特徴列パターンに基づく超高速・高精度な形態素解析

吉永 直樹
東京大学生産技術研究所

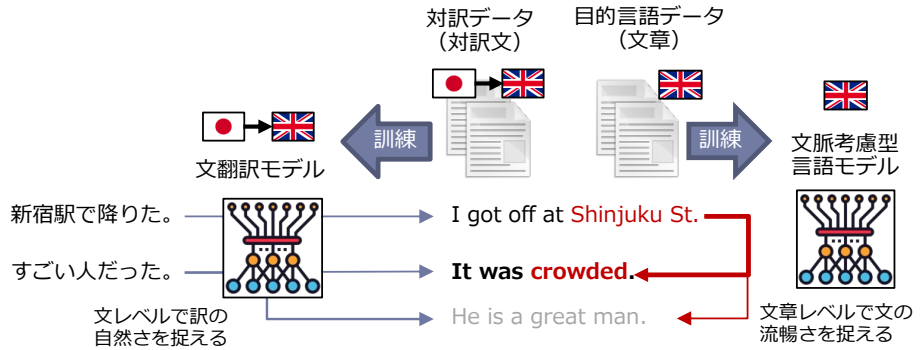
このスライドは、NLP2023 で公表した論文の発表スライドを増補・改訂したものです

自己紹介

- **名前: 吉永直樹**
  - 九州生まれ、関西育ち (滋賀, 京都; 1978 - 1996)
  - 東京大学大学院 情報理工学系研究科
    コンピュータ科学専攻 博士課程修了 (2005)
  - JSPS 特別研究員 (DC1, PD@JAIST, - 2008)
  - **東京大学生産技術研究所** (2008 -, 現在、 **准教授**)

研究室における取り組み (1/2)
実世界志向の「語用論的」言語処理応用

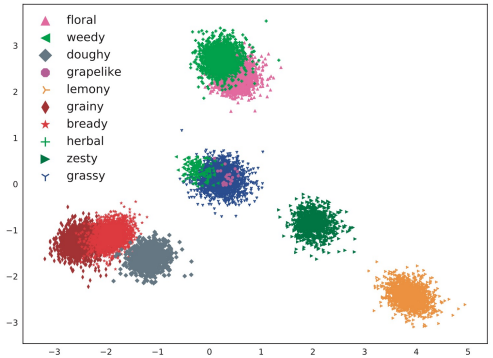
周辺文脈を考慮した言語処理 [NAACL-21]



文翻訳モデルで文脈を考慮した翻訳を実現

言語話者を考慮した言語処理 [NAACL-19]

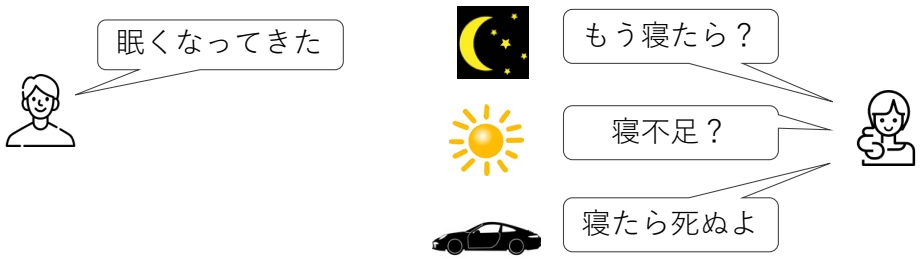
ビールのレビューにおける
個人ごとの形容詞の意味



個人が用いる単語の意味の揺らぎを指摘

時間情報を考慮した言語処理 [ACL-17 SRW]

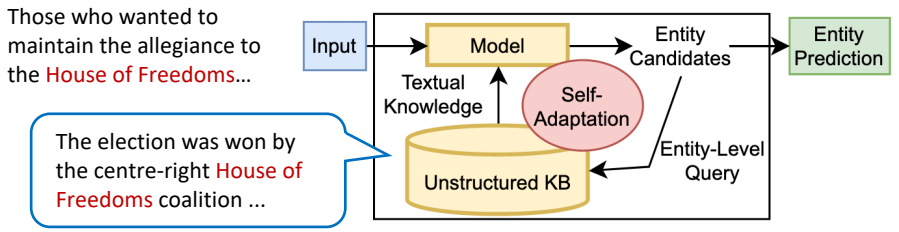
- 時間情報を追加入力して雑談応答を生成



対話における時間情報の重要性を示唆

学習済み深層学習モデルの動的適応

- 語彙・世界知識の補完・置換 [EMNLP-20/22 Findings]
- 不足する知識の自律的検索 [ACL-22, **EACL-23**]



学習外ドメインに頑健な深層学習モデル

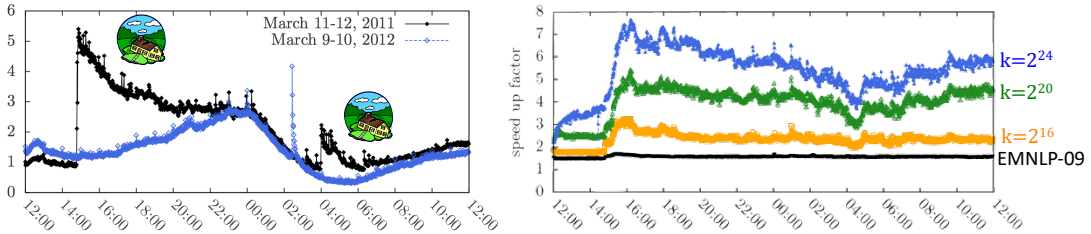
学習＝評価ドメインを仮定する文単位ベンチマーク言語処理からの脱却

研究室における取り組み (2/2)
膨大・多様な実世界言語データのための基礎解析基盤

超高速な基礎言語解析

- SNS **投稿の量の変化に応じ加速**する分類器
  - **J.DepP:** CaboCha (業界標準) より高精度で
    18倍高速な構文解析器 (5万文/秒)

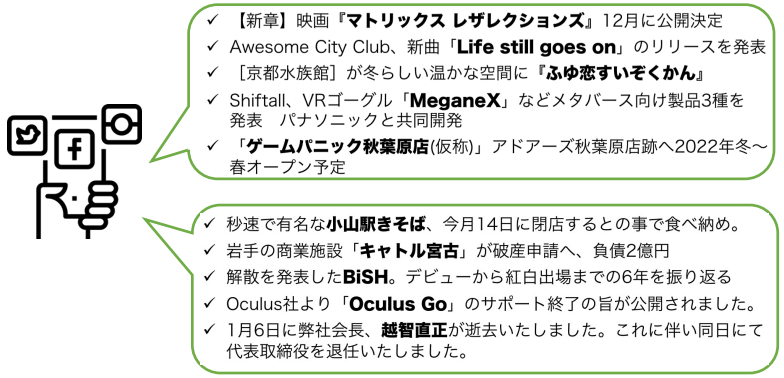
3.11震災ツイートの構文解析(分類)の高速化率



>500万投稿/秒の Twitter 解析基盤を構築

実世界変化のリアルタイム把握

- **新事物（エンティティ）の出現・消滅**を
  Twitter から即時的に発見・分類



Wikipedia登録の1年以上前に事物を検出

本日の講演

- 特徴列パターンに基づく**決定的言語解析**
  - **Jagger**: MeCab (業界標準) と同程度の精度で
    16倍高速な形態素解析器 (100万文/秒)

パターンに基づく最速言語処理を高精度化

情報処理学会全国大会大会優秀賞　人工知能学会研究会優秀賞　言語処理学会全国大会若手奨励賞　言語処理学会全国大会委員特別賞

トップ国際会議 COLING-14, IJCAI-17, EMNLP-21 Findings 採択

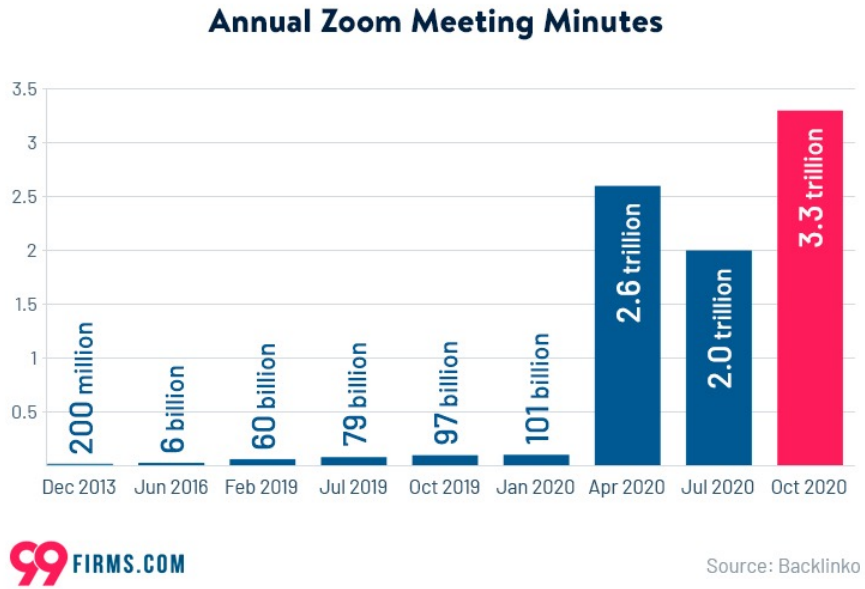
NICT DISAANA 対災害SNS情報分析システム

WISDOM X

2023/04/13 **解析基盤の共用・OSS化により、様々な企業・研究機関で社会分析に活用**

言語データの増加・多様化

- 個人の思想や体験を含む**ソーシャルビッグデータ**の出現
- コロナ禍における**オンラインコミュニケーション**の増加



https://99firms.com/blog/zoom-statistics/

超大規模言語データに基づく社会分析への期待

超大規模言語データ分析のための基礎解析

- **精度よりもまず速度**
  - 最速の手法＋微調整（辞書・少量の学習データ等）が現実的
  - 10倍高速 >> 1%精度が高い (sota)

- **既存の基礎解析手法で十分か？**
  - 解析手法はベンチマークデータでの in-domain 精度に焦点
  - 「効率の良い」手法は遅いモデルを少し速くする研究が多い

高速な日本語基礎解析は MeCab [Kudo+ 2004]**, J.DepP** [Yoshinaga+ 2009] **以降, 単語分割** [Sassano 2014] **など一部の処理を除き（昨年まで）停滞**

余談: 言語処理研究の方向性

- 前提: 人の言語能力と比べると、計算機による言語処理は精度が低く（短所）、速度は速い（長所）

- **精度を改善する**（＝短所を減らす）

論文で公表される学術的成果の大半

  - **公平な比較が容易**（学習・評価データを揃える必要あり）
  - 近年では、基盤モデルを通して**外部データを暗黙的に利用**

- **速度をさらに速くする**（＝長所を伸ばす）
  - **公平な比較が困難**（計算量を改善 or 劇的な効率化が必要）
  - 応用（翻訳など）では人が出力を読むより速ければ十分？
  - データが多ければサーバを増やして処理を並列化すればよい？

（個人的には）最高精度かつ最速の言語処理技術が目標

本研究の概要

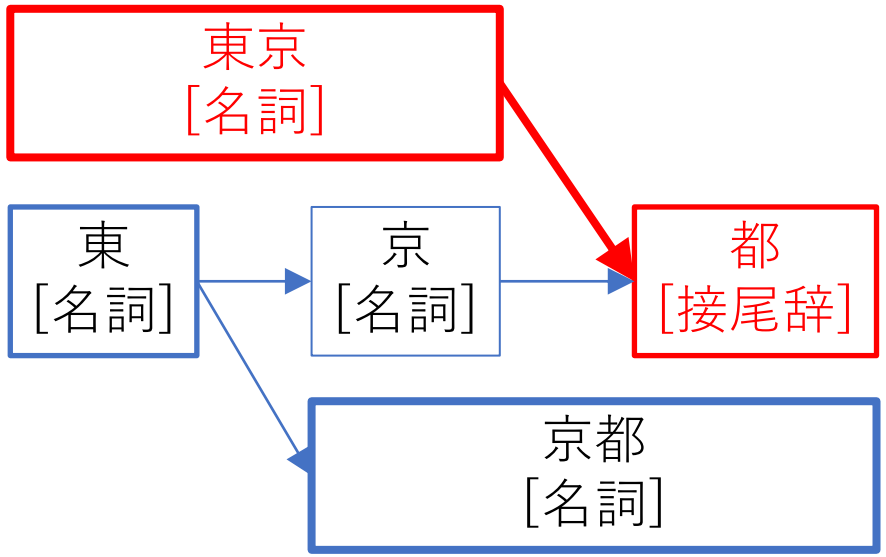
- **アプローチ**
  - 高精度で遅い手法を高速化するのでなく**速い手法を高精度化**

- **提案**: パターンに基づく**高速かつ高精度の形態素解析**
  - 単語分割における**最長一致法を拡張し、形態素解析に適用**

- **結果**
  - MeCab (Vibrato), Vaporetto と同程度の精度で 7-16x 高速
  - 1CPUで新聞記事100万文/秒、**ウェブ文書150万文/秒**で解析可

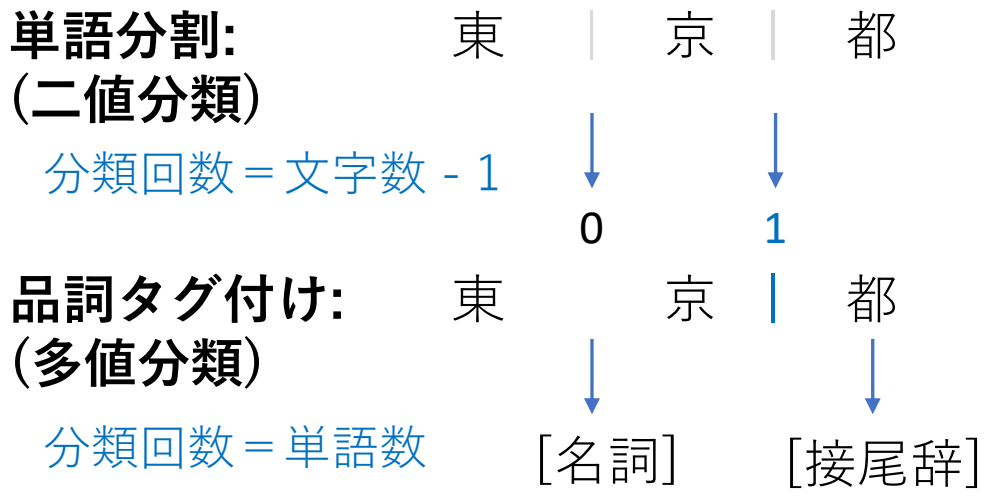
関連研究 – 標準的な形態素解析手法

- **最小コスト法** [Kudo+ 2004]
  - 連接・生起コストを動的計画法を用いて最小化し、形態素解析
  - 効率の良い実装: **MeCab**, Vibrato



決定的な解析手法と比べると遅い
辞書のカバレッジに精度が依存

- **点推定** [Neubig+ 2011]
  - 単語分割・品詞タグ付けを個別の分類問題として独立に解く
  - 効率の良い実装: Vaporetto



単語分割は速いが品詞タグ付けは
ラベル数が多く分類コスト大

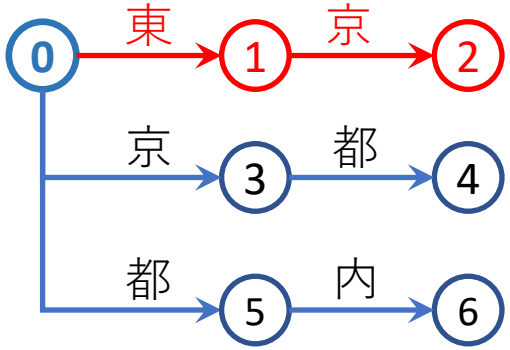
関連研究 – 最長一致法に基づく単語分割

- **最長一致法: 辞書（＝単語リスト）を利用した単語分割**
  - 直前の分割位置から始まる最長の単語の終端で**決定的に分割**
  - 修正規則を用いて MeCab 比 4x速を維持して精度改善 [Sassano 2014]

解析済み　**共通接頭辞検索**

. . . | 東　京 | 都

辞書：{東, 東京, 京, 京都, 都, 都内}



- **課題**
  - 最小コスト法や点推定との精度差は、依然大きい (>1-2%)
  - **単語分割のための経験則**であり、品詞タグ付けに適用困難

本研究の目標: 最長一致法に着想を得た高精度な形態素解析

提案手法 – 基本的なアイデア

- 形態素解析＝次の単語分割位置と切り出した単語の品詞を同時に分類する多クラス分類とみなす

  - 分類回数は点推定の品詞タグ付けと同じ（単語数）
  - 特徴量に後文脈表層、前文脈品詞を使うことで、**高精度化**

- 分類問題として素朴に解くと、分類ラベル数が多く遅いため**特徴量を並べてパターンにし、分類結果を直接取得**

提案手法 – 分類からパターンマッチへ

- 後文脈表層が最長一致するパターンを適用、分割位置と品詞を決定（**最長一致＝より多くの特徴を用いた分類**）
  - **特徴列パターン**＝後文脈表層文字列＋（解析済）前文脈品詞

- どのようにパターンを得るか？
  - 案1: 分類器を学習した後、実データに出現する**特徴列を列挙しスコアを事前計算** [Yoshinaga+ 2009, Morita+ 2019, 赤部+ 2021]
  - 案2: **特徴列を学習データからマイニング**し、実際の分割位置・品詞を数え、**最多の分割位置・品詞**を対応付け

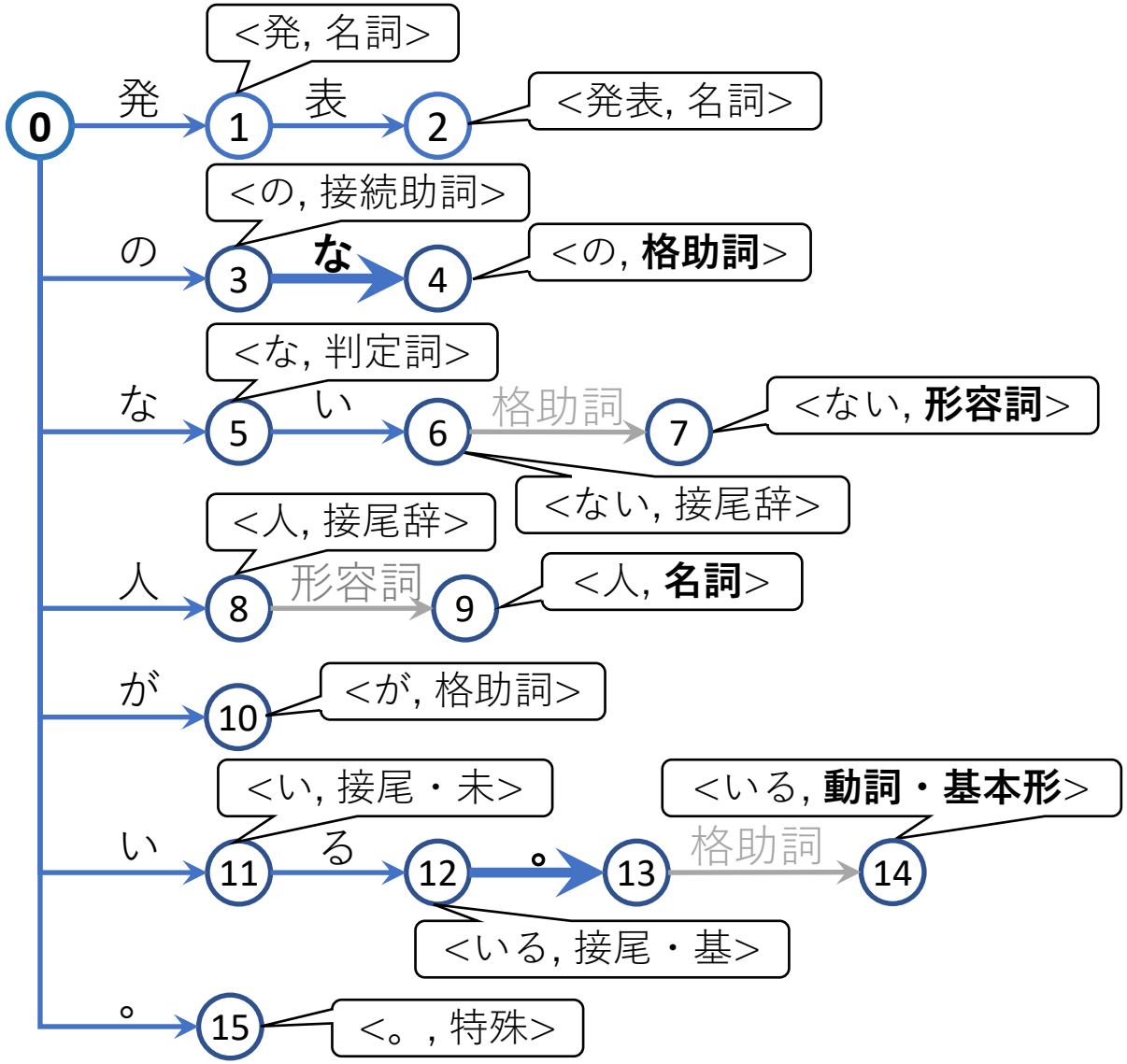
本研究では案2を採用し
学習データに出現しない辞書中の単語からもパターンを抽出

提案手法 – 適用事例

- 例: 発表のない人がいる。

パターン	単語	品詞
発表|	発表	名詞
の|**な**	の	格助詞
ない|_格助詞	ない	形容詞
人|_形容詞	人	名詞
が|	が	格助詞
いる|。_格助詞	いる	動詞
。|	。	特殊

特徴列（パターン）トライ（抜粋）

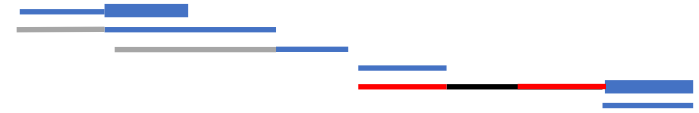


分割位置・品詞が同じ葉ノードは枝刈り

実装上の課題 (1/2) – パターンを辿る際の曖昧性

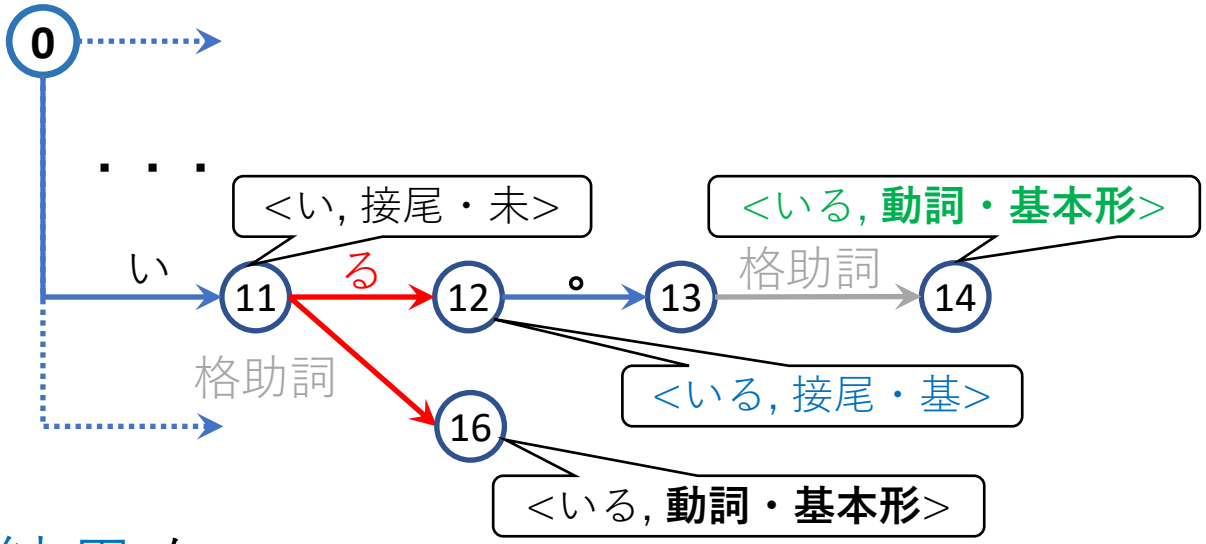
- マッチするパターンを特徴列トライから探索する際には **特徴を辿る順序**に曖昧性が存在する

発表のない人がいる。



- 解決策: **後文脈を優先**して探索
  - 実際はパターンの prefix ごとに辿る順序が一意であれば良い
  - 表層が最長一致するパターンの結果を保持しつつ、前文脈品詞を後戻り探索

特徴列（パターン）トライ（抜粋）



後文脈表層 or 前文脈品詞？

実装上の課題 (2/2) – 候補パターン探索の効率化

- 学習データからどのようにパターンを抽出するか？
  - 後文脈表層を辿と、最多の分割位置・品詞が変わりうる
  - 単純に、頻度に基づいて枝刈りすると精度は大きく低下

- 解決策: 固定長探索＋ヒューリスティクスを併用した列挙
  - パターン（後文脈表層）の最大長を辞書中の最大単語長に制限
  - パターンを列挙する際、既出系列から1文字だけ追加で走査し探索を打ち切る（ヒューリスティクス）
    - メモリ消費・抽出時間 1/2 - 1/5、精度 -0.05%

抽出時間は京都大学テキストコーパスの標準分割で6秒未満

効率的な実装のための工夫 (1/2)

- **高速化・省メモリ化のための工夫 (2x 高速化)**
  - 基本: 連想配列は配列で実装、かつ配列の数は極力減らす
  - 文字IDで遷移するダブル配列にパターンを格納（16%高速化）
  - 学習データ中の頻度順で文字・品詞をID化（5％高速化）
  - 出力をバッファリング・最適化（30％高速化）
  - メモリマップド I/O (60%メモリ消費減)

自作の動的ダブル配列 cedar を拡張

- **未知語処理**
  - 数字、アルファベットは連結、 カタカナは一定長まで連結
  - 品詞は前文脈品詞のみに基づくパターンにより決める

実装は C++ で 958 行（cedar 387 行を含む）とコンパクト

効率的な実装のための工夫 (2/2)

- 配列の数、及びアクセス回数を減らすため、パターンを保持したダブル配列の値 (32 bits) として、以下を格納
  - 分割位置: 9 bits (2^9 = 512 B; パターン長の offset にすれば削減可)
  - 文字種 (未知語処理用): 3 bits (2^3 = 8 種類)
  - 形態素 ID (パターン頻度順): 20 bits (2^{20} = 100 万語)

- 形態素 ID から、以下をまとめて引ける (64 bits)
  - 形態素情報を連結した文字列への offset: 30 bits (2^{30} = 1 GiB)
  - 形態素情報の文字列長 (出力コピー用): 13 bits (2^{13} = 8192 B)
  - 品詞情報の文字列長: 7 bits (2^7 = 128 B)
  - 品詞 ID: 14 bits (2^{14} = 16,384 種、文字と合わせて頻度順 ID 化)

現状の実装は形態素解析に最適化

評価実験

- 標準コーパスを用いて提案手法の実装 (**Jagger**) を評価
  - 京大コーパス(Kyoto)・ウェブ文書リードコーパス (KWDLC)
  - 辞書: mecab-jumandic-5.1, 7.0 (5.1 + ウェブから獲得した語彙)

語彙項目数: 475,716 (5.1) -> 702,358 (7.0)

- **比較手法 (実装)**: 実用的な解析速度の形態素解析器
  - 最小コスト法 [Kudo+ 2004]: MeCab 0.996, **Vibrato 0.5.0**
  - 点推定 [Neubig+ 2011]: Vaporetto **0.6.2** + 未知語処理（品詞タグ付け）

開発データでハイパラ調整しつつ学習

- **評価尺度**
  - 解析速度・消費メモリ（評価データを1000回解析x3 の中央値）
  - 解析精度: F_1（単語分割、品詞タグ付け）

M2 MacBook Air で計測

実験結果 (1/4)

- 辞書として **mecab-jumandic 5.1** を用いて評価
  - 標準分割を利用; 学習文数: 35,478 (Kyoto), 12,271 (KWDLC)

新聞 (Kyoto)	解析速度 [文/秒]↑	メモリ [MiB]↓	単語分割	品詞タグ
MeCab	66,455	55.81	98.68	95.97
Vibrato	142,983	97.75	-	-
Vaporetto	117,767	658.80	98.94	96.92
Jagger	**1,007,344**	**26.39**	98.73	96.55

ウェブ (KWDLC)	解析速度 [文/秒]↑	メモリ [MiB]↓	単語分割	品詞タグ
MeCab	92,110	53.88	97.13	94.30
Vibrato	190,703	97.92	-	-
Vaporetto	200,823	642.63	97.35	94.08
Jagger	**1,524,305**	**28.89**	97.17	94.20

- Jagger は**比較実装の 7-16x 高速、1/2 - 1/20 の消費メモリ**
- 解析精度は Kyoto で Vaporetto がやや良いが大差なし

実験結果 (2/4) – 自動拡張した辞書を利用

- 辞書として **mecab-jumandic 7.0** を用いて評価
  - 標準分割を利用; 学習文数: 35,478 (Kyoto), 12,271 (KWDLC)

新聞 (Kyoto)	解析速度 [文/秒]↑	メモリ [MiB]↓	単語分割	品詞タグ
MeCab	59,453	77.98	98.37	96.10
Vibrato	111,367	164.20	-	-
Vaporetto	105,316	828.85	99.08	97.05
Jagger	**974,316**	**35.09**	98.68	96.57

ウェブ (KWDLC)	解析速度 [文/秒]↑	メモリ [MiB]↓	単語分割	品詞タグ
MeCab	81,598	76.38	97.99	95.62
Vibrato	146,235	163.99	-	-
Vaporetto	174,900	842.40	97.53	94.68
Jagger	**1,503,424**	**40.22**	97.60	94.63

- Jagger と**比較実装との解析効率の差が拡大**（特にVibrato）
- MeCab は KWDLC で解析精度が改善、Kyoto で一部悪化

実験結果 (3/4) – 単語分割のみの処理効率

- 点推定との公平な比較のため、単語分割の効率を比較
  - 辞書: mecab-jumandic 7.0

新聞 (Kyoto)	解析速度 [文/秒]↑	メモリ [MiB]↓
MeCab	62,495	40.52
Vibrato	121,375	163.92
Vaporetto	366,119	283.49
Jagger	**1,264,539**	**21.05**
UTF-8 文字分割	5,751,612	-

ウェブ (KWDLC)	解析速度 [文/秒]↑	メモリ [MiB]↓
MeCab	92,110	39.59
Vibrato	157,460	164.30
Vaporetto	510,465	275.11
Jagger	**1,942,477**	**20.16**
UTF-8 文字分割	8,442,307	-

- Vaporetto の処理速度・消費メモリが大幅に改善
- Jagger は**比較手法の 3-21x 高速、1/2 - 1/13 の消費メモリ**

実験結果 (4/4) – クロスドメイン評価

- 学習したモデルを**他ドメイン**のテストデータで評価
  - 辞書: mecab-jumandic 7.0

学習: 新聞 (Kyoto)	評価: 新聞 (Kyoto)		評価: ウェブ (KWDLC)	
	単語分割	品詞タグ	単語分割	品詞タグ
MeCab	98.37	96.10	97.78	94.48
Vaporetto	99.08	97.05	97.05	92.72
Jagger	98.68	96.57	**97.22**	**93.12**

学習: ウェブ (KWDLC)	評価: 新聞 (Kyoto)		評価: ウェブ (KWDLC)	
	単語分割	品詞タグ	単語分割	品詞タグ
MeCab	97.90	94.82	97.99	95.62
Vaporetto	95.76	91.31	97.53	94.68
Jagger	**97.25**	**93.30**	97.60	94.63

- MeCab は性能低下が小さく、Vaporetto は大きい
- Jagger (提案手法) は**両手法の中間的な性能**

追加実験 – JUMAN++ V2 との比較

- JUMAN++ V2 と Jagger (辞書: **mecab-jumandic 7.0**) を比較
  - 標準分割を利用; 学習文数: 35,478 (Kyoto), 12,271 (KWDLC)
  - JUMAN++ は Wikipedia 由来の辞書と大規模 Web テキストから学習したRNN を追加で利用（精度で有利、速度で不利）

新聞 (Kyoto)	解析速度 [文/秒]↑	メモリ [MiB]↓	単語分割	品詞タグ
Jagger	**974,316**	**35.09**	98.68	96.57
JUMAN++	5384	300.80	99.37	97.74

ウェブ (KWDLC)	解析速度 [文/秒]↑	メモリ [MiB]↓	単語分割	品詞タグ
Jagger	**1,503,424**	**40.22**	97.60	94.63
JUMAN++	7753	290.05	98.37	96.42

- 単語分割精度の差は1%未満、品詞タグ付けの精度差は大
- 精度差を埋めるには外部データを経由した蒸留が必要か

考察: Jagger と Sassano (2014) の定性的な比較

- 前文脈品詞の追加による性能改善は軽微 (0.1%)

- 評価は同じ Kyoto コーパスだが、データの分割が異なる
  - 学習データが大きく、評価データより未来のデータが含まれる

- 考慮する表層が文字単位であるためパターンの汎用性大
  - 大|工学部| (Sassano 2014) vs. 大|工学 (Jagger)
  - Jagger で戻り読みを避けるには Aho-Corasick 法が必要

- パターンに割り当てる分割位置は最尤推定
  - Sassano (2014) では最長一致したパターンの直後で必ず分割

まとめと今後の課題

- **最長一致パターンに基づく形態素解析手法**を提案
  - 辞書・学習事例から得た特徴列パターンを利用（**介入容易**）
  - 最長一致パターン＝より多くの特徴量を使った分類

- 標準コーパスを用いて、提案手法の有効性を検証
  - **MeCab と同程度の解析精度で最大16倍高速、1/2の消費メモリ**
  - モデルの学習（＝パターンの抽出）も高速（～6秒）

- **今後の課題**
  - 高精度化（分類モデルを経由したパターン列挙、モデル近似）
  - 他の言語（中国語など）・タスク（構文解析など）への適応

実装 (Jagger) は GPL/LGPL/BSD ライセンスで近日公開予定