Correcting Errors in a Treebank Based on Tree Mining

Kanta Suzuki[1], **Yoshihide Kato**[2] and **Shigeki Matsubara**[1]
[1]Graduate School of Information Science, Nagoya University
[2]Information & Communications, Nagoya University
ksuzuki@db.ss.is.nagoya-u.ac.jp

Abstract

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 an infrequent 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 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.

Keywords: 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. So, detecting and correcting errors in annotated corpora are important tasks. Many studies suggest methods of detecting or correcting errors in various kinds of annotated corpora (see (Dickinson, 2015) for a survey). There are several methods of detecting annotation errors in a phrase structure treebank (Dickinson and Meurers, 2003; Ule and Simov, 2004; Dickinson and Meurers, 2005; Boyd et al., 2007; Dickinson, 2009; Przepiórkowski and Lenart, 2012; Kulick et al., 2013; Faria, 2014). However, there is little work on treebank error correction.

One exception is the work of Kato and Matsubara (2010). 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. 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.

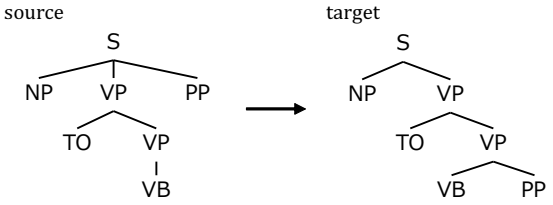

Figure 1: An example of STSG rule

2. Previous Work

Kato and Matsubara (2010) propose a method of correcting annotation errors in a treebank. Their method is based on *synchronous tree substitution grammar* (STSG) (Eisner, 2003). 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 *source*, and the other is called *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 one 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:

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

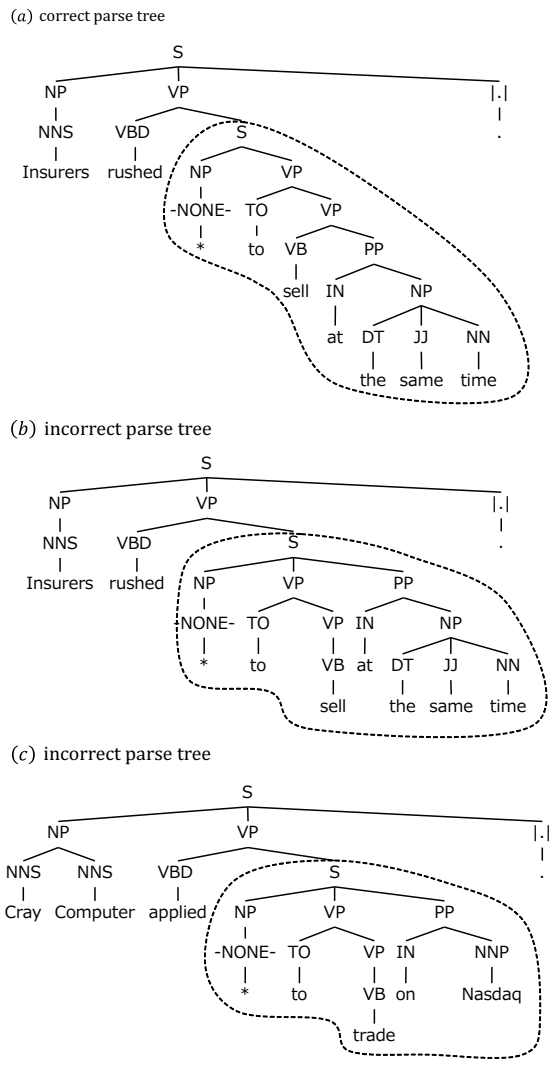

Figure 2: Examples of parse trees

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 Figure 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 Figure 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. So, 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 makes a pair of partial parse trees enclosed within the dotted lines in (a) and (c). This

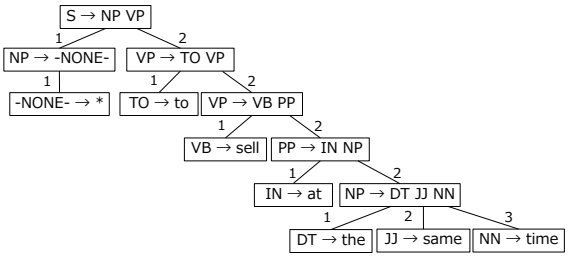

Figure 3: A derivation tree

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 Section 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 Figure 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:

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

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 Figure 3. A pattern with k nodes is called 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 assigned to a node, we call such element *leaf element*. A leaf element corresponds to a leaf node of the original parse tree pattern. In Figure 4, leaf elements are underlined.

3.1.3. Error Correction Rule

As described in Section 2, Kato and Matsubara (2010) 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 corresponding frequent ones.

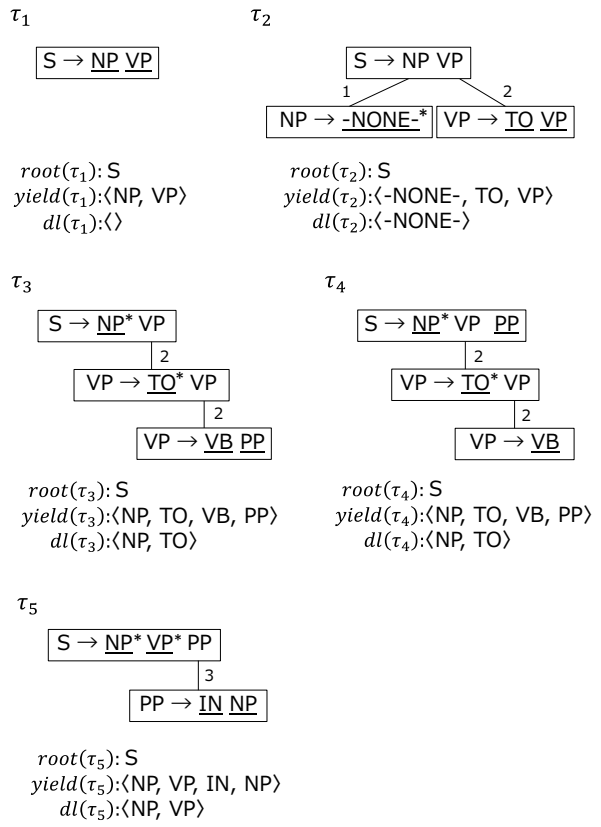

Figure 4: Examples of patterns

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

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

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_4 shown in Figure 4 can be transformed to each other since $Trans(\tau_3, \tau_4)$ 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 $F(T)$ be the set of frequent patterns in T. The following set $Rule(T)$ is the set of rules our method constructs from T:

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

3.2. Outline of Our Method

If we can enumerate all tree patterns included in a treebank and construct STSG rules from them, we can 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 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 (Asai et al., 2004).
2. Seek infrequent patterns which can be transformed to frequent ones.
3. Construct STSG rules which transform infrequent patterns to frequent ones.

Algorithm FREQT
Input: A threshold $\sigma > 0$, a treebank T.
Output: The set $\mathcal{F}$ of all frequent patterns in T.

```
F_1 := ∅
for each 1-pattern τ which appears in T do
  if f(τ) ≥ σ then
    F_1 := F_1 ∪ {τ}
k := 2
while F_{k-1} ≠ ∅ do
  F_k := ∅
  for each τ ∈ F_{k-1} do
    for each τ' s.t. τ ⇒ τ' do
      if f(τ') ≥ σ then
        F_k := F_k ∪ {τ'}
  k := k + 1
Return F = F_1 ∪ F_2 ∪ ··· ∪ F_{k-1}.
```

Figure 5: The algorithm of FREQT

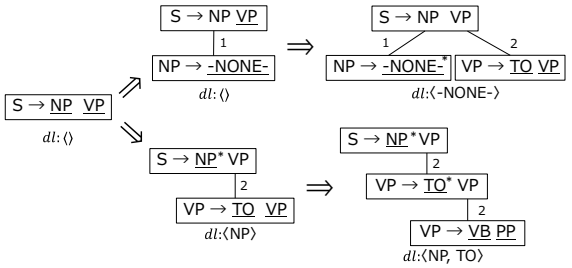

Figure 6: Examples of expansions

3.3. FREQT

In this section, we explain FREQT (Asai et al., 2004), 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 (We write $\tau \Rightarrow \tau'$ when τ' is obtained by expanding τ). 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 patterns are generated, FREQT enumerates all frequent patterns.

FREQT uses the rightmost expansion technique. When FREQT expands a pattern, a new node must be attached to a node on the rightmost branch of the pattern. This enables FREQT to enumerate all candidate pattern 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.

Algorithm Enumerate infrequent patterns
Input: A threshold $\sigma > 0$, a treebank T, the set $\mathcal{F}$ of all frequent patterns in T.
Output: The set $\mathcal{I}$ of infrequent patterns which includes all source patterns.

$\mathcal{C}_1 := \emptyset$
$\mathcal{I}_1 := \emptyset$
for each 1-pattern τ which appears in T **do**
  if there exists $\tau_t \in \mathcal{F}$ s.t. $root(\tau) = root(\tau_t)$ **then**
    $\mathcal{C}_1 := \mathcal{C}_1 \cup \{\tau\}$
    if $f(\tau) < \sigma$ **then**
      $\mathcal{I}_1 := \mathcal{I}_1 \cup \{\tau\}$
$k := 2$
while $\mathcal{C}_{k-1} \neq \emptyset$ **do**
  $\mathcal{C}_k := \emptyset, \mathcal{I}_k := \emptyset$
  for each $\tau \in \mathcal{C}_{k-1}$ **do**
    for each τ' s.t. $\tau \Rightarrow \tau' \wedge f(\tau') > 0$ **do**
      if there exists $\tau_t \in \mathcal{F}$ s.t. $root(\tau') = root(\tau_t)$ and $dl(\tau')$ is a prefix of $yield(\tau_t)$ **then**
        $\mathcal{C}_k := \mathcal{C}_k \cup \{\tau'\}$
        if $f(\tau') < \sigma$ **then**
          $\mathcal{I}_k := \mathcal{I}_k \cup \{\tau'\}$
  $k := k + 1$
Return $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2 \cup \cdots \cup \mathcal{I}_{k-1}$.

Figure 7: The algorithm of enumerating 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. According as pattern expansion proceeds from left to right, it is determined whether or not a grammar rule is applied to a leaf element. Once a leaf element is skipped, it never has a grammar rule. We call such element *determined leaf*. In Figure 4 and 6, determined leaves are marked with an asterisk. Our method expands a pattern τ only if there exists a frequent pattern τ_t which fulfills the following conditions:

1. $root(\tau) = root(\tau_t)$.
2. $dl(\tau)$ is a prefix of $yield(\tau_t)$.

where $dl(\tau)$ is the list of determined leaves of a pattern τ. If a pattern τ has no pattern τ_t satisfying the above conditions, the pattern τ does not contribute to constructing $Rule(T)$. This is because there is no target pattern $\tau_t \in F(T)$ for any τ' s.t. $\tau \Rightarrow^* \tau'$. That is, $Trans(\tau', \tau_t)$ does not hold for any τ' and τ_t. As an example, let us consider the patterns τ_4 and τ_5 shown in Figure 4. Here, $dl(\tau_5)$ is $\langle$NP, VP$\rangle$ and $yield(\tau_4)$ is $\langle$NP, TO, VB, PP$\rangle$. This pair does not fulfill the condition 2. For any τ'_5 s.t. $\tau_5 \Rightarrow^* \tau'_5$, $dl(\tau'_5)$ and $yield(\tau'_5)$ are in the form of $\langle$NP, VP, $\dots\rangle$. Therefore, $yield(\tau'_5) \neq yield(\tau_4)$. This also means that $Trans(\tau'_5, \tau_4)$ does not hold.
Figure 7 shows our algorithm of enumerating infrequent patterns.

4. Experiment

We performed an experiment to evaluate our method. We applied our method to 49,208 sentences in Wall Street Journal section of the Penn Treebank (Marcus et al., 1993). 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[1]. In these rules, 1,987 rules can not be obtained by Kato and Matsubara's method. To measure the precision of the rules, we applied rules to the WSJ section. Because it is time-consuming and expensive to evaluate all 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 (Kato and Matsubara, 2010):

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

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 (Kato and Matsubara, 2010) is 71.6%. We measured the precision of each rule. The precision of 196 rules achieved 100% precision. 155 of the 196 rules could not be obtained by Kato and Matsubara's method. This result shows that our method can obtain the useful error correction rules which the previous method can not obtain. Figure 8 shows some examples of correcting errors which our method correct but the previous method does not.

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

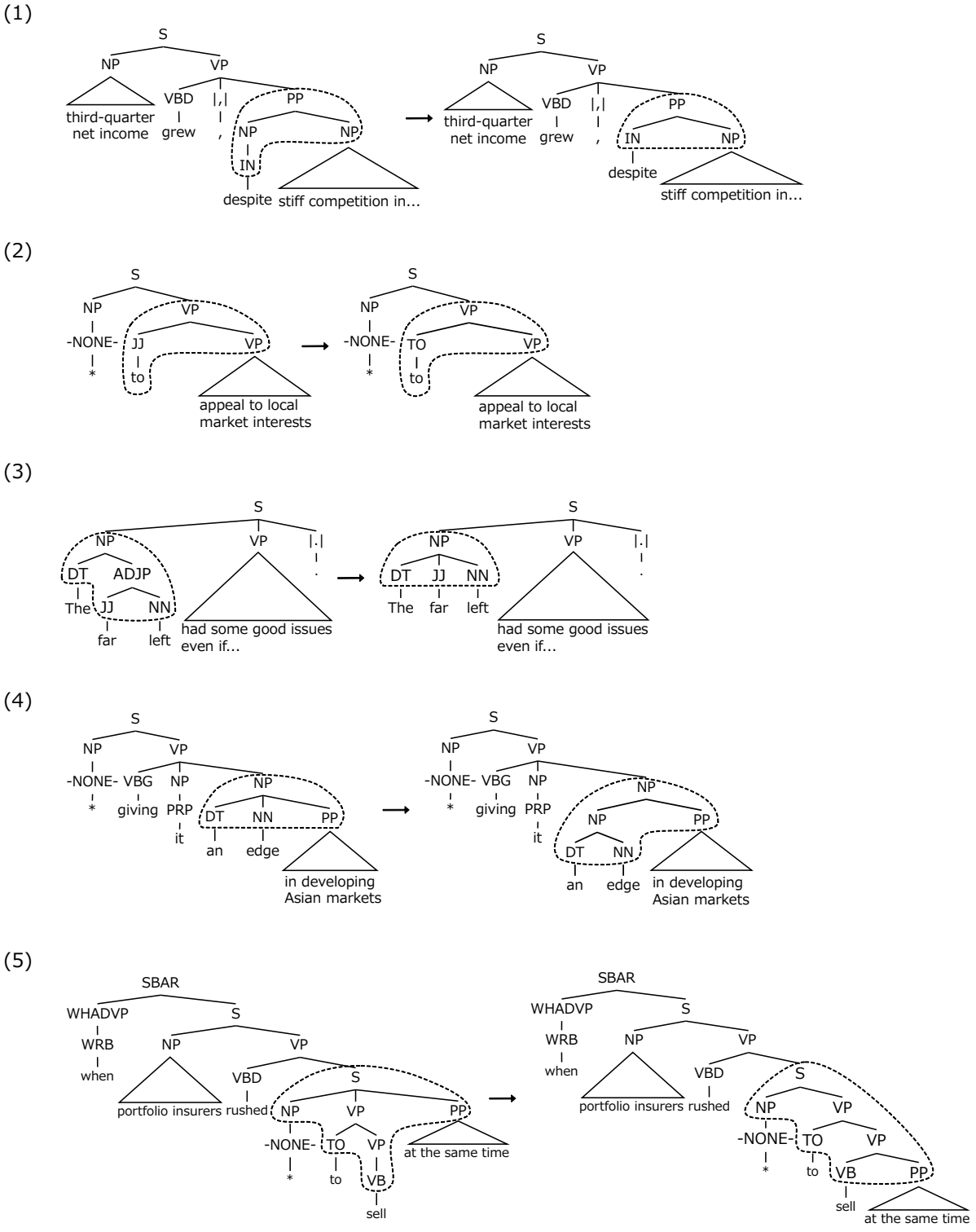

Figure 8: Examples of correcting syntactic annotation errors

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 method, 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. To improve the precision, we will explore how to select an appropriate target.

6. Acknowledgements

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

7. Bibliographical References

Tatsuya Asai, Kenji Abe, Shinji Kawasoe, Hiroshi Sakamoto, Hiroki Arimura, and Setsuo Arikawa. 2004. Efficient substructure discovery from large semi-structured data. *IEICE Transactions on Information and Systems*, E87-D(12):2754–2763.

Adriane Boyd, Markus Dickinson, and W. Detmar Meurers. 2007. Increasing the recall of corpus annotation error detection. In *Proceedings of the 6th Workshop on Treebanks and Linguistic Theories*, pages 19–30.

Markus Dickinson and W. Detmar Meurers. 2003. Detecting inconsistencies in treebanks. In *Proceedings of the 2nd Workshop on Treebanks and Linguistic Theories*, pages 45–56.

Markus Dickinson and W. Detmar Meurers. 2005. Prune diseased branches to get healthy trees! How to find erroneous local trees in a treebank and why it matters. In *Proceedings of the 4th Workshop on Treebanks and Linguistic Theories*, pages 41–52.

Markus Dickinson. 2009. Similarity and dissimilarity in treebank grammars. In *Current Issues in Unity and Diversity of Languages: Collection of the papers selected from the 18th International Congress of Linguists*, pages 1597–1611.

Markus Dickinson. 2015. Detection of annotation errors in corpora. *Language and Linguistics Compass*, 9(3):119–138.

Jason Eisner. 2003. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 205–208.

Pablo Faria. 2014. Using dominance chains to detect annotation variants in parsed corpora. In *10th IEEE International Conference on e-Science*, volume 2, pages 25–32.

Yoshihide Kato and Shigeki Matsubara. 2010. Correcting errors in a treebank based on synchronous tree substitution grammar. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 74–79.

Seth Kulick, Ann Bies, Justin Mott, Mohamed Maamouri, Beatrice Santorini, and Anthony Kroch. 2013. Using derivation trees for informative treebank inter-annotator agreement evaluation. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 550–555.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Adam Przepiórkowski and MichałLenart. 2012. Simultaneous error detection at two levels of syntactic annotation. In *Proceedings of the 6th Linguistic Annotation Workshop*, pages 118–123.

Tylman Ule and Kiril Simov. 2004. Unexpected productions may well be errors. In *Proceedings of the 4th International Conference on Language Resources and Evaluation*, pages 1795–1798.