最長一致パターンに基づく高速・高精度な日本語形態素解析

吉永 直樹
東京大学生産技術研究所
ynaga@iis.u-tokyo.ac.jp

概要

膨大な量のテキストを解析したり，言語処理応用で大量にユーザのクエリを処理する場合，処理効率の悪いモデルは高精度でも利用し難い．本稿では高効率な手法の精度を改善すべく，最長一致パターンに基づく高精度な形態素解析手法を提案する．提案手法では，既存の辞書項目を元に，学習データから抽出したパターンを用いて形態素解析を行う．実験では，複数の品詞タグ付きコーパスと辞書を用いて提案手法の評価を行い，最小コスト法や点推定に基づく形態素解析手法の既存実装と同程度の精度，1/2 から 1/20 程度の消費メモリで，1,000,000 文/秒を超える速度の形態素解析が可能なことを確認した．本手法の実装（C++で約 1000 行）は以下で公開する．
http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/jagger

1 はじめに

Twitter や Zoom, Slack などのコミュニケーションプラットフォームの普及に従い，電子化された言語データの量は爆発的に増加し続けており，深層学習によって高精度化された機械翻訳などの実サービスは数千万を超えるユーザを獲得するようになった．このように，実世界テキストの量や実時間処理するユーザクエリがこれまでになく増加する状況下では，現実的な計算資源で運用できる効率の良い言語処理技術の重要度が高まっている．

しかしながら，深層学習の導入以降，精度に偏重して研究が行われた結果，「効率の良い」手法 [1] は他分野のように処理速度や消費メモリの限界を更新するものではなく，低速で消費メモリの大きい深層学習モデルの効率を，深層学習の枠内で相対的かつ限定的に改善するものが大半である．結果，最高精度のモデルと古典的ではあるが効率の良いモデルとの間の速度差は埋め難く，深層学習を用いた高精度の言語処理サービスは，膨大な GPU 資源を有する一部の企業のみが展開するに留まっている．

本研究では，遅いモデルの効率を改善するのではなく，効率の良いモデルの精度を改善する，真に「効率の良い」言語処理を実現すべく，機械学習を経由しない単純で高精度な日本語形態素解析手法を提案する．日本語における形態素解析は，一般に，単語分割，品詞解析，見出し語化の混合タスクとしてモデル化されるが，我々の提案手法は，古典的な最長一致法に基づく単語分割手法を拡張したもので，次の分割位置と切り出した単語の品詞・見出し語を決定するパターンに基づき形態素解析を行う．解析に用いるパターンは形態素解析辞書の項目をシードとして，学習データから頻度に基づいて抽出する．抽出したパターンはダブル配列 [2, 3] に格納し，時空間効率の良い処理を実現する．

実験では，多様なドメインの標準コーパス [4, 5] と形態素解析辞書を組み合わせて提案手法の評価を行った．その結果，提案手法が，探索 (最小コスト法) や分類 (点推定) に基づく実装 (MeCab, Vaporetto) と同程度の精度で，MeCab の約 15 倍， Vaporetto の約 10 倍 (M2 MacBook Air で 1,000,000 文/秒以上) の速度で形態素解析を行えることを確認した．

2 パターンに基づく形態素解析

本節では，単語分割，品詞タグ付け，見出し語化を同時に行う決定的な形態素解析手法を提案する．速度に妥協のない実装を得るため，本研究では古典的な最長一致法 [6] を拡張した最長一致パターンに基づく形態素解析アルゴリズムを提案する．

最長一致法は辞書 (単語の集合) に基づく決定的な単語分割手法であり，入力文字列の先頭から辞書中の単語と一致する最長の文字列を，繰り返し単語として認識することで単語分割を行う．最長一致法は，日本語や中国語の単語分割において，修正規則と組み合わせることで高精度化できること [7, 8] が知られているが，機械学習を用いた最小コスト法や点推定などの標準的な単語分割手法 [9, 10] と比べると分割精度には一定の差 [11] が存在する．

Algorithm 1 Pattern-based morphological analysis

INPUT: 入力テキスト (文字列 $\boldsymbol{c}$), trie に保存されたパターン集合 $\mathcal{P} = \{\langle \text{pattern}, \text{shift}, t\rangle\}$
OUTPUT: 単語と品詞のペアの系列 $\boldsymbol{s} = (\langle w_j, t_j\rangle)$

```
1: i ← 0
2: while i < len(c) do
3:     ⟨shift, t̂⟩ = longest_prefix_search(c_{≥i}, P)
4:     append(s, ⟨c_i^{i+shift}, t̂⟩)
5:     i ← i + shift
6: return s
```

2.1 基本的なアルゴリズム

本節では，単語分割，品詞タグ付け，見出し語化[1)]を同時に行う決定的な形態素解析アルゴリズムを提案する．Algorithm 1 は，与えられた文字列に対し，トライに格納されたパターン $\mathcal{P}$ を繰り返し適用して，後続する単語 $w = \boldsymbol{c}_i^{i+\text{shift}}$ を切り出すと同時に品詞 (と見出し語) を決定するアルゴリズムである．単語分割に対する最長一致法と同様に，本手法でも最長一致するパターンが選択・適用されるが，分割位置がパターン長と必ずしも一致しない点に注意されたい．3 節で確認するように，この単純なアルゴリズムは有効なパターン集合が用意できれば，最小コスト法や点推定などの探索・分類ベースの手法に匹敵する精度で動作する．

本手法は，系列ラベリングや組合わせ特徴に基づく分類器における事前計算 [9, 12, 13, 14] に着想を得ている．これらの効率的な実装では，事前計算した特徴量の重みを，表層など単純な特徴の系列をキーとして取得することで推論を行う．機械学習で用いる特徴量が単純なキー (パターン) に畳み込めるのであれば，直接，分類結果 (単語分割の有無，品詞，見出し語) に写像可能なパターン（特徴列）が存在しうるというのが本手法の基本的なアイデアである．

2.2 辞書に基づく特徴列パターンの抽出

最小コスト法 [9] や点推定 [10] を実装した形態素解析器の素性テンプレートを参考に，本稿では提案手法のためのパターンテンプレートとして，分割位置以降の表層文字列 $\boldsymbol{c}_p$ と直前の単語の品詞 t_{j-1} の連結，すなわち $\boldsymbol{c}_p; t_{j-1}$(; は文字の連結) を採用した．

Algorithm 2 は，形態素解析の学習データから特徴列パターンをマイニングするアルゴリズムである．単語分割，品詞タグ及び見出し語が付与された学習データ $\mathcal{D}$ と単語がとりうる品詞タグ・見出し語を定義した形態素解析辞書 $\mathcal{V}$ を入力として，本手法は逐次的に特徴列パターンを学習データから抽出する．

Algorithm 2 Extracting patterns from training data

INPUT: 学習データ $\mathcal{D} = \{\langle \boldsymbol{c}_i, \boldsymbol{s}_i\rangle\}$, 辞書 (単語と品詞候補) $\mathcal{V}$, 最大パターン長 L (= max(len($\boldsymbol{w}$)); $\boldsymbol{w} \in \mathcal{V}$)
OUTPUT: パターン集合 $\mathcal{P} = \{\langle p, \text{shift}, t\rangle\}$

```
1: P̂ ← φ (パターンに対する分割位置・品詞タグ候補)
2: for all training example ⟨c, s = (⟨w_j, t_j⟩)⟩ ⊂ D do
3:     i ← 0
4:     for j = 0 to len(s) do
5:         shift = len(w_j)
6:         for k = shift to L do
7:             P̂[c_i^{i+k} : t_{j-1}][⟨shift, t_j⟩] += 1
8:         i ← i + shift
9: P ← {⟨w, len(w), t⟩}
10:        where (w, t) ⊂ V, t = argmax_{t'} P̂[w][⟨len(w), t'⟩]
11: for all p̂ ⊂ P̂ from the shortest one do
12:     shift = argmax'_{shift} Σ_{t'} P̂[p̂][⟨shift', t'⟩]
13:     t = argmax_{t'} P̂[p̂][⟨shift, t'⟩]
14:     ⟨shift', t'⟩ = longest_match_search(p̂, P)
15:     if ⟨shift, t⟩ ≠ ⟨shift', t'⟩ then
16:         P ← P ∪ {⟨p̂, shift, t⟩}
17: return P
```

具体的にはまず，学習データ中の全ての単語分割位置に対し，その位置から次の分割位置以降までを含む文字列 $\boldsymbol{c}_i^{i+k}$ を表層パターンとして抽出し，直前の単語の品詞タグ t_{j-1} と組み合わせてパターン候補を得る (4-8 行目)．これらのパターン候補に対して，分割位置 (shift) ・品詞タグ (・見出し語，以後省略) t の頻度を数えて，最多の分割位置・品詞タグをそのパターンが適用されたときの分割位置・品詞タグとして採用する (12-13 行目)．冗長なパターンを枝刈りするために，接頭文字列が同じで分割位置・品詞タグが同じパターンについては，最も短いパターンのみを採用する (14-16 行目)．提案手法は最長一致法と比べて，後続文脈や前文脈品詞について追加の走査が必要となるが，大きなオーバーヘッドとならないことを確認している．[2)]

3 実験

本節では提案手法を多様なドメインの形態素情報付きコーパス [4, 5] を用いて評価し，探索や分類に基づく効率的な形態素解析手法 [9, 10] と速度，消費メモリ，精度の観点で比較する．

1) 見出し語化については，一部比較手法が対応しないため，本稿では評価の対象としない．

2) Aho–Corasick ライクな手法で表層について戻り読みのない処理を行うことも可能だが，逆に処理速度が低下した．

表 1　評価データの統計.

	KYOTO			KWDLC		
	train	dev	test	train	dev	test
文数	35,478	1145	1783	12,271	1585	2195
語/文	25.37	26.24	25.83	15.85	14.27	16.34

表 2　形態素解析辞書の統計.

	jumandic-5.1	jumandic-7.0
収録語数	475,716	702,358
品詞タグ (level 1)	14	14
(level 1-4)	980	1188

3.1　設定

データセット　実験では新聞記事に注釈を付与した京都大学テキストコーパス[3] (KYOTO) [4] とウェブページの冒頭三文に注釈を付与した京都大学ウェブ文書リードコーパス[4] (KWDLC) [5] を用いた．開発・評価データとしては，github リポジトリに含まれる分割を用い，残りを学習データに用いた (表 1).

比較モデル　本稿では，以下の実装を比較する．学習ベースのモデルではハイパーパラメタ c[5] を開発データを用いて調整し，活用を含む品詞タグ (level 1-4) の F_1 が最大となるモデルを評価に用いた．

Jagger は提案手法の C++実装であり，入力テキストの先頭から後続する表層文字列と前文脈品詞に対して特徴列パターンを繰り返し適用し，決定的に単語分割，品詞タグ付け，見出し語化を行う．これらのパターンは学習データと辞書から抽出される．

MeCab[6] は最小コスト法の C++実装[7] であり，条件付き確率場 [15] により，学習データからパラメタを推定する [9]. 最小コスト法では，辞書に基づき可能な分割と品詞の組み合わせをラティスで表現して，最尤となる分割・品詞を動的計画法で求める．

Vaporetto は点推定 [10] の Rust 実装[9] である．点推定では，各文字間を分割候補とする二値分類により単語分割を行った後，各単語に対して多クラス分類により品詞タグ付けを行う．公平な比較のために，単語特徴量には他の手法と同じ形態素解析辞書を用いる．Vaporetto では単語の品詞候補ごとに分類器を学習しており，学習データに出現しない単語[10] には，辞書中で最初に現れる品詞タグが付与される [16].

3)　https://github.com/ku-nlp/KyotoCorpus
4)　https://github.com/ku-nlp/KWDLC
5)　$c = \{0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}$ を試した.
6)　https://taku910.github.io/mecab/
7)　MeCab の高速な再実装である Vibrato[8] は，今回用いた実験設定で品詞タグ付けまで行うと，MeCab より遅くメモリ消費も大きかったため，比較しなかった.
8)　https://github.com/daac-tools/vibrato
9)　https://github.com/daac-tools/Vaporetto
10)　学習データにも辞書にも出現しない単語には品詞タグが付与されないため，文献 [9] を参考にサ変名詞を割り当てた．なお，この処理にかかる時間は処理速度には含めない.

なお，Rust と C++が生成するコードの実行速度は，多くのベンチマークで同程度[11] である．

提案手法は，大域的な最適化を行う最小コスト法よりは，局所的な最適化を行う点推定に近い手法であるが，二つの点で異なる．まず，提案手法は全ての文字間に対して分割の有無を判定するのでなく，パターンに基づき，次の分割位置を決定する (分類問題としては次の分割位置を予測する多クラス分類に相当)．また，提案手法では，単語分割，品詞タグ付け (，見出し語化) を同時かつ決定的に行う．

形態素解析辞書　各手法が用いる辞書としては，形態素解析器 JUMAN[12] の辞書を MeCab 用に変換したものを用いる．具体的には mecab-jumandic-5.1-20070304 と mecab-jumandic-7.0-20130310 を，辞書の質とカバレッジの影響を分析するために比較する．このうち，jumandic-7.0 ではウェブから自動獲得した語彙 [17] が含まれており，収録語数が 475,716 語 (jumandic-5.1) から 702,358 語 (jumandic-7.0) に増加している．品詞タグは 4 階層 (2 段階の品詞分類，活用型，活用形) からなる形態・統語論的なカテゴリが付与されている (表 2).

評価方法　解析精度に関する評価尺度としては，既存研究 [9] に倣い，単語分割と品詞タグに関する精度，再現率，F_1 値を用いる．処理効率の評価としては，評価データを 1000 回コピーしたデータの解析時間 (秒)，解析速度 (文/秒)，最大消費メモリ (MiB) を`/usr/bin/time -l` コマンドにより 3 回計測して，中央値となる処理時間 (速度) と，そのときの最大消費メモリを報告する．実験は 3.5Ghz CPU と主記憶 24 GB を備えた M2 MacBook Air 上で行った．なお，付録 C，D に単語分割のみの評価，及び深層学習に基づく形態素解析器との比較を載せた．

3.2　結果

表 3, 4 に KYOTO, KWDLC コーパスでの単語分割・品詞タグ付けの評価結果を示す．我々の提案手法の参照実装である Jagger が，最小コスト法を実装した MeCab や点推定を実装した Vaporetto と遜色のない

11)　https://github.com/kostya/benchmarks/
12)　https://nlp.ist.i.kyoto-u.ac.jp/?JUMAN

表 3 KYOTO コーパスの解析結果, F_1 (precision/recall).

KYOTO	time [秒] ↓	speed [文/秒] ↑	space [MiB] ↓	seg	top (level 1)	all (level 1-4)
			w/ jumandic-5.1			
MeCab	26.83	66,455	55.81	98.68 (98.47/98.89)	97.32 (97.12/97.53)	95.97 (95.76/96.17)
Vaporetto	18.23	97,805	658.70	**98.94** (**98.97**/**98.92**)	**98.30** (**98.32**/**98.27**)	**96.92** (**96.95**/**96.90**)
Jagger (ours)	**1.77**	**1,007,344**	**26.39**	98.73 (98.62/98.83)	97.62 (97.52/97.72)	96.55 (96.45/96.65)
			w/ jumandic-7.0			
MeCab	29.99	59,453	77.98	98.37 (98.02/98.72)	97.19 (96.84/97.54)	96.10 (95.75/96.44)
Vaporetto	20.03	89,016	957.72	**99.08** (**99.08**/**99.08**)	**98.42** (**98.42**/**98.43**)	**97.05** (**97.04**/**97.05**)
Jagger (ours)	**1.83**	**974,316**	**35.09**	98.68 (98.51/98.86)	97.63 (97.46/97.80)	96.57 (96.74/96.40)

表 4 KWDLC コーパスの解析結果, F_1 (precision/recall).

KWDLC	time [秒] ↓	speed [文/秒] ↑	space [MiB] ↓	seg	top (level 1)	all (level 1-4)
			w/ jumandic-5.1			
MeCab	23.83	92,110	53.88	97.13 (96.82/**97.44**)	95.62 (95.32/95.93)	**94.30** (94.00/**94.60**)
Vaporetto	13.90	157,913	632.31	**97.35** (**97.39**/97.32)	**96.16** (**96.20**/**96.13**)	94.08 (**94.11**/94.04)
Jagger (ours)	**1.44**	**1,524,305**	**28.89**	97.17 (96.94/97.40)	95.71 (95.49/95.94)	94.20 (93.98/94.42)
			w/ jumandic-7.0			
MeCab	26.90	81,598	76.38	**97.99** (**97.82**/**98.16**)	**96.66** (**96.49**/**96.83**)	**95.62** (**95.45**/**95.78**)
Vaporetto	15.47	141,887	844.86	97.53 (97.58/97.49)	96.39 (96.43/96.34)	94.68 (94.72/94.63)
Jagger (ours)	**1.46**	**1,503,424**	**40.22**	97.60 (97.49/97.71)	96.14 (96.04/96.25)	94.63 (94.52/94.73)

精度を示すと共に，MeCab の約 15 倍，Vaporetto の約 10 倍高速，かつ MeCab の約 1/2，Vaporetto の約 1/20 程度の消費メモリで解析が行えている．Jagger が用いる特徴列パターンは枝刈り (Algorithm 2 の 14 行目) によって最小限に抑えられており，機械学習を経由せず，浮動小数点数で表現されるパラメタも含まないことから，高効率となったと考えられる (他手法の学習に相当するパターンの抽出時間は 6 秒未満であった)．MeCab の解析精度は辞書の性質に依存し，jumandic-7.0 を用いた場合，KWDLC で優れた精度を示す一方で KYOTO では単語分割精度が低下している．Vaporetto は，学習データの多い KYOTO では MeCab や Jagger より高精度だが，学習データの少ない KWDLC では Jagger と同程度の精度に留まる．

表 5 に，学習データとは異なるドメインの評価データを用いて各手法を評価したときの解析精度を示す．辞書に基づいて単語分割候補を列挙する最小コスト法 (MeCab) に対し，辞書を間接的に用いる点推定 (Vaporetto) の解析精度が低く，学習ドメインに依存する点推定のリスクを示している．提案手法の実装である Jagger は MeCab よりは性能が低下しているものの，Vaporetto ほどには性能が落ちておらず，辞書と学習データの両方をバランスよく活用した手法となっている．提案手法は，辞書，あるいはパターンを追加する形で学習という余分なプロセスを経ずに処理を即座に変更することもできる．

表 5 クロスドメイン評価 (辞書: jumandic-7.0), F_1 .

	seg	top (level 1)	all (level 1-4)
学習: KWDLC→ 評価: KYOTO			
MeCab	**97.90**	**96.56**	**94.82**
Vaporetto	95.76	93.81	91.31
Jagger (ours)	97.25	95.42	93.30
学習: KYOTO→ 評価: KWDLC			
MeCab	**97.78**	**96.02**	**94.48**
Vaporetto	97.05	95.15	92.72
Jagger (ours)	97.22	95.01	93.12

4 まとめ

本稿では最速の言語処理技術を高精度化すべく，最長一致パターンに基づく高精度な形態素解析器を提案した．我々の手法は，辞書中の単語をシードとして学習データから単語分割位置と品詞タグ (及び見出し語) を決定するパターンを抽出し，最長一致パターンを順次切り出す形で効率よく高精度の解析を行う．複数の品詞タグ付きコーパスを用いた実験により，提案手法が最小コスト法や点推定の効率的な既存実装に匹敵する解析精度を達成し，同時に，ノート PC で 1,000,000 文/秒という基礎解析として望ましい解析速度を実現することを確認した．

今後は，深層学習により得られる埋め込み表現を離散化してパターンに組み込むなどして，本論文のアプローチを他の言語処理タスクに応用したい．

謝辞

本研究は JSPS 科研費 JP21H03494 の助成を受けたものです．Vaporetto の未知語処理に関する実装をご教示くださった LegalOn Technologies の赤部晃一氏，本稿の草稿にコメントをくださった Rakuten Institute of Technology Americas の新里圭司氏に感謝いたします．

参考文献

[1] Marcos Treviso, Tianchu Ji, Ji-Ung Lee, Betty van Aken, Qingqing Cao, Manuel R. Ciosici, Michael Hassid, Kenneth Heafield, Sara Hooker, Pedro H. Martins, André F. T. Martins, Peter Milder, Colin Raffel, Edwin Simpson, Noam Slonim, Niranjan Balasubramanian, Leon Derczynski, and Roy Schwartz. Efficient methods for natural language processing: A survey, 2022.

[2] Jun'ichi Aoe. An efficient digital search algorithm by using a double-array structure. **IEEE Transactions on Software Engineering**, Vol. 15, No. 9, pp. 1066–1077, 1989.

[3] Naoki Yoshinaga and Masaru Kitsuregawa. A self-adaptive classifier for efficient text-stream processing. In **Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers**, pp. 1091–1102, Dublin, Ireland, August 2014.

[4] Sadao Kurohashi and Makoto Nagao. Building a Japanese parsed corpus. In **Treebanks: Building and Using Parsed Corpora**, pp. 249–260. Kluwer Academic Publishers, 2003.

[5] Masatsugu Hangyo, Daisuke Kawahara, and Sadao Kurohashi. Building a diverse document leads corpus annotated with semantic relations. In **Proceedings of the 26th Pacific Asia Conference on Language, Information, and Computation**, pp. 535–544, Bali, Indonesia, November 2012.

[6] Masaaki Nagata. A stochastic Japanese morphological analyzer using a forward-DP backward-A* n-best search algorithm. In **COLING 1994 Volume 1: The 15th International Conference on Computational Linguistics**, Kyoto, Japan, August 1994.

[7] David D. Palmer. A trainable rule-based algorithm for word segmentation. In **35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics**, pp. 321–328, Madrid, Spain, July 1997.

[8] Julia Hockenmaier and Chris Brew. Error-driven learning of Chinese word segmentation. In **Proceedings of the 12th Pacific Asia Conference on Language, Information and Computation**, pp. 218–229, Singapore, February 1998.

[9] Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. Applying conditional random fields to Japanese morphological analysis. In **Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing**, pp. 230–237, Barcelona, Spain, July 2004.

[10] Graham Neubig, Yosuke Nakata, and Shinsuke Mori. Pointwise prediction for robust, adaptable Japanese morphological analysis. In **Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies**, pp. 529–533, Portland, Oregon, USA, June 2011.

[11] Manabu Sassano. Deterministic word segmentation using maximum matching with fully lexicalized rules. In **Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers**, pp. 79–83, Gothenburg, Sweden, April 2014.

[12] Nobuhiro Kaji, Yasuhiro Fujiwara, Naoki Yoshinaga, and Masaru Kitsuregawa. Efficient staggered decoding for sequence labeling. In **Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics**, pp. 485–494, Uppsala, Sweden, July 2010.

[13] Naoki Yoshinaga and Masaru Kitsuregawa. Kernel slicing: Scalable online training with conjunctive features. In **Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)**, pp. 1245–1253, Beijing, China, August 2010.

[14] 赤部晃一森信介. Vaporetto: 点予測法に基づく高速な日本語トークナイザ. 言語処理学会 第 28 回年次大会 発表論文集, pp. 256–261, 2022.

[15] Percy Liang, Hal Daumé, and Dan Klein. Structure compilation: Trading structure for features. In **Proceedings of the 25th International Conference on Machine Learning**, ICML '08, p. 592–599, New York, NY, USA, 2008.

[16] 森信介, 中田陽介, Neubig Graham, 河原達也. 点予測による形態素解析. 自然言語処理, Vol. 18, No. 4, pp. 367–381, 2011.

[17] Yugo Murawaki and Sadao Kurohashi. Online acquisition of Japanese unknown morphemes using morphological constraints. In **Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing**, pp. 429–437, Honolulu, Hawaii, October 2008.

[18] 工藤拓. 形態素解析の理論と実装. 近代科学社, 2018.

[19] Huidan Liu, Minghua Nuo, Longlong Ma, Jian Wu, and Yeping He. Compression methods by code mapping and code dividing for Chinese dictionary stored in a double-array trie. In **Proceedings of 5th International Joint Conference on Natural Language Processing**, pp. 1189–1197, Chiang Mai, Thailand, November 2011.

[20] Arseny Tolmachev, Daisuke Kawahara, and Sadao Kurohashi. Juman++: A morphological analysis toolkit for scriptio continua. In **Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations**, pp. 54–59, Brussels, Belgium, November 2018.

[21] Hajime Morita, Daisuke Kawahara, and Sadao Kurohashi. Morphological analysis for unsegmented languages using recurrent neural network language model. In **Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing**, pp. 2292–2297, Lisbon, Portugal, September 2015.

表 6　JUMAN++との比較, F_1 (precision/recall).

	time [秒]↓	speed [文/秒]↑	space [MiB]↓	seg	top (level 1)	all (level 1-4)
			KYOTO			
JUMAN++-V2	331.14	5384	300.80	**99.37 (99.30/99.45)**	**98.72 (98.65/98.80)**	**97.74 (97.66/97.82)**
Jagger (ours)	**1.83**	**974,316**	**35.09**	98.68 (98.51/98.86)	97.63 (97.46/97.80)	96.57 (96.74/96.40)
			KWDLC			
JUMAN++-V2	283.11	7753	290.05	**98.37 (98.25/98.50)**	**97.61 (97.49/97.73)**	**96.42 (96.30/96.55)**
Jagger (ours)	**1.46**	**1,503,424**	**40.22**	97.60 (97.49/97.71)	96.14 (96.04/96.25)	94.63 (94.52/94.73)

A　未知語処理

最小コスト法と同様に，提案手法でも辞書・学習データに含まれない未知語の扱いは，問題となる．現在の実装では，既存手法 [9] と同様に，文字種に基づいて，数字，アルファベット，カタカナを連結する単純な未知語処理を実装している．具体的には，連続する数字，アルファベットはそれぞれ単純に連結し，連続するカタカナ語については，単語の合計長が一定サイズ以下のときに連結する．いずれの場合も，前文脈品詞，および連結する末尾の語に対するパターンに基づき品詞を割り当てる．

B　実装上の工夫

Jagger の C++実装には，MeCab で用いられているゼロコピーや mmap [18]，J.DepP[13] でも用いられているキャッシュ効率化のための頻度順 ID[19] に加えて，Vaporetto や Vibrato で用いられている文字単位で遷移するダブル配列を利用している．文字単位のダブル配列には，バイト単位で遷移する動的ダブル配列 cedar[14] に軽微な修正を加えたものを用いた．

C　単語分割のみの処理効率の比較

品詞タグ付けに対するアプローチ（未知語処理，見出し語化への対応）の違いから，MeCab / Jagger と Vaporetto を形態素解析器として厳密な意味で公平に比較することは困難である．そこで，各実装で jumandic-7.0 を用いて，単語分割のみを行い，処理速度・消費メモリを比較した．なお，MeCab, Jagger については表 3, 4 と同じモデルを用いた．Vaporetto については，省メモリ化のため単語分割のみの学習データから，単語リストとして辞書を与えて，単語分割のみを行うモデルを再学習した[15]．

13）http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/jdepp/
14）http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/cedar/
15）提案手法でも，単語分割のみが付与されたコーパスから単語分割のみを行うモデルを学習可能であり，前文脈品詞をパターンで考慮するモデルと分割精度はほぼ同じであった．

表 7　単語分割の処理効率 (辞書: jumandic-7.0).

	time [秒]↓	speed [文/秒]↑	space [MiB]↓
	KYOTO		
MeCab	28.53	62,495	40.52
Vaporetto	6.90	258,405	280.58
Jagger (ours)	**1.41**	**1,264,539**	**21.05**
UTF-8 split (参考)	0.31	5,751,612	-
	KWDLC		
MeCab	25.70	85,408	39.59
VAporreto	6.30	348,412	275.55
Jagger (ours)	**1.13**	**1,942,477**	**20.16**
UTF-8 split (参考)	0.26	8,442,307	-

表 7 に，3.1 節の評価方法に倣って計測した単語分割の処理効率を示す．表で UTF-8 split は，UTF-8 で符号化された評価データを文字ごとに分割して出力する処理である．Vaporetto の処理効率が改善しているが，(モデルの読み込みのオーバーヘッド (約 1.7 秒) 点を考慮しても) 提案手法による処理の方が高速かつ省メモリである．

D　ニューラル形態素解析との比較

現時点で最高精度の日本語形態素解析器である JUMAN++-V2 [20][16] との比較を行った．モデルは，表 1 と同じ学習データから公式スクリプト[17] 及び，学習コーパスに対し最適化されたハイパーパラメタを用いて学習した．なお，JUMAN++は，Wikipedia から抽出した辞書と大規模コーパスから訓練された RNN 言語モデルを追加で用いており，精度の観点では Jagger に不利で，速度の観点では JUMAN++に不利な比較となっている点に注意されたい．

表 6 に jumandic-7.0 を辞書に用いた Jagger と比較した結果を示す．Jagger と JUMAN++の解析精度の差は，品詞タグ付けでは大きいが単語分割では 1%以内に収まっている．JUMAN++-V2 は JUMAN++ [21] に対して 250 倍の高速化を達成したと報告されているが，Jagger はその JUMAN++-V2 よりさらに 180 倍以上高速で，1/7 以下の消費メモリであった．

16）https://github.com/ku-nlp/jumanpp
17）https://github.com/ku-nlp/jumanpp-jumandic