半構造データマイニングに基づく構文木コーパスの誤り訂正

鈴木 寛大 † 加藤 芳秀 ‡ 松原 茂樹 §
† 名古屋大学工学部電気電子・情報工学科 ‡ 名古屋大学情報連携統括本部
§ 名古屋大学大学院情報科学研究科

1 はじめに

近年，様々なタグ付きコーパスが開発され，自然言語処理の研究に活用されている．しかし，タグ付きコーパスの作成には人手が介在するため，コーパスへの誤りの混入は避けがたい問題である．これに対し，タグ付きコーパス中の誤りを検出・訂正する手法が提案されている．品詞タグや係り受けタグの誤り訂正 [1, 2] では，各単語に付与されたタグの中から誤りを検出し，それを別のタグに置き換えることにより誤りを訂正している．しかし，句構造を表現する構文木コーパスは階層構造をとるため，各単語に付与されたタグの置き換えのみでは誤り訂正として不十分である．階層構造の違いによる誤りを検出・訂正する必要がある．

Kato らは構文木コーパスに含まれる誤りを検出･訂正する手法を提案している [3]．この手法では，コーパス中の複数箇所に出現する単語列の構文構造が異なる場合，一方を誤りを含むもの，他方をその誤りを取り除いたものと判断し，前者を後者へ変換する誤り訂正ルールを抽出する．しかし，コーパス中に誤りを含む部分木が存在しても，その部分木が覆う単語の列がコーパス中のほかの場所に出現していなければ，誤りを訂正するルールは獲得できない．

そこで本稿では，単語列を参照することなく，階層構造に対する誤り訂正ルールを抽出するために，半構造データマイニングに基づく手法を提案する．本手法ではまず，コーパス中に出現する頻出パターンを半構造データマイニングによって列挙し，次に，頻出パターンに変換可能な低出現頻度のパターンを抽出する．後者を前者へと変換するルールを誤り訂正ルールとする．Kato らの手法と同様に Penn Treebank[4] を用いた実験を行ったところ，Kato らの手法では獲得できなかった誤り訂正ルールが獲得できることを確認した．

2 従来手法

本節では，構文木コーパスの誤り訂正手法である Kato らの手法 [3] を概観し，その問題点を議論する．

Kato らの手法では，まず，コーパス中に出現する構文木の部分木のうち，根のラベル，及び葉のラベルの並び（単語列）が一致し，内部の構造の異なる二つの部分木を対にする．この部分木の対から構造が異なる部分のみを残し，構文構造を変換するパターンを獲得する．コーパス中で出現する頻度の高いパターンは正しいパターンである可能性が高く，頻度の低いパターンは誤りを含む可能性が高いと考えられるため，出現頻度の低い方を，出現頻度の高い方へと変換するルールを作成する．図 1 に示す構文木 $(a), (b)$ がコーパスに含まれていた場合，この二つの構文木に含まれる部分木（点線部）で対が作成され，図 2 に示す誤り訂正ルールが獲得できる． しかし，もし構文木 (b) がコーパス中に出現せず，同種の誤りを含み，単語列が異なる図 3 の構文木が出現する場合，部分木の対は作成されず，誤り訂正ルールも作成できない．

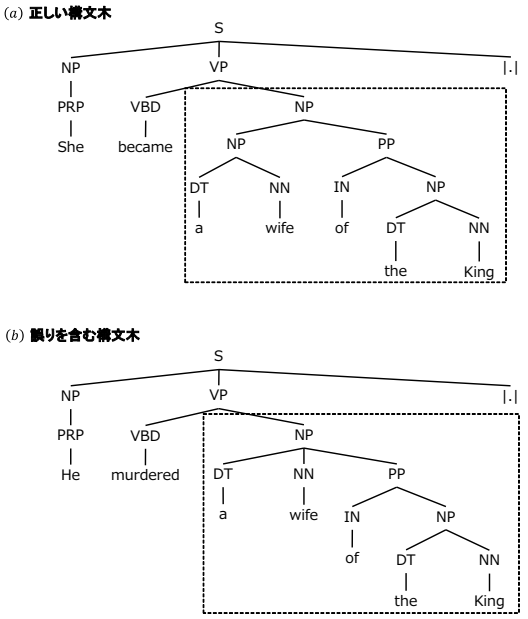

図 1: 構文木の例

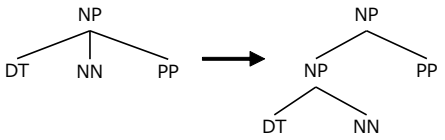

図 2: 誤り訂正ルールの例

3 半構造データマイニングに基づく誤り訂正ルールの獲得

2 節で述べた問題を解決するために，本節では，Kato らの手法と同種の誤り訂正ルールを，半構造データマイニングのアプローチに基づき抽出する手法を提案する．原理的には，コーパス中に出現するパターンすべてを抽出すれば，任意の誤り訂正ルールを構成できるが，コーパス中に出現する全てのパターンを列挙するには莫大な計算量を要する．そのため，本手法では，誤り訂正ルールの変換元，変換先になりえないパターンの列挙を回避することにより，効率的な誤り訂正ルールの獲得を実現する．

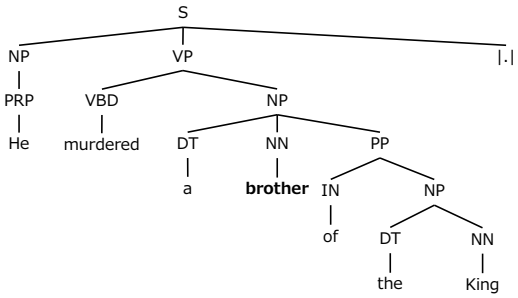

図 3: 誤りを含む構文木

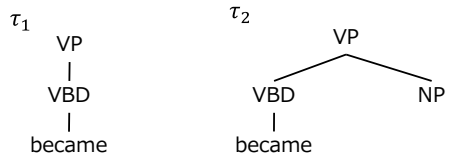

図 4: パターンの例

3.1 定義

手法について述べる前に，その準備として本手法に関していくつかの定義を与える．

3.1.1 パターン

構文木に含まれる任意の連結された部分グラフをパターンと呼ぶ．図 4 の τ_1, τ_2 は図 1 の構文木 (a) に含まれるパターンの例である．パターン τ の全ての内部節点について，その子の数が，構文木中で対応する節点の子の数と等しいとき，τ を完全パターンと呼ぶ．図 4 の τ_1 は完全パターンではなく，τ_2 は完全パターンである．

3.1.2 誤り訂正ルール

コーパス中に出現する構文木の集合を T とする．T 中の構文木に含まれる完全パターンの集合を $Pat(T)$ とする．このとき，完全パターン $\tau \in Pat(T)$ に変換可能な完全パターンの集合 $C(\tau)$ は以下のように定義される．

$$C(\tau) = \{\tau' | \tau' \in Pat(T) \wedge root(\tau) = root(\tau') \wedge yield(\tau) = yield(\tau')\}$$

$root(\tau)$ は τ の根のラベル，$yield(\tau)$ は τ の葉のラベルを左から順に並べた列である．つまり，τ' は τ と根と葉のラベルがそれぞれ一致する完全パターンである．誤り訂正ルールは，完全パターンの対 $\langle \tau', \tau \rangle$ から構成され，$\tau' \in C(\tau)$ が成り立つ．このルールは，τ' にマッチした部分を τ に変換することを意味する．

誤り訂正ルールは，Kato らの手法に従えば，低頻度のパターンを高頻度のパターンへと変換するルールと位置づけられる．完全パターン $\tau \in Pat(T)$ が T に出現する回数を $f(\tau)$ とし，事前に設定した閾値 σ に対して，$f(\tau) \geq \sigma$ を満たすとき，τ は頻出であるとする．T 中の頻出完全パターンの集合を $F(T)$ とすると，変換先 τ は $F(T)$ の要素である．τ を変換先とするような誤り訂正ルールの変換元の候補の集合 $Source(\tau)$ は以下のように定義できる．

$$Source(\tau) = \{\tau' \in C(\tau) | f(\tau') < \sigma\}$$

3.2 手法の概要

提案手法の概要は以下のとおりである．まず半構造データマイニングによって，コーパス中の頻出完全パターンを全て列挙する．これらは，誤り訂正ルールの変換先の候補である．頻出完全パターンの列挙には半構造データマイニングの代表的なアルゴリズムである FREQT[5] を使用する．次に，列挙した頻出完全パターンに変換可能な低頻度の完全パターンのみを抽出する．このようなパターンを列挙するために，提案手法では FREQT を拡張する．最後に，抽出された低頻度のパターンは頻出パターンの誤りであると判断し，前者を後者へ変換する誤り訂正ルールを獲得する．

3.3 FREQT

本節では，提案手法のベースとなるアルゴリズムである FREQT について説明する．FREQT は木の集積データ中の頻出パターンを効率よく列挙するアルゴリズムである．FREQT ではまず，集積データ中に出現する単一の節点から出現頻度が閾値以上のものだけを取り出す．次に，これらの節点それぞれに節点を追加し，サイズが 2 のパターンを作成する．節点を追加する操作を拡張と呼ぶ．FREQT では拡張を最右拡張のみに制限し，効率的に頻出パターンを列挙する．最右拡張では，あるパターンを拡張するとき，新しい節点はパターンの最右の経路に含まれる節点にのみ末弟として追加できる．これにより考えうるすべてのパターンを重複なく作成できる．拡張により得られた複数のパターンから，再び出現頻度が閾値以上のものだけを残して，最右拡張を行う．これを，新しいパターンが生成できなくなるまで繰り返すことによって，集積データ中に出現するすべての頻出パターンを列挙できる．

3.4 コーパス中の構文木の表現法

構文木コーパスに FREQT を適用すると，列挙されるのは頻出パターンであって，頻出完全パターンではない．そこで，構文木の頻出完全パターンのみを列挙するよう，構文木を導出過程木として表現する．構文木に対する導出過程木は，次のようにして得られる．まず，構文木の各内部節点 v に対応する節点 V を作る．このとき，親子関係は保存する．各節点 V に対して，以下の文法規則をラベルとして与える．

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

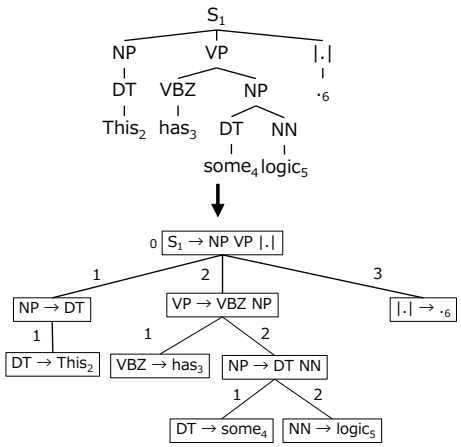

図 5: 構文木の導出過程木への変換

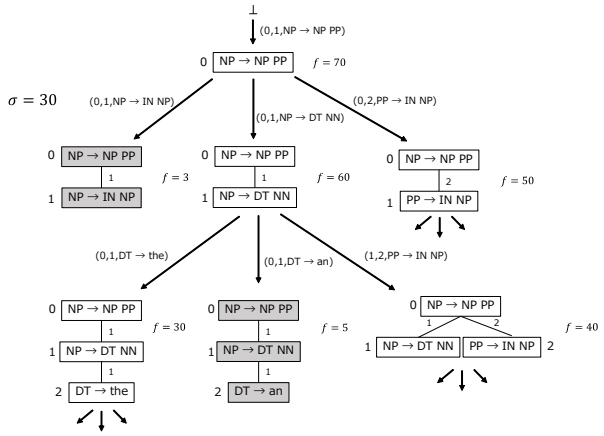

図 6: FREQT の動作例

ここで，v のラベルを $L(v)$ とし，v の子を左から順に $c_1, c_2, \ldots, c_n$ とする．

図 5 は構文木の導出過程木への変換例を示している．V が V' を子に持つとき，文法規則 $L(V')$ が $L(V)$ の右辺の要素の i 番目に適用されることを明示するために，エッジにラベルとして i を付与する．

パターンの最右葉の p 代目の親の節点 V_p に，文法規則 r をラベルに持つ節点を追加し，節点間を結ぶエッジのラベルを i とする最右拡張を以下では (p, i, r) 拡張と呼ぶ．これは文法規則 $L(V_p)$ の右辺の i 番目の要素に文法規則 r を適用することに相当する．図 6 に FREQT の導出過程木での動作例を示す．閾値は出現回数 30 回に設定している．f はそのパターンの出現回数を表す．

導出過程木のパターンは，構文木の完全パターンに対応する．コーパス中の構文木を導出過程木で表現した集合 D に FREQT を適用し，$F(D)$ を求める．次に，$F(D)$ 内の導出過程木パターンを構文木パターンに変換することにより，$F(T)$ を求める．

3.5 変換元候補の抽出

FREQT は，効率的に頻出パターンを列挙するために，頻出パターンに対してのみ拡張を行う．提案手法では，FREQT を拡張し，変換元候補となる低頻度のパターンを列挙する．本アルゴリズムでは，拡張を続ければいずれかの頻出パターンへの変換元になる可能性のあるパターンに対してのみ、拡張を行う．これにより，各 $\tau \in F(T)$ に対する $Source(\tau)$ を求めることができる．

3.5.1 変換元候補への拡張可能性の判定

本節では，あるパターンがどんな条件を満たせば，拡張を続けるといずれかの頻出パターンの変換元になる可能性があるのかを述べる．導出過程木パターンに対して最右拡張を繰り返し適用するにつれて，それに対応する構文木パターンの葉は左から順に確定していく．確定した部分が頻出構文木パターンの葉のラベルの系列のプレフィックスでない場合，この導出過程木パターンにどんな拡張を行っても，誤り訂正ルールの変換元とはなり得ない．導出過程木パターンの各節点について，最右拡張で文法規則を適用することができなくなった文法規則の右辺の要素は，今後どんな拡張を行おうとも，対応する構文木パターンの葉として出現する．このようなラベルを確定葉と呼び，確定葉を左から順に並べたものを確定葉列と呼ぶこととする．δ の確定葉列を $DY(\delta)$ と書く．δ に (p, i, r) 拡張を行い，作成された導出過程木パターンを δ_e とする．$DY(\delta_e)$ を求めるアルゴリズム *decided-yield*(δ, p, i) を以下に示す．なお，アルゴリズム中では，節点 V のラベルの左辺の要素を $L(V).LHS$，右辺の n 番目の要素を $L(V).RHS(n)$，右辺の要素の個数を $L(V).RHS.length$ とし，V の親節点を $V.parent$ とする．V とその最右の子を繋ぐエッジのラベルを $V.RME$ とする．V が子を持たない場合は $V.RME = 0$ とする．また，リスト A, B に対して，$A + B$ はリスト A の末尾にリスト B を連結させたリストを表す．

Algorithm 1 *decided-yield*(δ, p, i)

```
NDY = 空リスト ε
V = δ の最右葉
while p > 0 do
  for j = V.RME + 1 to L(V).RHS.length do
    L(V).RHS(j) を NDY に追加
  end for
  V = V.parent
  p = p − 1
end while
for j = V.RME + 1 to i − 1 do
  L(V).RHS(j) を NDY に追加
end for
DY(δ) + NDY を出力
```

図 7 に *decided-yield* の動作例を示す．δ_e は δ の (1,4,NP → DT NN) 拡張で作成されるパターンである．$DY(\delta)$ は ⟨IN⟩ である．*decided-yield*$(\delta, 1, 4)$ ではこの拡張により確定葉となったラベルの列 ⟨DT, NN,

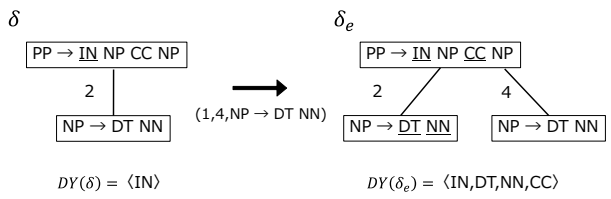

図 7: *decided-yield* の動作例

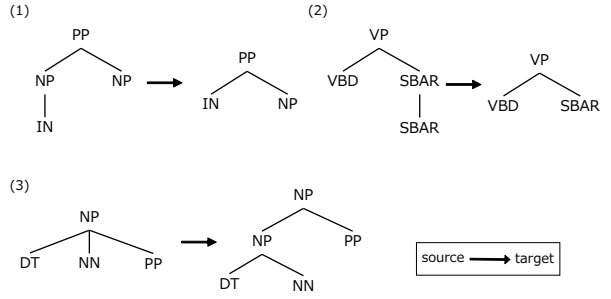

図 8: 提案手法で獲得したルールの例

CC⟩ を $DY(\delta)$ の末尾に連結した列を出力する．これにより $DY(\delta_e) = \langle$IN, DT, NN, CC$\rangle$ が求まる．

導出過程木パターン δ' に対して以下の二つの条件を満たす頻出の導出過程木パターン $\delta \in F(D)$ が存在しなければ，δ' には以後拡張を行わない．

1. $root(\delta).LHS = root(\delta').LHS$
2. $DY(\delta')$ が $yield(\delta)$ のプレフィックスである．

ここで，$yield(\delta)$ は，導出過程木パターン δ に対応する構文木パターンの葉のラベルを左から並べた列とする．

3.5.2 誤り訂正ルールの獲得

FREQT により，変換先候補の集合である $F(T)$ を求めることができ，提案手法により，各 $\tau \in F(T)$ に対して，$Source(\tau)$ を求めることができる．誤り訂正ルールはこれらより，以下のように求められる．

$$\{\langle \tau', \tau \rangle | \tau \in F(T), \tau' \in Source(\tau)\}$$

4 実験

提案手法の有効性を確認するために，構文木コーパスである Penn Treebank[4] に対して提案手法を適用した．

Kato らの手法と同様に，Penn Treebank の Wall Street Jarnal コーパスの全 49,208 文に対し，提案手法を適用することによって誤り訂正ルールを獲得した．FREQT によるコーパス中の頻出パターンの獲得において，閾値は 1,000 回とした．獲得したルールの総数は 213 個であった．そのうち 183 個が Kato らの手法で獲得できなかったルールであった．Kato らの手法で獲得できなかったルールの例を図 8 に示す．(1) は不要な NP を取り除くルールである．(2) は冗長な SBAR を取り除くルールである．(3) は DT,NN を付加する位置の誤りを訂正するルールである．これらのように，Kato らの手法では獲得できなかった構文構造の誤りを訂正するルールを，提案手法により獲得できることが確認できた．

5 おわりに

本稿では，半構造データマイニングに基づく構文木コーパスの誤り訂正手法を提案した．提案手法では，コーパス中の頻出パターンに変換可能なパターンをデータマイニングによって検出し，誤り訂正のルールを獲得する．実験により，従来の手法では獲得できなかったルールの獲得を確認し，提案手法の有効性を示した．

今後は獲得したルールでの誤り訂正の精度の評価や，閾値の変化による獲得パターン数の変動の観察を行う予定である．

参考文献

[1] Eleazar Eskin. Detecting errors within a corpus using anomaly detection. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pp. 148–153, 2000.

[2] 乾孝司, 乾健太郎. 統計的部分係り受け解析における係り受け確率の利用法: コーパス中の構文タグ誤りの検出. 情報処理学会研究報告, NL-134, pp. 15–22, 1999.

[3] Yoshihide Kato and Shigeki Matsubara. Correcting syntactic annotation errors using a synchronous tree substitution grammar. *IEICE transactions on information and systems*, E93-D, No. 9, pp. 2660–2663, 2010.

[4] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, Vol. 19, No. 2, pp. 313–330, 1993.

[5] Kenji Abe, Shinji Kawasoe, Hiroshi Sakamoto, Hiroki Arimura, and Setsuo Arikawa. Efficient substructure discovery from large semi-structured data. *IEICE TRANSACTIONS on Information and Systems*, E87-D, No. 12, pp. 2754–2763, 2004.