing it," Proc. of Third Conference of the European Chapter of the ACL: pp. 38–45, Univ. of Copenhagen, Copenhagen, Denmark.

Brace, C. 1992. "RightWriter: State of the Art?" Language Industry Monitor, No. 12 Nov.-Dec.: p.9.

Chanod, J.P., M. El-Beze & S. Guillemin-Lanne. 1992. "Coupling an Automatic Dictation System with a Grammar, " Proc. of Coling-92: pp. 940–944, Nantes, France.

Church, K. 1988. "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text," ACL Proceedings of 2nd Conference on Applied Natural Processing: pp.136–143, Austin, Texas, USA.

Devijver, P.A. & J. Kittler. 1982. *Pattern Recognition: A Statistical Approach*, Prentice-Hall Int., Inc., London.

Hopcroft, J. & J. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing Company, Philippines.

Kay, M. 1987. "Nonconcatenative Finite-State Morphology," Proc. of ACL: pp. 2–10, Stanford Univ., Stanford, CA, USA.

Karttunen, L., R.M. Kaplan & A. Zaenen. 1992. "Two-Level Morphology with Composition," Proc. of Coling-92: pp. 141–148, Nantes, France.

Lin, Y.-C., T.-H. Chiang & K.-Y. Su. 1992. "Discrimination Oriented Probabilistic Tagging," ROCLING 5: pp. 87–96, Taipei.

Pfaltz, J., W. Berman & E. Cagley. 1980. "Partial-Match Retrieval Using Indexed Descriptor Files," Communications of ACM 23 (9): pp. 522–528.

Rabinovitz, R. 1991. "Write on Target: 15 Writer's Tools," PC Magzine (Sept. 24, 1991): pp. 321–369.

Smith, J. 1992. "Mark Your Words with Grammar-Checking Software," PC/Computing (Oct. 1992): pp. 243–252.

Strunk, W. & E.B. White. 1979. *The Elements of Style*, Macmillam Publishering Co., Inc., New York.

Su, C-L (eds.) 1991. *Common Mistakes in English Composition and Translation*, Bookman Co., Inc., Taipei.

Su, K-Y & J-S Chang. 1992 "Why Corpus-based Statistics-oriented Machine Translation," 4th Int. Conf. on Theoretical and Methodological Issues in Machine Translation, Proc. of TMI-92: pp. 249–262, Montreal, Canada.

Vergne, J. & I. Paris. 1986. "Synergy of Syntax and Morphology in Automatic Parsing of French Language with a Minimum of Data," Proc. of Coling '86: pp. 269–271, Univ. of Bonn, Bonn, Germany.

Vosse, T. 1992. "Detecting and Correcting Morpho-syntactic Errors in Real Texts," Proc. of Third Conference on Applied Natural Language Processing: pp. 111–118, Trento, Italy.