IEICE TRANS. INF. & SYST., VOL.E100–D, NO.5 MAY 2017

PAPER

Correcting Syntactic Annotation Errors Based on Tree Mining*

Kanta SUZUKI†a), *Nonmember*, Yoshihide KATO††, *Member*, *and* Shigeki MATSUBARA†, *Senior Member*

SUMMARY This paper provides a new method to correct annotation errors in a treebank. The previous error correction method constructs a pseudo parallel corpus where incorrect partial parse trees are paired with correct ones, and extracts error correction rules from the parallel corpus. By applying these rules to a treebank, the method corrects errors. However, this method does not achieve wide coverage of error correction. To achieve wide coverage, our method adopts a different approach. In our method, we consider that if an infrequent pattern can be transformed to a frequent one, then it is an annotation error pattern. Based on a tree mining technique, our method seeks such infrequent tree patterns, and constructs error correction rules each of which consists of an infrequent pattern and a corresponding frequent pattern. We conducted an experiment using the Penn Treebank. We obtained 1,987 rules which are not constructed by the previous method, and the rules achieved good precision.

key words: *error correction, synchronous tree substitution grammar, FREQT*

1. Introduction

It is inevitable for annotated corpora to contain errors caused by manual or semi-manual annotation process. Thus, detecting and correcting errors in annotated corpora are important tasks. Many studies suggested methods of detecting or correcting errors in POS-tag or dependency annotation [2]–[7]. On the other hand, there is little work on error correction in phrase structure treebank while there are several methods of detecting annotation errors in a treebank [8]–[15].

One exception is the work of Kato and Matsubara [16]. Their method constructs a *pseudo parallel corpus* where incorrect parse trees are paired with correct ones, and extracts error correction rules from the parallel corpus. The rules transform incorrect tree patterns to correct ones. By applying these rules to a treebank, the method corrects errors. However, this method does not achieve wide coverage of error correction.

To solve this problem, we propose another approach to construct error correction rules. Our method does not construct a pseudo parallel corpus. In our method, we consider that an infrequent tree pattern which can be transformed to a frequent one is an annotation error pattern. Based on a tree mining technique, our method seeks such infrequent patterns efficiently. The method constructs error correction rules by pairing the infrequent tree patterns with the frequent ones. We conducted an experiment using the Penn Treebank [17]. We obtained 1,987 rules which are not constructed by the previous method, and the rules achieved good precision.

This paper is organized as follows: Section 2 introduces the previous method of correcting errors in a treebank. Section 3 explains our method which is based on tree mining. Section 4 reports experimental results using the Penn Treebank.

2. Previous Work

Kato and Matsubara [16] proposed a method of correcting annotation errors in a treebank. Their method is based on *synchronous tree substitution grammar* (STSG) [18]. An STSG defines a tree-to-tree mapping, and consists of rules each of which is defined as a pair of trees called *elementary trees*. The one tree is called a *source*, and the other is called a *target*. Figure 1 shows an example of STSG rule. The rule transforms the structure which matches the source into the target's structure. To correct annotation errors in a treebank, the method constructs STSG rules which transform incorrect structures to correct ones and applies them to the treebank.

The STSG rules are constructed as follows:

1. Make a *pseudo parallel corpus*, which is a collection of pairs of partial parse trees which cover a same word sequence.
2. Extract STSG rules which represent a correspondence in the pseudo parallel corpus.

To select useful rules for error correction, they define a score function. Let $\langle \tau_s, \tau_t \rangle$ be a rule whose source is τ_s and whose target is τ_t. The score of $\langle \tau_s, \tau_t \rangle$ is defined as follows:

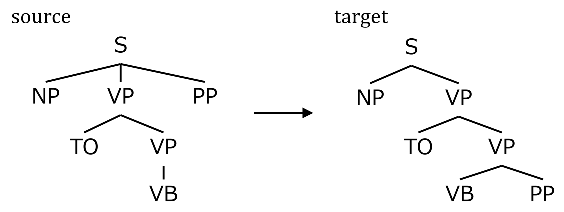


Fig. 1 Example of STSG rule

Manuscript received August 24, 2016.
Manuscript revised December 9, 2016.
Manuscript publicized January 23, 2017.

†The authors are with the Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8601 Japan.
††The author is with the Information & Communications, Nagoya University, Nagoya-shi, 464–8601 Japan.

*The preliminary version of this paper was presented in the 10th International Conference on Language Resources and Evaluation (LREC 2016) by the same authors [1].

a) E-mail: ksuzuki@db.ss.is.nagoya-u.ac.jp

DOI: 10.1587/transinf.2016EDP7357

$$Score(\langle \tau_s, \tau_t \rangle) = \frac{f(\tau_t)}{f(\tau_s) + f(\tau_t)}$$

where $f(\tau)$ is the frequency of an elementary tree τ in a treebank. They assume that the frequency of an incorrect parse tree in a treebank is very low. The lower $f(\tau_s)$ is, the higher $Score(\langle \tau_s, \tau_t \rangle)$ is. STSG rules with high scores are useful for error correction.

For example, let us consider a treebank which includes the parse trees shown in Fig. 2. The parse tree (a) is correct, but (b) and (c) include a same annotation error. In (a) and (b), the word sequence "to sell at the same time" has different partial parse trees enclosed within the dotted line. The method makes a pair of these partial parse trees and extracts the STSG rule shown in Fig. 1 from the pair. Applying this rule to the treebank, we can correct the error in (b). Moreover, the error in (c) can be corrected by this rule.

However, this method has a problem. It can not extract any rule from a partial parse tree assigned to a word sequence which occurs only once in a treebank. Thus, annotation errors included in only such partial parse tree can not be corrected by the method. Let us consider another case where the treebank does not include (b). In (c), the word sequence "to trade on Nasdaq" has incorrect partial parse tree. But, the method can not make a pair of partial parse trees enclosed within the dotted lines in (a) and (c). This is because these partial parse trees have different word sequences. This means that it constructs no rule. As the result, the method fails to correct the annotation error in (c).

3. Correcting Errors by Tree Mining

To solve the problem described in Sect. 2, we adopt a different approach. Our method does not construct a pseudo parallel corpus. STSG rules are constructed based on a tree mining technique.

3.1 Definition

In this section, we give some definitions.

3.1.1 Derivation Tree

In our method, a parse tree is represented by a *derivation tree*. Figure 3 shows the derivation tree corresponding to the partial parse tree enclosed within the dotted line in Fig. 2 (a). A derivation tree for a parse tree is defined as follows: for each inner node v of a parse tree, there exists a node v' which corresponds to v. v' preserves the parent-child relations on v. The label of v' is the following grammar rule:

$$l(v) \rightarrow l(c_1)\ l(c_2)\ \ldots\ l(c_n)$$

where $l(v)$ is the label of v and $c_1, c_2, \ldots, c_n$ are the children of v. We label the edge between v' and c'_i with i in order to indicate that a grammar rule $l(c'_i)$ is applied to the i-th element of the right-hand side of $l(v')$.

3.1.2 Pattern

We define a *pattern* as a connected subgraph included in a tree. Figure 4 shows examples of patterns. τ_1, τ_2 and τ_3 are included in the derivation tree shown in Fig. 3. A pattern with k nodes is called a k-pattern.

In a derivation tree pattern, if no grammar rule is applied to an element in the right-hand side of a grammar rule

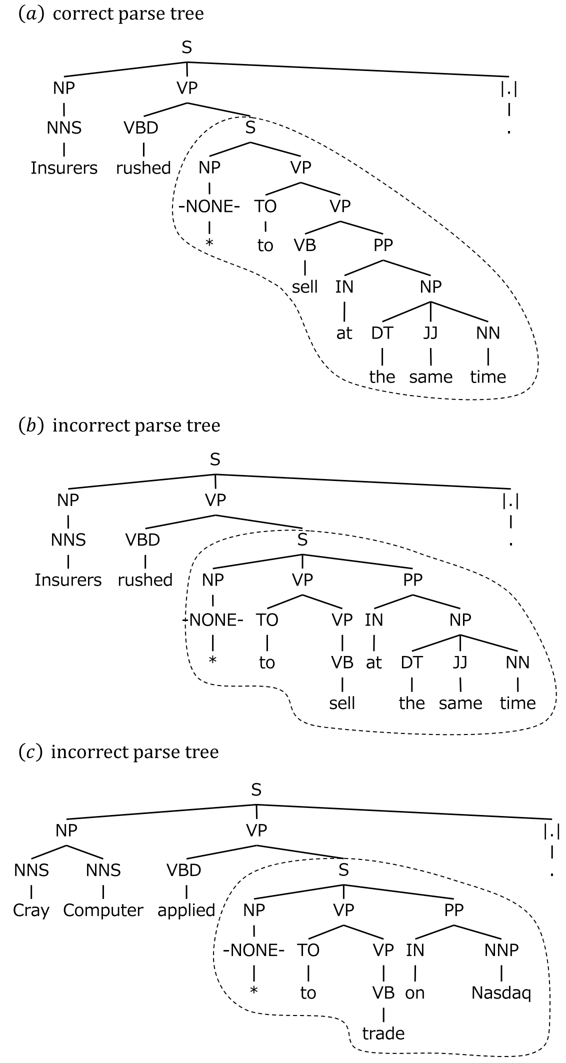


Fig. 2 Examples of parse trees

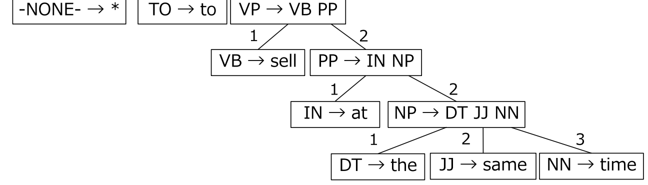


Fig. 3 Derivation tree

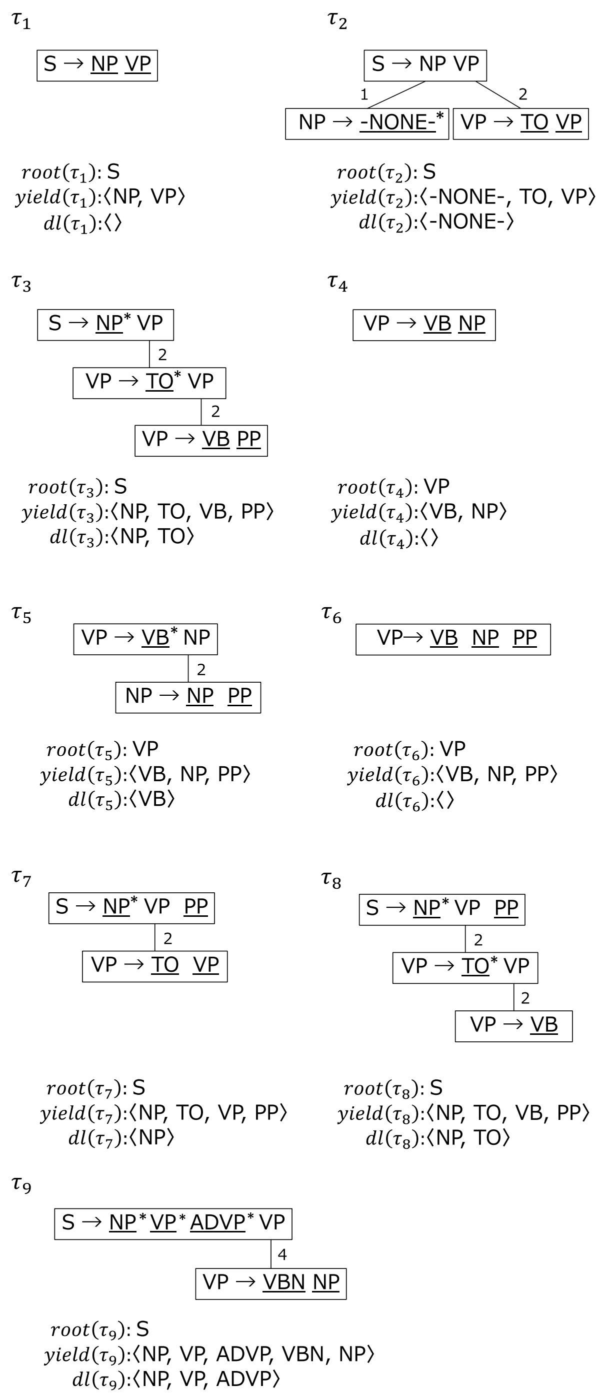


Fig. 4 Examples of patterns

assigned to a node, we call such element a *leaf element*. A leaf element corresponds to a leaf node of the original parse tree pattern. In Fig. 4, leaf elements are underlined.

3.1.3 Error Correction Rule

As described in Sect. 2, Kato and Matsubara [16] assume that the frequency of an incorrect pattern is very low. According to this assumption, we consider that an infrequent pattern which can be transformed to a frequent one is an annotation error pattern. Our method seeks such patterns in a treebank and constructs STSG rules which transform them to the corresponding frequent ones.

The following formula represents whether or not two patterns τ and τ' can be transformed to each other:

$$Trans(\tau, \tau') \equiv (root(\tau) = root(\tau') \\ \wedge yield(\tau) = yield(\tau'))$$

where $root(\tau)$ is the left-hand side of the grammar rule of τ's root and $yield(\tau)$ is the list of τ's leaf element. τ_3 and τ_8 shown in Fig. 4 can be transformed to each other since $Trans(\tau_3, \tau_8)$ is satisfied.

We say that a pattern τ is frequent if $f(\tau) \geq \sigma$ where σ is a threshold. Let T be a treebank. Let $P(T)$ be the set of all patterns in T and $F(T)$ be the set of frequent patterns in T. The following set $Rule(T)$ is the set of rules our method constructs from T:

$$Rule(T) = \{\langle \tau_s, \tau_t \rangle \in P(T) \times P(T) \mid \tau_s \notin F(T) \wedge \tau_t \in F(T) \\ \wedge Trans(\tau_s, \tau_t)\}$$

3.2 Outline of Our Method

If we enumerated all tree patterns included in a treebank, we could easily obtain any kind of rules which can be extracted from the treebank. However, such naive method is intractable because it requires an exponential computational complexity. To construct rules efficiently, our method avoids the enumeration of patterns which do not contribute to error correction by using a tree mining technique.

The procedure of our method is as follows:

1. Enumerate frequent patterns in a treebank by using a tree mining algorithm FREQT [19].
2. Seek infrequent patterns which can be transformed to frequent ones.
3. Construct STSG rules which transform infrequent patterns to frequent ones.

3.3 FREQT

In this section, we explain FREQT [19], which is the basis of our method. FREQT efficiently enumerates all frequent patterns in a tree set. Figure 5 shows the algorithm of FREQT. First, FREQT creates the set $\mathcal{F}_1$ of all frequent 1-patterns by traversing a treebank T. Next, the algorithm generates candidate 2-patterns by expanding each frequent 1-pattern $\tau \in \mathcal{F}_1$ by attaching a new node. For each candidate 2-pattern τ', if $f(\tau') \geq \sigma$, τ' is added to $\mathcal{F}_2$. The algorithm iteratively expands frequent $(k-1)$-patterns, and adds frequent k-patterns to $\mathcal{F}_k$. By continuing this process until no pattern is generated, FREQT enumerates all frequent patterns.

FREQT uses the rightmost expansion technique. FREQT expands a pattern by attaching a new node v to a

```
Algorithm FREQT
Input: A threshold $\sigma > 0$, a treebank $T$.
Output: The set $\mathcal{F}$ of all frequent patterns in $T$.
 1: $\mathcal{F}_1 := \emptyset$
 2: for each 1-pattern $\tau$ which appears in $T$ do
 3:     if $f(\tau) \geq \sigma$ then
 4:         $\mathcal{F}_1 := \mathcal{F}_1 \cup \{\tau\}$
 5: $k := 2$
 6: while $\mathcal{F}_{k-1} \neq \emptyset$ do
 7:     $\mathcal{F}_k := \emptyset$
 8:     for each $\tau \in \mathcal{F}_{k-1}$ do
 9:         for each $\tau'$ s.t. $\tau \Rightarrow \tau'$ do
10:             if $f(\tau') \geq \sigma$ then
11:                 $\mathcal{F}_k := \mathcal{F}_k \cup \{\tau'\}$
12:     $k := k + 1$
13: Return $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2 \cup \cdots \cup \mathcal{F}_{k-1}$.
```

Fig. 5 Algorithm of FREQT

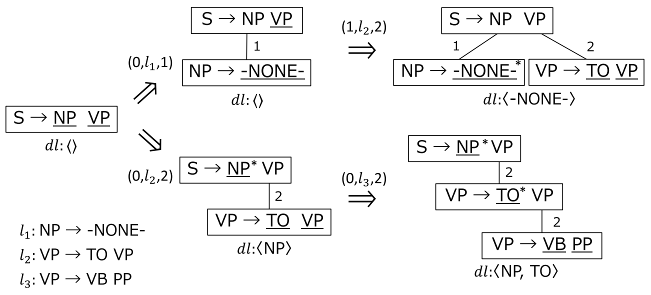


Fig. 6 Examples of expansions

node v' on the *rightmost branch* of the pattern as a rightmost child of v'. The rightmost branch of a pattern is defined as the path starting from the root to the *rightmost leaf*. When l is the label of v, v' is the p-th parent of the rightmost leaf and the label of new edge between v and v' is i, we call a generated pattern τ' the (p, l, i)-expansion of τ and we write $\tau \Rightarrow \tau'$. For example, in Fig. 4, τ_8 is the (0,VP → VB,2)-expansion of τ_7. The rightmost expansion technique enables FREQT to enumerate all candidate patterns without overlapping. Figure 6 shows examples of expansions.

3.4 Constructing Error Correction Rules

After calculating $F(T)$ by FREQT, our method seeks infrequent source patterns by expanding infrequent patterns. For an infrequent pattern τ_s, if there exists some $\tau_t \in F(T)$ s.t. $Trans(\tau_s, \tau_t)$, our method constructs the rule $\langle \tau_s, \tau_t \rangle$.

3.4.1 Efficient Enumeration of Infrequent Source Patterns

To seek infrequent source patterns efficiently, we focus on leaf elements of patterns. In our method, pattern expansion proceeds by applying grammar rule to leaf elements from left to right. Once a leaf element is skipped, it never has a grammar rule. We call such element a *determined leaf*. In Figs. 4 and 6, determined leaves are marked with an asterisk. Our method expands a pattern τ only if τ satisfies the following condition:

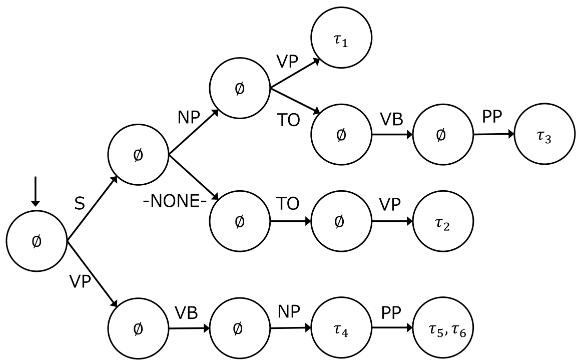


Fig. 7 Example of transducer

$$\begin{aligned} &\exists \tau_t (\tau_t \in F(T) \\ &\quad \wedge\ root(\tau) = root(\tau_t) \\ &\quad \wedge\ dl(\tau) \text{ is a prefix of } yield(\tau_t)) \end{aligned} \tag{1}$$

where $dl(\tau)$ is the list of determined leaves of a pattern τ. Here, $\alpha \cdot \beta$ is the list generated by concatenating a list β to a list α. The condition (1) is rewritten as follows:

$$root(\tau) \cdot dl(\tau) \in prefix(L) \tag{2}$$

where L is the set defined as follows:

$$L = \{\alpha \mid \exists \tau_t \in F(T), \alpha = root(\tau_t) \cdot yield(\tau_t)\}$$

and $prefix(L)$ is the set of prefixes of L's elements. If a pattern τ does not satisfy the condition (2), τ does not contribute to constructing $Rule(T)$. This is because for any τ' s.t. $\tau \Rightarrow^* \tau'$, there is no target pattern $\tau_t \in F(T)$ which satisfies $Trans(\tau', \tau_t)$. For example, let us assume $F(T) = \{\tau_1, \tau_2, \ldots, \tau_6\}$ in Fig. 4. Here, $root(\tau_7) \cdot dl(\tau_7)$ is ⟨S, NP⟩. This is a prefix of $root(\tau_3) \cdot yield(\tau_3)$ = ⟨S, NP, TO, VB, PP⟩. Thus $root(\tau_7) \cdot dl(\tau_7) \in prefix(L)$. By expanding τ_7, we can obtain τ_8. τ_8 can be transformed to τ_3, that is, τ_7 contributes to constructing $Rule(T)$. On the other hand, $root(\tau_9) \cdot dl(\tau_9)$ is ⟨S, NP, VP, ADVP⟩. This is not included in $prefix(L)$. For any τ'_9 s.t. $\tau_9 \Rightarrow^* \tau'_9$, $root(\tau'_9) \cdot yield(\tau'_9)$ are in the form of ⟨S, NP, VP, ADVP, ...⟩. This means that $Trans(\tau'_9, \tau_t)$ does not hold for any pattern $\tau_t \in F(T)$, that is, τ_9 does not contribute to constructing $Rule(T)$.

Our method checks whether or not a pattern satisfies the condition (2) by using a transducer. We make a deterministic transducer $TD = (\Sigma, Q, q_0, \delta, O)$ which accepts L. Figure 7 shows the transducer when $F(T)$ is $\{\tau_1, \tau_2, \ldots, \tau_6\}$ in Fig. 4. Q is the set of states contained in TD. Each state corresponds to an element of $prefix(L)$. The initial state q_0 corresponds to an empty list. An input symbol is a word or a label in the treebank. Σ is for the set of input symbols. Let q_α be a state which corresponds to a list $\alpha \in prefix(L)$. The transition function $\delta : Q \times \Sigma \rightarrow Q$ is defined as follows:

$$\delta(q_\alpha, a) = \begin{cases} q_{\alpha \cdot a} & (\alpha \cdot a \in prefix(L)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

If $\delta(q_0, root(\tau) \cdot dl(\tau)) \in Q$, a pattern τ satisfies the condition

```
Algorithm Create error correction rules
Input: A threshold $\sigma > 0$, a treebank $T$.
Output: The set $Rule$ of error correction rules.
 1: Calculate the set $\mathcal{F}$ of all frequent patterns in $T$ by FREQT.
 2: Make a transducer $TD = (\Sigma, Q, q_0, \delta, O)$ from $\mathcal{F}$.
 3: $C_1 := \emptyset$
 4: $Rule := \emptyset$
 5: for each 1-pattern $\tau$ which appears in $T$ do
 6:     if $q := \delta(q_0, root(\tau)) \in Q$ then
 7:         $C_1 := C_1 \cup \{\langle \tau, q \rangle\}$
 8:         if $f(\tau) < \sigma$ then
 9:             for each $\tau_t \in O(\delta(q, yield(\tau)))$ do
10:                 $Rule := Rule \cup \{\langle \tau, \tau_t \rangle\}$
11: $k := 2$
12: while $C_{k-1} \neq \emptyset$ do
13:     $C_k := \emptyset$
14:     for each $\langle \tau, q \rangle \in C_{k-1}$ do
15:         for each $\tau'$ s.t. $\tau'$ is the $(p,l,i)$-expansion of $\tau$ and $f(\tau') > 0$ do
16:             if $q' := \delta(q, new\_dl(\tau, p, i)) \in Q$ then
17:                 $C_k := C_k \cup \{\langle \tau', q' \rangle\}$
18:                 if $f(\tau') < \sigma$ then
19:                     for each $\tau_t \in O(\delta(q', yield(\tau')/dl(\tau')))$ do
20:                         $Rule := Rule \cup \{\langle \tau', \tau_t \rangle\}$
21:     $k := k + 1$
22: Return $Rule$.
```

Fig. 8 Algorithm of creating error correction rules

(2). For a state $q \in Q$, TD outputs a set of frequent patterns. The output function $O : Q \to 2^{F(T)}$ is defined as follows:

$$O(q) = \{\tau \in F(T) \mid \delta(q_0, root(\tau) \cdot yield(\tau)) = q\}$$

For all $\tau_t \in O(\delta(q_0, root(\tau) \cdot yield(\tau)))$, $Trans(\tau, \tau_t)$ holds.

Figure 8 shows the algorithm of creating error correction rules. In the first step, the algorithm adds 1-patterns which satisfy the condition (2) to a set C_1 (lines 5 – 7). For each pattern τ in C_k, C_k keeps the state q s.t. $q = \delta(q_0, root(\tau) \cdot dl(\tau))$. Then, for each pattern τ in C_1, if τ is infrequent, the algorithm creates error correction rules from τ (lines 8 – 10). In the subsequent steps, the algorithm creates C_k ($k \geq 2$) by expanding patterns in C_{k-1} (lines 14 – 17). Here, $new\_dl(\tau, p, i)$ is a function which calculates a list of leaf elements which newly become determined leaves in the (p, l, i)-expansion of τ. That is, when a pattern τ' is the (p, l, i)-expansion of τ, $dl(\tau') = dl(\tau) \cdot new\_dl(\tau, p, i)$ holds. Therefore, the following equation holds:

$$\delta(q_0, root(\tau') \cdot dl(\tau')) = \delta(q, new\_dl(\tau, p, i))$$

where $q = \delta(q_0, root(\tau) \cdot dl(\tau))$. Like the first step, for each pattern τ' in C_k, if τ' is infrequent, the algorithm creates error correction rules from τ' (lines 18 – 20). Here, α/β is the list s.t. $\alpha = \beta \cdot \alpha/\beta$.

Figure 9 shows the algorithm of $new\_dl(\tau, p, l)$. $v.RME$ is the label of the edge which connects a node v and the rightmost child of v. If v has no child, $v.RME$ is 0. $L(v).RHS(i)$ is the i-th element of the right-hand side of v's label. $|L(v).RHS|$ is the number of elements of the right-hand side of v's label. For example, in Fig. 4, τ_8 is the (0,VP $\to$ VB,2)-expansion of τ_7. So, $new_dl(\tau_7, 0, 2) = \langle$TO$\rangle$ and $dl(\tau_8) = dl(\tau_7) \cdot new_dl(\tau_7, 0, 2) = \langle$NP, TO$\rangle$.

```
Algorithm $new\_dl(\tau, p, i)$
1: $NDL := \langle\rangle$
2: $v :=$ the rightmost leaf of $\tau$
3: for $j := 0$ to $p - 1$ do
4:     for $k := v.RME + 1$ to $L(v).|RHS|$ do
5:         $NDL := NDL \cdot L(v).RHS(k)$
6:     $v := v.parent$
7: for $j := v.RME + 1$ to $i - 1$ do
8:     $NDL := NDL \cdot L(v).RHS(j)$
9: Return $NDL$.
```

Fig. 9 Algorithm of $new\_dl(\tau, p, l)$

Table 1 The number of rules which two methods obtained

		our method		
		obtained	not obtained	total
K&M	obtained	392	3,946	4,338
	not obtained	1,987		
	total	2,379		6,325

4. Experiment

We performed an experiment to evaluate our method. We applied our method to 49,208 sentences in the Wall Street Journal section of the Penn Treebank [17]. We implemented our method in Java. The experiment was run on a PC (Intel core i7 3.40GHz) with 8GB main memory, running Windows 7 Professional. The threshold σ was set to 100. We obtained 2,379 rules. This took about 34 minutes[†]. Table 1 shows the number of rules which were obtained by our method and Kato and Matsubara's method. Our method obtained 1,987 rules which Kato and Matsubara's method did not. This means that our method increases the coverage of error correction. On the other hand, 3,946 rules can not be obtained by our method. However, this is not disadvantageous since we can use all rules together.

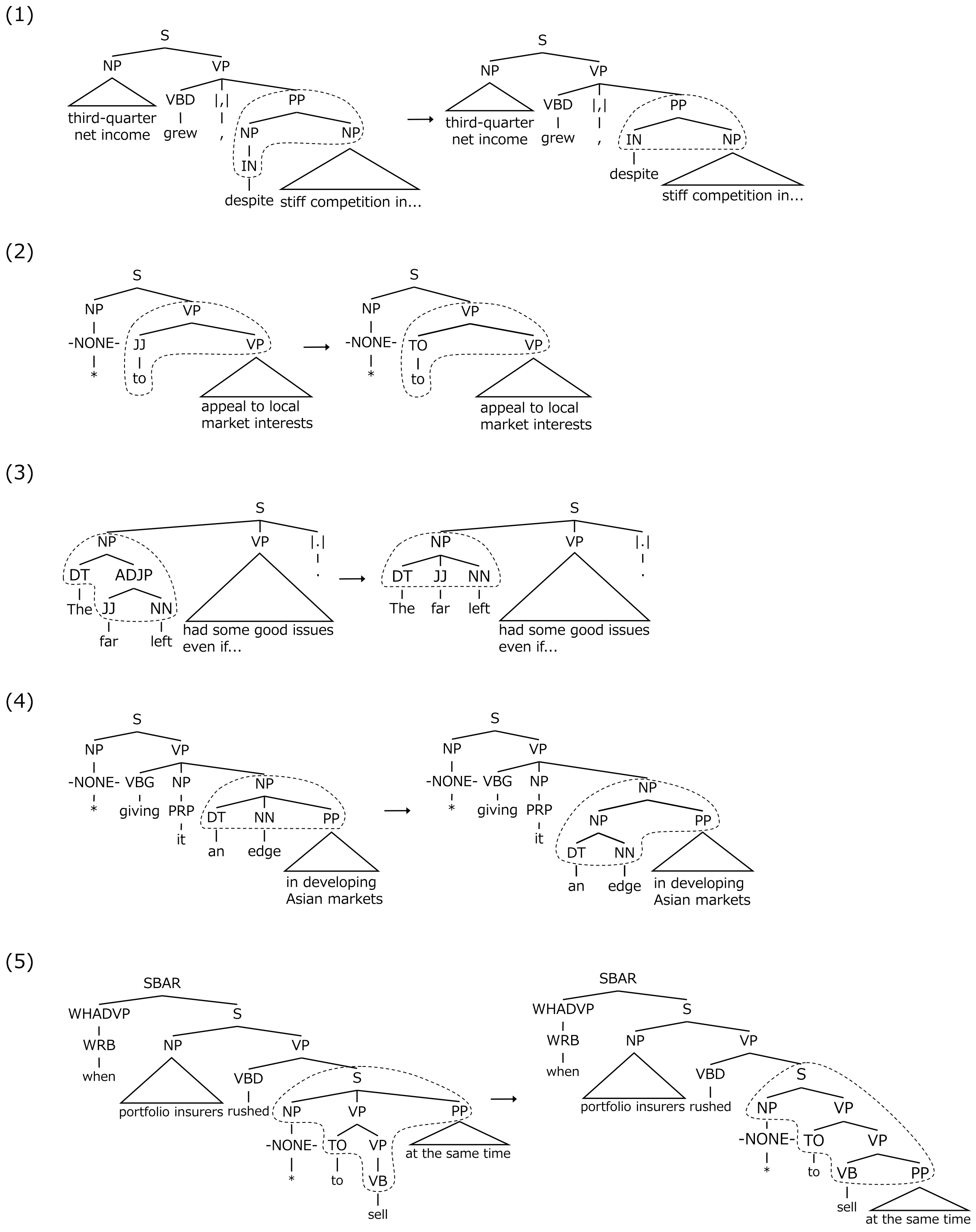
To measure the precision of the rules which our method obtains, we applied the rules to the WSJ section[††]. Because it is time-consuming and expensive to evaluate all the rules, we only evaluated the rules with the 300 highest scores. A person (not the authors) manually checks whether or not each rule corrects errors. The precision p is measured in the same way as [16]:

$$p = \frac{\textit{\# of the positions where an error is corrected}}{\textit{\# of the positions to which some rule is applied}}$$

The number of the positions to which 300 rules are applied is 605. The number of the positions where an error is corrected is 466. Therefore, the precision of our method is 77.0%. The precision of the previous method [16] is 71.6%. We measured the precision of each rule. The precision of 196 rules reached 100%. 155 of the 196 rules could not

[†]A naive method which enumerates all patterns could not work.

[††]It is difficult to measure the recall of our method because it requires the treebank which includes no error.



(1)
S NP VP VBD |,| PP NP NP IN
third-quarter net income grew , despite stiff competition in...
→
S NP VP VBD |,| PP IN NP
third-quarter net income grew , despite stiff competition in...
(2)
S NP VP -NONE- * JJ VP to
appeal to local market interests
→
S NP VP -NONE- * TO VP to
appeal to local market interests
(3)
S NP VP |.| . DT ADJP JJ NN The far left
had some good issues even if...
→
S NP VP |.| . DT JJ NN The far left
had some good issues even if...
(4)
S NP VP -NONE- * VBG giving NP PRP it NP DT NN PP an edge
in developing Asian markets
→
S NP VP -NONE- * VBG giving NP PRP it NP NP PP DT NN an edge
in developing Asian markets
(5)
SBAR WHADVP WRB when S NP VP portfolio insurers VBD rushed S NP VP PP -NONE- * TO to VP VB sell
at the same time
→
SBAR WHADVP WRB when S NP VP portfolio insurers VBD rushed S NP VP -NONE- * TO to VP VB sell PP
at the same time

Fig. 10 Examples of correcting syntactic annotation errors

be obtained by Kato and Matsubara's method. Those results show that our method can obtain the useful error correction rules which the previous method can not obtain and therefore increases the coverage of error correction. Figure 10 shows some examples of correcting errors which our method corrects but the previous method does not. The partial parse trees enclosed within the dotted lines are sources and targets. Rule (1) deletes a useless node NP. Rule (2) replaces a label JJ with the correct label TO. Rule (3)-(5) corrects structural annotation errors.

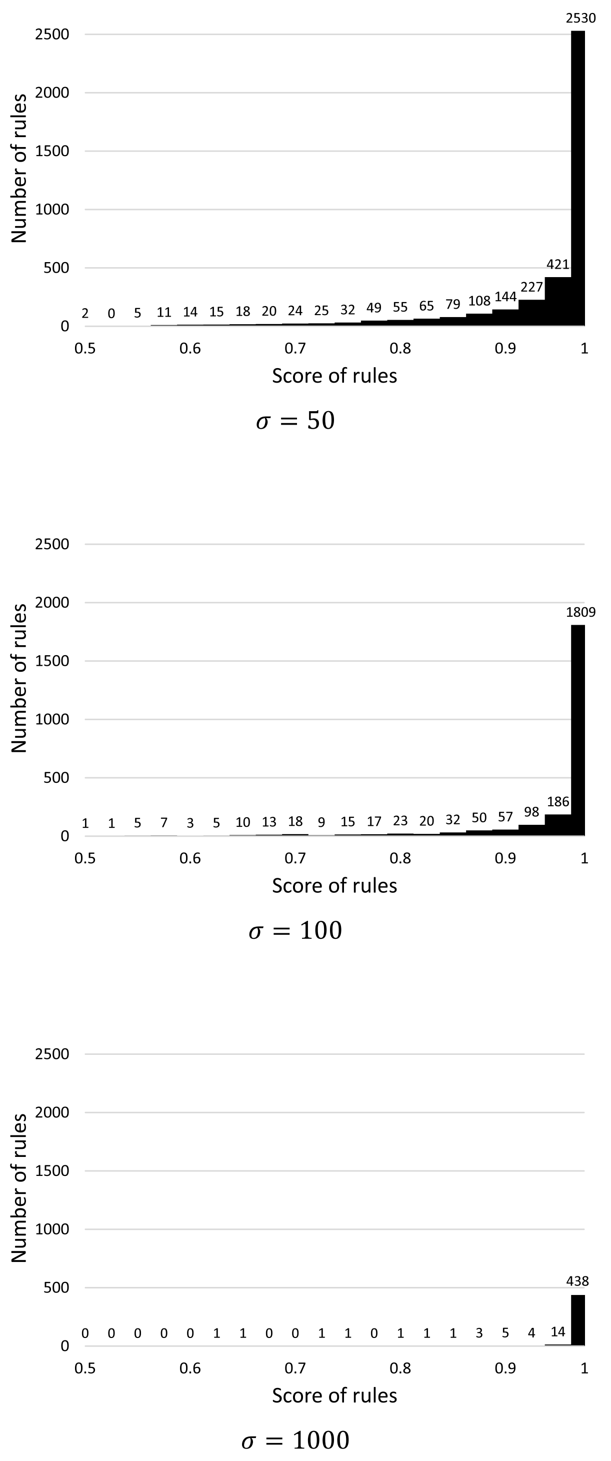


Fig. 11 The distributions of rules for scores when the threshold σ is varied

Figure 11 shows the distribution of the rules for scores when the threshold σ is varied. These graphs show that our method tends to obtain rules with high scores. This tendency became stronger in the case where the threshold σ was set to a high value, while the total number of obtained rules decreased.

5. Conclusion

In this paper, we proposed a new method of correcting annotation errors in a treebank. Our method is based on tree mining. An experiment showed that our method can obtain rules which the previous method can not obtain. The proposed method and the previous one are complementary. That is, by both methods, we can expect to achieve wider coverage of error correction.

If a source pattern has several target patterns, our method simply transforms the source to the most frequent target. Furthermore, there exist cases where source patterns do not include errors. To improve the precision, we will explore how to select a correct target and how to identify whether or not an occurrence of a source pattern includes errors.

Acknowledgments

This research was partially supported by the Grand-in-Aid for Scientific Research (B) (No. 26280082) of JSPS.

References

[1] K. Suzuki, Y. Kato, and S. Matsubara, "Correcting errors in a treebank based on tree mining," Proc. 10th International Conference on Language Resources and Evaluation, pp.1540–1545, 2016.

[2] E. Eskin, "Detecting errors within a corpus using anomaly detection," Proc. 1st North American Chapter of the Association for Computational Linguistics, pp.148–153, 2000.

[3] M. Dickinson and W.D. Meurers, "Detecting errors in part-of-speech annotation," Proc. 10th Conference of the European Chapter of the Association for Computational Linguistics, pp.107–114, 2003.

[4] M. Murata, M. Utiyama, K. Uchimoto, H. Isahara, and Q. Ma, "Correction of errors in a verb modality corpus for machine translation with a machine-learning method," ACM Trans. Asian Language Information Processing, vol.4, no.1, pp.18–37, 2005.

[5] M. Dickinson, "From detecting errors to automatically correcting them," Proc. 11th Conference of the European Chapter of the Association for Computational Linguistics, pp.265–272, 2006.

[6] M. Dickinson and C.M. Lee, "Detecting errors in semantic annotation.," Proc. 6th International Conference on Language Resources and Evaluation, pp.605–610, 2008.

[7] M. Dickinson, "Correcting dependency annotation errors," Proc. 12th Conference of the European Chapter of the Association for Computational Linguistics, pp.193–201, 2009.

[8] M. Dickinson and W.D. Meurers, "Detecting inconsistencies in treebanks," Proc. 2nd Workshop on Treebanks and Linguistic Theories, pp.45–56, 2003.

[9] T. Ule and K. Simov, "Unexpected productions may well be errors.," Proc. 4th International Conference on Language Resources and Evaluation, pp.1795–1798, 2004.

[10] M. Dickinson and W.D. Meurers, "Prune diseased branches to get healthy trees! How to find erroneous local trees in a treebank and why it matters," Proc. 4th Workshop on Treebanks and Linguistic Theories, pp.41–52, 2005.

[11] A. Boyd, M. Dickinson, and W.D. Meurers, "Increasing the recall of corpus annotation error detection," Proc. 6th Workshop on Treebanks and Linguistic Theories, pp.19–30, 2007.

[12] M. Dickinson, "Similarity and dissimilarity in treebank grammars," Current Issues in Unity and Diversity of Languages: Collection of

the papers selected from the 18th International Congress of Linguists, pp.1597–1611, 2009.
[13] A. Przepiórkowski and M. Lenart, "Simultaneous error detection at two levels of syntactic annotation," Proc. 6th Linguistic Annotation Workshop, pp.118–123, 2012.
[14] S. Kulick, A. Bies, J. Mott, M. Maamouri, B. Santorini, and A. Kroch, "Using derivation trees for informative treebank inter-annotator agreement evaluation," Proc. 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp.550–555, 2013.
[15] P. Faria, "Using dominance chains to detect annotation variants in parsed corpora," 10th IEEE International Conference on e-Science, pp.25–32, 2014.
[16] Y. Kato and S. Matsubara, "Correcting errors in a treebank based on synchronous tree substitution grammar," Proc. ACL 2010 Conference Short Papers, pp.74–79, 2010.
[17] M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz, "Building a large annotated corpus of English: the Penn Treebank," Computational Linguistics, vol.19, no.2, pp.313–330, 1993.
[18] J. Eisner, "Learning non-isomorphic tree mappings for machine translation," Proc. 41st Annual Meeting of the Association for Computational Linguistics, pp.205–208, 2003.
[19] T. Asai, K. Abe, S. Kawasoe, H. Sakamoto, H. Arimura, and S. Arikawa, "Efficient substructure discovery from large semi-structured data," IEICE Trans. Inf. & Syst., vol.E87-D, no.12, pp.2754–2763, 2004.

Kanta Suzuki received the B.E. degree in 2015 in information engineering from Nagoya University. He is currently a graduate student at the Graduate School of Information Science, Nagoya University. His research interests include natural language processing and data mining.

Yoshihide Kato received the B.E. degree, the M.E. degree, and the Dr. of Engineering degree in information engineering from Nagoya University, in 1997, 1999 and 2003, respectively. He was an Assistant Professor in Graduate School of International Development, Nagoya University, from 2003 to 2009. He was a Researcher at Information Technology Center, Nagoya University, from 2009 to September 2011, and a Designated Assistant Professor, from October 2011 to March 2012. He is currently an Associate Professor at Information & Communications, Nagoya University. His research interests include natural language processing and information retrieval. He is a member of the IPSJ, the NLP and the ACL.

Shigeki Matsubara received the B.E. degree in electrical and computer engineering from Nagoya Institute of Technology in 1993, and the M.E. degree, and the Dr. of Engineering from Nagoya University in 1995 and 1998, respectively. After becoming an Assistant Professor at Nagoya University, he became an Associate Professor at Information Technology Center in 2002, and he is currently an Associate Professor in Graduate School of Information Science. His research interests include natural language processing, information retrieval and digital library. He is a member of the IPSJ, the JSAI and the NLP.