A Robust Parser Based on Syntactic Information

Kong Joo Lee Cheol Jung Kweon Jungyun Seo Gil Chang Kim
Department of Computer Scinence and CAIR
Korea Advanced Institute of Science and Technology
Taejon, Korea 305-701
{kjlee,cjkwn}@csone.kaist.ac.kr

Abstract

An extragrammatical sentence is what a normal parser fails to analyze. It is important to recover it using only syntactic information although results of recovery are better if semantic factors are considered. *A general algorithm for least-errors recognition*, which is based only on syntactic information, was proposed by G. Lyon to deal with the extragrammaticality. We extended this algorithm to recover extragrammatical sentence into grammatical one in running text. Our robust parser with recovery mechanism – extended general algorithm for least-errors recognition – can be easily scaled up and modified because it utilize only syntactic information. To upgrade this robust parser we proposed heuristics through the analysis on the Penn treebank corpus. The experimental result shows 68% ~ 77% accuracy in error recovery.

1 Introduction

Extragrammatical sentences include patently ungrammatical constructions as well as utterances that may be grammatically acceptable but are beyond the syntactic coverage of a parser, and any other difficult ones that are encountered in parsing (Carbonell and Hayes, 1983).

I am sure this is what he means.
This is, I am sure, what he means.

The progress of machine does not stop even a day.
Not even a day does the progress of machine stop.

Above examples show that people are used to write same meaningful sentences differently. In addition, people are prone to mistakes in writing sentences. So, the bulk of written sentences are open to the extragrammaticality.

In the Penn treebank tree-tagged corpus(Marcus, 1991), for instance, about 80 percents of the rules are concerned with peculiar sentences which include inversive, elliptic, parenthetic, or emphatic phrases. For example, we can drive a rule *VP → vb NP comma rb comma PP* from the following sentence.

The same jealousy can breed confusion, *however*, in the absence of any authorization bill this year.

```
(
(S
 (NP The/dt
  (ADJP same/jj) jealousy/nn) can/md
 (VP breed/vb
  (NP confusion/nn) ,/, however/rb ,/,
  (PP in/in
   (NP
    (NP the/dt absence/nn)
    (PP of/in
     (NP any/dt authorization/nn bill/nn))
    (NP this/dt year/nn)))))
./.)
```

A robust parser is one that can analyze these extragrammatical sentences without failure. However, if we try to preserve robustness by adding such rules whenever we encounter an extragrammatical sentence, the rulebase will grow up rapidly, and thus processing and maintaining the excessive number of rules will become inefficient and impractical. Therefore, extragrammatical sentences should be handled by some recovery mechanism(s) rather than by a set of additional rules.

Many researchers have attempted several techniques to deal with extragrammatical sentences such as Augmented Transition Network(ATN) (Kwasny and Sondheimer, 1981), network-based semantic grammar (Hendrix, 1977), partial pattern matching (Hayes and Mouradian, 1981), conceptual case frame (Schank et al., 1980), and multiple cooperating methods (Hayes and Carbonell, 1981). Above mentioned techniques take into account various semantic factors depending on specific domains on question in recovering extragrammatical sentences. Whereas they can provide even better solutions intrinsically, they are usually adhoc and are lack of extensibility. Therefore, it is important to recover extragrammatical sentences using syntactic factors only, which are independent of any particular system and any particular domain.

Mellish (Mellish, 1989) introduced some chart-based techniques using only syntactic information for extragrammatical sentences. This technique has an advantage that there is no repeating work for the chart to prevent the parser from generating the same edge as the previously existed edge. Also, because the recovery process runs when a normal parser terminates unsuccessfully, the performance of the normal parser does not decrease in case of handling grammatical sentences. However, his experiment was not based on the errors in running texts but on artificial ones which were randomly generated by human. Moreover, only one word error was considered though several word errors can occur simultaneously in the running text.

A general algorithm for least-errors recognition (Lyon, 1974), proposed by G. Lyon, is to find out the least number of errors necessary to successful parsing and recover them. Because this algorithm is also syntactically oriented and based on a chart, it has the same advantage as that of Mellish's parser. When the original parsing algorithm terminates unsuccessfully, the algorithm begins to assume errors of insertion, deletion and mutation of a word. For any input, including grammatical and extragrammatical sentences, this algorithm can generate the resultant parse tree. At the cost of the complete robustness, however, this algorithm degrades the efficiency of parsing, and generates many intermediate edges.

In this paper, we present a robust parser with a recovery mechanism. We extend ***the general algorithm for least-errors recognition*** to adopt it as the recovery mechanism in our robust parser. Because our robust parser handle extragrammatical sentences with this syntactic information oriented recovery mechanism, it can be independent of a particular system or particular domain. Also, we present the heuristics to reduce the number of edges so that we can upgrade the performance of our parser.

This paper is organized as follows : We first review a general algorithm for least-errors recognition. Then we present the extension of this algorithm, and the heuristics adopted by the robust parser. Next, we describe the implementation of the system and the result of the experiment of parsing real sentences. Finally, we make conclusion with future direction.

2 Algorithm and Heuristics

2.1 General algorithm for least-errors recognition

The general algorithm for least-errors recognition (Lyon, 1974), which is based on Earley's algorithm, assumes that sentences may have insertion, deletion, and mutation errors of terminal symbols. The objective of this algorithm is to parse input string with the least number of errors.

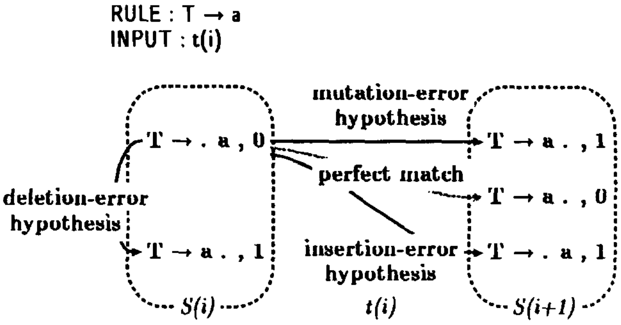


Figure 1: SCAN processing

A ***state*** used in this algorithm is quadruple *(p, j, f, e)*, where *p* is a production number in grammar, *j* marks a position in *RHS(p)*, *f* is a start position of the state in input string, and *e* is an error value.[1] A ***final state*** *(p, $\underline{p}$+1, f, e)* denotes recognition of a phrase *RHS(p)* with *e* errors where $\underline{p}$ is a number of components in rule *p*. A ***stateset*** *S(i)*, where *i* is the position of the input, is an ordered set of states. States within a stateset are ordered by ascending value of *j*, within a *p* within a *f* ; *f* takes descending value.

When adding to statesets, if state *(p, j, f, e)* is a candidate for admission to a stateset which already has a similar member *(p, j, f, e')* and *e'* ≤ *e*, then *(p, j, f, e)* is rejected. However, if *e'* > *e*, then *(p, j, f, e')* is replaced by *(p, j, f, e)*.

The algorithm works as follows : A procedure SCAN is carried out for each state in *S(i)*. SCAN checks various correspondences of input token *t(i)* against terminal symbols in *RHS* of rules. Once SCAN is done, COMPLETER substitutes all final states of *S(i)* into all other analyses which can use them as components.

SCAN

SCAN handles states of *S(i)*, checking each input terminal against requirements of states in *S(i)* and various error hypotheses. Figure 1 shows how SCAN processes.

Let *c(p,j)* be ***j-th*** component of *RHS(p)* and *t(i)* be ***i-th*** word of input string.

- ***perfect match*** :
  If *c(p,j)* = *t(i)* then add *(p, j+1, f, e)* to *S(i+1)* if possible.
- ***insertion-error hypothesis*** :
  Add *(p, j, f,* $e+\alpha_{insertion}$*)* to *S(i+1)* if possible.
  $\alpha_{insertion}$ is the cost of an insertion-error for a terminal symbol.
- ***deletion-error hypothesis*** :

[1] Lyon said that e is an error count

If *c(p,j)* is terminal, then add *(p, j+1, f, e+$\alpha_{deletion}$)* to *S(i)* if possible.
$\alpha_{deletion}$ is the cost of a deletion-error for a terminal symbol.

- ***mutation-error hypothesis*** :
  If *c(p,j)* is terminal but not equal to *t(i)*, then add *(p, j+1, f, e+$\alpha_{mutation}$)* to *S(i+1)* if possible.
  $\alpha_{mutation}$ is the cost of a mutation-error for a terminal symbol.[2]

COMPLETER
COMPLETER handles substitution of final states in *S(i)* like that of original Earley's algorithm. Each final state means the recognition of a nonterminal.

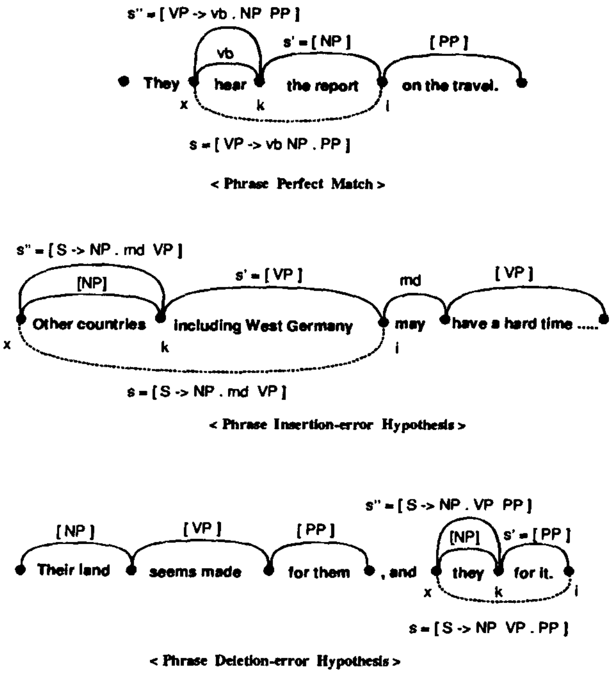
2.2 Extension of least-errors recognition algorithm

The algorithm in section 2.1 can analyze any input string with the least number of errors. But this algorithm can handle only the errors of terminal symbols because it doesn't consider the errors of nonterminal nodes. In the real text, however, the insertion, deletion, or inversion of a phrase – namely, nonterminal node – occurs more frequently. So, we extend the original algorithm in order to handle the errors of nonterminal symbols as well.

In our extended algorithm, the same SCAN as that of the original algorithm is used, while COMPLETER is modified and extended. Figure 2 shows the processing of extended-COMPLETER. In figure 2, *[NP]* denotes the final state whose rule has *NP* as its *LHS*. In other words, it means the recognition of a noun phrase.

extended-COMPLETER
If there is a final state $s' = (p', p' + 1, k, e')$ in $S(i)$,

- ***phrase perfect match***
  If there exists a state $s'' = (p, j, x, e)$ in $S(k)$, $k < i$ and $c(p, j) = LHS(p')$ then add $s = (p, j + 1, x, e + e')$ into $S(i)$.
- ***phrase insertion-error hypothesis*** [3]
  If there exists a state $s'' = (p, j, x, e)$ in $S(k)$ then add $s = (p, j, x, e + \beta_{insertion})$ into $S(i)$ if possible.
  $\beta_{insertion}$ is the cost of a insertion-error for a nonterminal symbol.



< Phrase Perfect Match >

< Phrase Insertion-error Hypothesis >

< Phrase Deletion-error Hypothesis >

Figure 2: Examples of extended-COMPLETER processing

- ***phrase deletion-error hypothesis***
  If there exists a state $s'' = (p, j, x, e)$ in $S(k)$ and $c(p, j)$ is a nonterminal then add $s = (p, j + 1, x, e + \beta_{deletion})$ into $S(k)$ if possible.
  $\beta_{deletion}$ is the cost of a deletion-error for a nonterminal symbol.
- ***phrase mutation-error hypothesis*** [4]
  If there exists a state $s'' = (p, j, x, e)$ in $S(k)$ and $c(p, j)$ is a nonterminal but not equal to $L(p')$ then add $s = (p, j + 1, x, e + \beta_{mutation})$ into $S(i)$ if possible.
  $\beta_{mutation}$ is the cost of a mutation-error for a nonterminal symbol.

The extended least-errors recognition algorithm can handle not only terminal errors but also nonterminal errors.

2.3 Heuristics

The robust parser using the extended least-errors recognition algorithm overgenerates many error-hypothesis edges during parsing process. To cope with this problem, we adjust error values according to the following heuristics. Edges with more error values are regarded as less important ones, so that those edges are processed later than those of less error values.

[2] $\alpha_{insertion}$, $\alpha_{deletion}$, $\alpha_{mutation}$ are all strictly 1 in Lyon's original paper.

[3] In fact, there are cases that an inserted phrase cannot be constructed to form a nonterminal node. In phrase insertion-error hypothesis of figure 2, the original sentence is "Other countries, including West Germany, may have ...", where the inserted phrase VP is surrounded by commas. So, the substring(*comma VP comma*) should be dealt with as a constituent in extended-COMPLETER. In fact, we implemented the algorithm to allow substring insertions as well as insertions of nonterminal nodes.

[4] We know that the phrase mutation-error hypothesis is not meaningful in the real text because we cannot find out any example of phrase mutation-error in the corpus. So we didn't implement the phrase mutation-error hypothesis.

- **Heuristics 1: error types**
  The analysis on 3,538 sentences of the Penn treebank corpus WSJ shows that there are 498 sentences with phrase deletions and 224 sentences with phrase insertions. So, we assign less error value to the deletion-error hypothesis edge than to the insertion- and mutation-errors.

  $\alpha < \beta$

  $\alpha_{deletion} < \alpha_{insertion} < \alpha_{mutation}$
  $\beta_{deletion} < \beta_{insertion}$

  where α is the error cost of a terminal symbol, β is the error cost of a nonterminal symbol.

- **Heuristics 2: fiducial nonterminal**
  People often make mistakes in writing English. These mistakes usually take place rather between small constituents such as a verbal phrase, an adverbial phrase and noun phrase than within small constituents themselves. The possibility of error occurrence within noun phrases are lower than between a noun phrase and a verbal phrase, a preposition phrase, an adverbial phrase. So, we assume some phrases, for example noun phrases, as fiducial nonterminals, which means error-free nonterminals. When handling sentences, the robust parser assings more error values(δ_1) to the error hypothesis edge occurring within a fiducial nonterminal.

- **Heuristics 3: kinds of terminal symbols**
  Some terminal symbols like punctuation symbols, conjunctions and particles are often misused. So, the robust parser assigns less error values($-\delta_2$) to the error hypothesis edges with these symbols than to the other terminal symbols.

- **Heuristics 4: inserted phrases between commas or parentheses**
  Most of inserted phrases are surrounded by commas or parentheses. For example,

  a. They're active , *generally* , at night or on damp, cloudy days.
  b. All refrigerators , *whether they are defrosted manually or not* , need to be cleaned.
  c. I was a last-minute *(read interloping)* attendee at a French journalism convention ···

  We will assign less error values($-\delta_3$) to the insertion-error hypothesis edges of nonterminals which are embraced by comma or parenthesis.

δ_1 and δ_2 are weights for the error of terminal nodes, and δ_3 is a weight for the error of nonterminal nodes.

The error value e of an edge is calculated as follows. All error values are additive.
The error value e for a rule $X \rightarrow a_1 A_1 a_2 \cdots a_i A_j$, where a is a terminal node and A is a nonterminal node, is

1. $e = \sum_1^i e_T + \sum_1^j e_{NT}$

2. $e_T = \begin{cases} \alpha + \delta_1 - \delta_2 & \text{if terminal error} \\ 0 & \text{otherwise} \end{cases}$

3. $e_{NT} = \begin{cases} \beta - \delta_3 + e_{child} & \text{if nonterminal error} \\ e_{child} & \text{otherwise} \end{cases}$

where $\alpha \in \{\alpha_{insertion}, \alpha_{deletion}, \alpha_{mutation}\}$, $\beta \in \{\beta_{insertion}, \beta_{deletion}\}$ and e_{child} is an error value of a child edge.

By these heuristics, our robust parser can process only plausible edges first, instead of processing all generated edges at the same time, so that we can enhance the performance of the robust parser and result in the great reduction in the number of resultant trees.

3 Implementation and Evaluation

3.1 The robust parser

Our robust parsing system is composed of two modules. One module is a normal parser which is the bottom-up chart parser. The other is a robust parser with the error recovery mechanism proposed herein. At first, an input sentence is processed by the normal parser. If the sentence is within the grammatical coverage of the system, the normal parser succeed to analyze it. Otherwise, the normal parser fails, and then the robust parser starts to execute with edges generated by the normal parser. The result of the robust parser is the parse trees which are within the grammatical coverage of the system. The overview of the system is shown in figure 3.

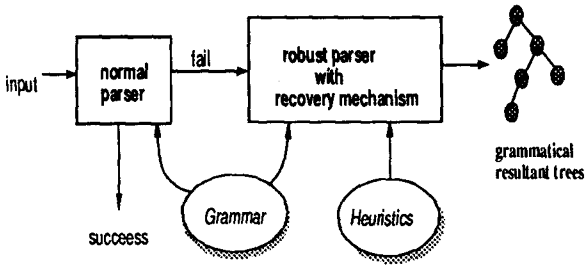

input
normal parser
fail
robust parser with recovery mechanism
grammatical resultant trees
successs
Grammar
Heuristics

Figure 3: The overview of the system

3.2 Experimental result

To show usefulness of the robust parser proposed in this paper, we made some experiments.

- *Rule*
  We can derive 4,958 rules and their frequencies out of 14,137 sentences in the Penn

Table 1: The results of the robust parser on WSJ

Experiment 1 : WSJ 410 sentences		
	with Heuristics	without Heuristics
Average sentence length	16.27 words (2-25 words)	16.27 words (2-25 words)
Average processing time	6.52 sec	22.47 sec
Average number of edges	7726.03	10346.6
Accuracy (%)	77.1	72.8
no-crossing sentences	23.28%	20.28%
% of $\leq$ 1-crossing sentences	40.52%	37.14%
% of $\leq$ 2-crossing sentences	55.17%	48.57%

treebank tree-tagged corpus, the Wall Street Journal. The average frequency of each rule is 48 times in the corpus. Of these rules, we remove rules which occurs fewer times than the average frequency in the corpus, and then only 192 rules are left. These removed rules are almost for peculiar sentences and the left rules are very general rules. We can show that our robust parser can compensate for lack of rules using only 192 rules with the recovery mechanism and heuristics.

- *Test set*
  First, 1,000 sentences are selected randomly from the WSJ corpus, which we have referred to in proposing the robust parser. Of these sentences, 410 are failed in normal parsing, and are processed again by the robust parser. To show the validity of these heuristics, we compare the result of the robust parser using heuristics with one not using heuristics. Second, to show the adaptability of our robust parser, same experiments are carried out on 1,000 sentences from the ATIS corpus in Penn treebank, which we haven't referred to when we propose the robust parser. Among 1,000 sentences from the ATIS, 465 sentences are processed by the robust parser after the failure of the normal parsing.
- *Parameter adjustment*
  We chose the best parameters of heuristics by executing several experiments.

$\alpha_{insertion}$	: 10.2	$\beta_{insertion}$	: 15.0
$\alpha_{deletion}$	: 10.4	$\beta_{deletion}$	: 20.0
$\alpha_{mutation}$	: 10.8		
δ_1	: 0.01	δ_2	: 5.0
δ_3	: 1.0		

Accuracy is measured as the percentage of constituents in the test sentences which do not cross any Penn treebank constituents (Black, 1991). Table 1 shows the results of the robust parser on WSJ. In table 1, *5th, 6th* and *7th* raw mean that the percentage of sentences which have no crossing constituents, less than one crossing and less than two crossing respectively. With heuristics, our robust parser can enhance the processing time and reduce the number of edges. Also, the accuracy is improved from 72.8% to 77.1% even if the heuristics differentiate edges and prefer some edges. It shows that the proposed heuristics is valid in parsing the real sentences. The experiment says that our robust parser with heuristics can recover perfectly about 23 sentences out of 100 sentences which are just failed in normal parsing, as the percentage of no-crossing sentences is about 23.28%.

Table 2 is the results of the robust parser on ATIS which we did not refer to before. The accuracy of the result on ATIS is lower than WSJ because the parameters of the heuristics are adjusted not by ATIS itself but by WSJ. However, the percentage of sentences with constituents crossing less than 2 is higher than the WSJ, as sentences of ATIS are more or less simple.

The experimental results of our robust parser show high accuracy in recovery even though 96% of total rules are removed. It is impossible to construct complete grammar rules in the real parsing system to succeed in analyzing every real sentence. So, parsing systems are likely to have extragrammatical sentences which cannot be analyzed by the systems. Our robust parser can recover these extragrammatical sentences with 68 $\sim$ 77% accuracy.

It is very interesting that parameters of heuristics reflect the characteristics of the test corpus. For example, if people tend to write sentences with inserted phrases, then the parameter $\beta_{insertion}$ must increase. Therefore we can get better results if the parameter are fitted to the characteristics of the corpus.

4 Conclusion

In this paper, we have presented the robust parser with the extended least-errors recognition algorithm as the recovery mechanism. This robust parser can easily be scaled up and applied to various domains because this parser depends only on syntactic factors. To enhance the performance of the robust parser for extragrammatical sentences,

Table 2: The results of the robust parser on ATIS

Experiment 2 : ATIS 465 sentences		
	with Heuristics	without Heuristics
Average sentence length	10.55 words (2-25 words)	10.55 words (2-25 words)
Average processing time	8.68 sec	71.98 sec
Average number of edges	12974.2	25652.5
Accuracy (%)	68.5	59.4
no-crossing sentences	26.02%	13.28%
% of ≤ 1-crossing sentences	47.10%	36.06%
% of ≤ 2-crossing sentences	66.24%	52.46%

we proposed several heuristics. The heuristics assign the error values to each error-hypothesis edge, and edges which has less error values are processed first. So, not all the generated edges are processed by the robust parser, but the most plausible parse trees can be generated first. The accuracy of the recovery in our robust parser is about 68% ~ 77%. Hence, this parser is suitable for systems in real application areas.

Our short term goal is to propose an automatic method that can learn parameter values of heuristics by analyzing the corpus. We expect that automatically learned values of parameters can upgrade the performance of the parser.

Acknowledgement

This work was supported(in part) by Korea Science and Engineering Foundation(KOSEF) through Center for Artificial Intelligence Research(CAIR), the Engineering Research Center(ERC) of Excellence Program.

References

[Black, 1991] E. Black et al. A Procedure for quantitatively comparing the syntactic coverage of English grammars. *Proceedings of Fourth DARPA Speech and Natural Language Workshop*, pp. 306–311, 1991.

[Carbonell and Hayes, 1983] J. G. Carbonell and P. J. Hayes. Recovery Strategies for Parsing Extragrammatical Language. *American Journal of Computational Linguistics*, vol. 9, no. 3–4, pp. 123–146, 1983.

[Hayes and Carbonell, 1981] P. Hayes and J. Carbonell. Multi-strategy Construction-Specific Parsing for Flexible Data Base Query Update. *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pp. 432–439, 1981.

[Hayes and Mouradian, 1981] P. J. Hayes and G. V. Mouradian. Flexible Parsing. *American Journal of Computational Linguistics*, vol. 7, no. 4, pp. 232–242, 1981.

[Hendrix, 1977] G. Hendrix. Human Engineering for Applied Natural Language Processing. *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pp. 183–191, 1977.

[Kwasny and Sondheimer, 1981] S. Kwasny and N. Sondheimer. Relaxation Techniques for Parsing Grammatically Ill-Formed Input in Natural Language Understanding Systems. *American Journal of Computational Linguistics*, vol. 7, no. 2, pp. 99–108, 1981.

[Lyon, 1974] G. Lyon. Syntax-Directed Least-Errors Analysis for Context-Free Languages. *Communications of the ACM*, vol. 17, no. 1, pp. 3–14, 1974.

[Marcus, 1991] M. P. Marcus. Building very Large natural language corpora : the Penn Treebank, 1991.

[Mellish, 1989] C. S. Mellish. Some Chart-Based Techniques for Parsing Ill-Formed Input. *Association for Computational Linguistics*, pp. 102–109, 1989.

[Schank et al., 1980] R. C. Schank, M. Lebowitz and L. Brinbaum. An Intergrated Understander. *American Journal of Computational Linguistics*, vol. 6, no. 1, pp. 13–30, 1980.