情報処理学会論文誌 Vol.59 No.1 227–235 (Jan. 2018)

K-best反復ビタビパージング

林 克彦[1,a)] 永田 昌明[1,b)]

受付日 2017年5月29日, 採録日 2017年10月3日

概要：本稿では確率文脈自由文法に対して，効率的，かつ，最適解の出力を保証したパージングアルゴリズムを提案する．パージングを高速化するには，探索空間から不要な辺を効率的に枝刈りすることが重要である．提案手法では探索空間を徐々に拡げながら，ビタビ内側・外側アルゴリズムを繰り返し動作させることで，最適解のモデルスコア値に対する上限と下限値を効率的に求める．そして，それに基づいて不要な辺の枝刈りを行うことで，本来の探索空間よりも大幅に縮小された空間を探索するだけで最適解を発見する．さらに，本稿では提案手法を K-best パージングに拡張する方法についても示す．英語の Penn Treebank コーパスを用いた実験により，提案手法が通常の CKY 法よりも高速に動作することを示す．また，K-best のサイズが小さい場合，従来の K-best 法よりも高速に動作することも示す．

キーワード：構文解析，K-best 探索法，分枝限定法

K-best Iterative Viterbi Parsing

Katsuhiko Hayashi[1,a)] Masaaki Nagata[1,b)]

Received: May 29, 2017, Accepted: October 3, 2017

Abstract: This paper presents an efficient and optimal parsing algorithm for probabilistic context-free grammars (PCFGs). To achieve faster parsing, our proposal employs a pruning technique to reduce unnecessary edges in the search space. The key is to repetitively conduct Viterbi inside and outside parsing, while gradually expanding the search space to efficiently compute heuristic bounds used for pruning. This paper also shows how to extend this algorithm to extract K-best Viterbi trees. Our experimental results show that the proposed algorithm is faster than the standard CKY parsing algorithm. Moreover, its K-best version is much faster than the Lazy K-best algorithm when K is small.

Keywords: syntactic parsing, K-best search algorithm, branch-and-bound algorithm

1. はじめに

CKY（または，ビタビ内側アルゴリズム）は確率文脈自由文法（Probabilistic context-free grammars; PCFGs）のパージングアルゴリズムとして良く知られている [11]．与えられた PCFG と入力文に対して，このアルゴリズムはチャート表を使った動的計画法に基づいて最適な構文木（最適解）を求める．CKY 法は実装が容易であり，自然言語の統語解析において良く利用されるが，文法のサイズが大きな場合，全解をしらみつぶしに解析するため，解析速度の面で問題が生じる．

この CKY 法でかかるパージングの計算コストを削減するための手段として，解析中にチャート表上で生成される辺を枝刈りすることが考えられる．近年開発された統語解析システムの多くでも，ビーム探索 [21] や多段探索 [1] のような枝刈り技術を採用することで，パージングにかかる解析速度を大幅に削減することに成功している．しかし，そのような実用面における成功の反面，前述した枝刈り技術は近似的な手法であるため，統語解析システムが最適解を出力する保証は失われる．

別の研究路線として，A*探索に基づくパージングアルゴリズムが良く研究されてきた（A*法）．A*法 [14] では元

[1] 日本電信電話株式会社 NTT コミュニケーション科学基礎研究所
NTT Communication Science Laboratories, NTT Corporation, Soraku-gun, Kyoto 619–0237, Japan
[a)] hayashi.katsuhiko@lab.ntt.co.jp
[b)] nagata.masaaki@lab.ntt.co.jp

の問題よりも簡単な問題を解くことで得られる見積り値を使って，解析中に処理する辺に優先度を付けることで解析を高速化する．また，見積り値が *consistent* であるとき，A*法は最適解を出力することが保証される．しかしながら，文献 [23] で言及されているように，A*法は実装の観点から深刻な問題を有する：

> One of the most efficient ways to implement an agenda, which keeps edges to be processed in A* parsing, is to use a priority queue, which requires a computational cost of $O(\log(n))$ at each action, where n is the number of edges in the agenda. The cost of $O(\log(n))$ makes it difficult to build a fast parser by using the A* algorithm.

そのため，著者らが知る限り，A*法に基づく統語解析システムで実用化されているものは Stanford Lexicalized PCFG パーザなど限られている．

本稿では統語解析システムが出力する解の最適性を保持しながら，不要な辺を効率的に枝刈りする新たな手法を提案する．この手法に基づくアルゴリズムを反復ビタビパージング（iterative Viterbi parsing; IVP）法と呼ぶ．この名称の理由として，提案手法では反復的な動作がその中心的な役割を果たすからである．IVP 法は探索空間を徐々に拡げながら，ビタビ内側・外側アルゴリズムを繰り返し動作させることで，最適解のモデルスコア値に対する上限および下限値を効率的に計算する．そして，その上限，下限値を使って，辺の枝刈りを行うことで，探索空間の一部を探索するだけで最適解を発見することが可能になる．IVP 法はチャート表を使った動的計画法に基づくため実装は容易であり，実験から実用面でも CKY 法より高速に動作することを示す．

さらに，本稿では IVP 法を K-best 解が得られるように拡張する方法についても示す．このアイディアは単純であり，IVP 法の反復動作に Huang と Chiang の K-best アルゴリズム 3 [9]（Lazy アルゴリズム）を統合するものである．Lazy アルゴリズムはビタビ内側アルゴリズムを動作させた後，トップダウン式に K-best 解を探索する．Lazy アルゴリズムにおいて，最初のビタビ内側アルゴリズムが処理の大半を占めるため，速度面での大きなボトルネックとなる．しかし，我々の K-best IVP 法では通常の IVP 法と同じく，ビタビ内側アルゴリズムの処理を大幅に削減することが可能である．

本稿は文献 [6] での発表を拡張した内容である．

2. 反復ビタビパージング

文献 [18] に従って，いくつかの記法を定義する．IVP 法は PCFG G と文長 n の文 $x = t_0 \dots t_{n-1}$ を入力として受け取る（$t_0 \dots t_{n-1}$ は終端記号）．一般性を失うことなく，ここでは G の文法規則がチョムスキー標準形に従うとする：非終端記号を持つ G 中の各規則 r が A → B C の形式をとる．ここで A, B, C は非終端記号であり，この規則の対数尤度を $\log q(r)$ として書く．本稿では A → B の形をとる規則（B は非終端記号または終端記号）については説明を簡略化するため，定義上は扱わないこととする．

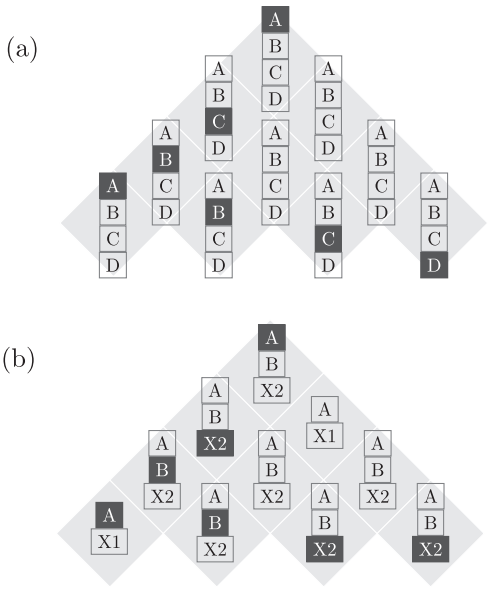

図 1 (a) 非終端記号のみからなる本来のチャート表．(b) 非終端記号および縮退記号からなる縮退チャート表．図 (b) の黒塗りされた記号から構成される導出 A(X2(B(A B) X2) X2) は図 (a) の黒塗りされた記号から構成される導出 A(C(B(A B) C) D) に対応する

Fig. 1 (a) An original chart table consisting of non-terminal symbols only. (b) A coarse chart table consisting of both non-terminal symbols and shrinkage symbols. There exists a corresponding derivation A(X2(B(X1 B) X2) X2) in (b) to a derivation A(C(B(A B) C) D) in (a), both consist of black-shaded symbols.

CKY 法や IVP 法では**図 1** のようなチャート表を使って解析が行われる．チャート表の各セルにはチャート辺，または，辺の集合が格納される．辺とは記号付きの入力文スパン $e = (\mathrm{A}, i, j)$ である．辺 $e = (\mathrm{A}, i, j)$ の導出とは記号 A をルートとして，終端記号 $t_i \dots t_{j-1}$ を葉として広がる木である．ある導出 d に含まれる規則の集合を $R(d)$ とすると，d のスコアは $s(d) = \sum_{r \in R(d)} \log q(r)$ と書ける．辺 e に対して最良となる導出のスコア $\beta(e)$ はビタビ内側スコアと呼ばれる．一方，TOP → $t_0 \dots t_{i-1}$ A $t_j \dots t_{n-1}$ の導出に対する最良のスコア $\alpha(e)$ をビタビ外側スコアと呼ぶ．PCFG パージングの目的は特殊な記号 TOP を持つ終了辺 $(\mathrm{TOP}, 0, n)$ に対する最良の導出を求めることである．各終了辺に対して，その導出を終了導出と呼ぶ．本稿では最良の終了導出をビタビ終了導出と呼ぶ．

ここで非終端記号の集合 $\{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$ を考える．仮にチャート表のセルに現れるいくつかの非終端記号をグループ化すると，本来のチャート表よりも縮退されたチャー

ト表を構築することができる．図 1 (a) は非終端記号のみからなる本来のチャート表を表している一方，図 1 (b) のチャート表は非終端記号と縮退された記号 X1 と X2 からなる．この新たな記号はいくつかの非終端記号をグループ化することで作られ，ここでは縮退記号と呼ぶ．また，縮退記号を含むチャート表を縮退チャート表と呼ぶ．たとえば，縮退記号 X1 と X2 はそれぞれ非終端記号の集合 $\{\mathrm{B},\mathrm{C},\mathrm{D}\}$ と $\{\mathrm{C},\mathrm{D}\}$ とをグループ化して作られており，図 1 (b) は縮退チャート表である．

縮退記号 X, X′, X″ が現れる文法規則の対数尤度パラメータを次のように設定する：

$$\log q(\mathrm{X} \to \mathrm{B}\ \mathrm{C}) = \max_{\mathrm{A} \in I(\mathrm{X})} \log q(\mathrm{A} \to \mathrm{B}\ \mathrm{C})$$

$$\log q(\mathrm{A} \to \mathrm{X}\ \mathrm{C}) = \max_{\mathrm{B} \in I(\mathrm{X})} \log q(\mathrm{A} \to \mathrm{B}\ \mathrm{C})$$

$$\log q(\mathrm{A} \to \mathrm{B}\ \mathrm{X}) = \max_{\mathrm{C} \in I(\mathrm{X})} \log q(\mathrm{A} \to \mathrm{B}\ \mathrm{C})$$

$$\log q(\mathrm{A} \to \mathrm{X}\ \mathrm{X}') = \max_{\mathrm{B} \in I(\mathrm{X}), \mathrm{C} \in I(\mathrm{X}')} \log q(\mathrm{A} \to \mathrm{B}\ \mathrm{C})$$

$$\log q(\mathrm{X} \to \mathrm{X}'\ \mathrm{C}) = \max_{\mathrm{A} \in I(\mathrm{X}), \mathrm{B} \in I(\mathrm{X}')} \log q(\mathrm{A} \to \mathrm{B}\ \mathrm{C})$$

$$\log q(\mathrm{X} \to \mathrm{B}\ \mathrm{X}') = \max_{\mathrm{A} \in I(\mathrm{X}), \mathrm{C} \in I(\mathrm{X}')} \log q(\mathrm{A} \to \mathrm{B}\ \mathrm{C})$$

$$\log q(\mathrm{X} \to \mathrm{X}'\ \mathrm{X}'') = \max_{\substack{\mathrm{A} \in I(\mathrm{X}) \\ \mathrm{B} \in I(\mathrm{X}') \\ \mathrm{C} \in I(\mathrm{X}'')}} \log q(\mathrm{A} \to \mathrm{B}\ \mathrm{C}).$$

ここで $I(\mathrm{X})$ は縮退記号 X からそれに対応する非終端記号の集合への写像とする．この構築により，縮退チャート表上の導出スコアは本来のチャート表上でそれに対応する導出スコアの上限値を与える [14]．そして，次の定理を導くことができる：

定理 1. 縮退チャート表上におけるビタビ終了導出 $\hat{d}$ が縮退記号を 1 つも含まない場合，その導出は本来のチャート表におけるビタビ終了導出と一致する．

証明. $\mathcal{Y}$ を本来のチャート表に含まれるすべての終了導出の集合とする．$\mathcal{Y}' \subset \mathcal{Y}$ を $\mathcal{Y}$ の部分集合とし，この集合の要素は現在の縮退チャート表に現れないすべての終了導出とする．$\mathcal{Y}''$ は現在の縮退チャート表に現れるすべての終了導出の集合とする．ここで各終了導出 $d' \in \mathcal{Y}'$ に対して，図 1 に示した導出の例のように，縮退チャート表にはそれに対応する終了導出が唯一存在する．このとき，$\mathcal{Y}''$ において，d' に対応する終了導出を d'' とすると，

$$\forall d' \in \mathcal{Y}', \exists d'' \in \mathcal{Y}'',\ s(d') \leq s(d'') < s(\hat{d})$$

は明らかであり，これは $\hat{d}$ が本来のチャート表においても最良の終了導出であることを意味する． □

ここですべての大文字アルファベットが非終端記号であると考える．このとき，IVP 法では縮退記号を $\mathrm{X1} = \{B, \ldots, Z\}$，$\mathrm{X2} = \{C, \ldots, Z\}$，$\mathrm{X3} = \{E, \ldots, Z\}$，$\mathrm{X4} = \{I, \ldots, Z\}$，... のように集合に含まれない記号が倍々となるよう順に定義しておく．IVP 法のアルゴリズムは 1 つの非終端記号と 1 つの縮退記号 X1 からなる縮退チャート表を初期化して始まる．次に最良の導出を探索するため，ビタビ内側アルゴリズムを行う．もし，そのビタビ終了導出が縮退記号を 1 つも含まない場合，アルゴリズムはその結果を返して，終了する．そうでない場合，チャート表を拡大して，上記の手続きを終了するまで繰り返す．

Algorithm 1 反復ビタビパージング

```
lb ← det(x, G) or lb ← −∞
chart ← init-chart(x, G)
for all i ∈ [1 . . . ] do
    t̂ ← Viterbi-inside(chart)
    if t̂ consists of non-terminals only then
        return t̂
    end if
    if lb < best(chart) then
        lb ← best(chart)
    end if
    expand-chart(chart, t̂, G)
    Viterbi-outside(chart)
    prune-chart(chart, lb)
end for
```

IVP 法の疑似コードを Algorithm 1 に与える．ここでは，各イテレーションごとの処理を効率化するため，不要な辺を枝刈りしている．また，これにともなう処理を効率化するには，チャート表をどのように展開するかも重要である．よって，以下ではこれらの詳細について述べる．

2.1 枝刈りの手続き

ある辺 $e = (\mathrm{A}, i, j)$ に対して，e を経由するビタビ終了導出のスコアを $\alpha\beta(e) = \alpha(e) + \beta(e)$ と書くことができる (e のビタビ内側・外側スコア)．ここで $\mathcal{Y}$ を本来のチャート表上でのすべての終了導出の集合とする．このとき，仮に $lb \leq \max_{d \in \mathcal{Y}} s(d)$ となる下限値 lb が分かっていれば，$\alpha\beta(e) < lb$ となる辺 e はもはや不要であることが分かる．$\alpha\beta(e)$ を本来のチャート表上で計算することは非常にコストがかかる一方，縮退チャート表上ではビタビ内側・外側アルゴリズムを使って，そのスコアの上限値を効率的に求めることができる：

$$\alpha\beta(e) \leq \widehat{\alpha}(e) + \widehat{\beta}(e) = \widehat{\alpha\beta}(e).$$

ここで $\widehat{\alpha}(e)$ と $\widehat{\beta}(e)$ は縮退チャート表上の辺 e のビタビ内側スコアとビタビ外側スコアとする．もし，$\widehat{\alpha\beta}(e) < lb$ となれば，上の不等式から本来のチャート表の中で e に対応する辺がもはや不要であることが分かるので，その縮退チャート表から e を安全に枝刈りすることができる．

Algorithm 1 では本来のチャート表上で決定的なチャートパージング $\det()$ を使って得られる終了導出のスコアを

下限値 lb として設定する．det() では，チャートの各セルにおいて，最良の導出に対する辺だけを残して，ボトムアップにチャート解析を行う．det() ではほとんどの辺を枝刈りするため（ビーム幅 1 のビーム探索と等価），非常に高速である．しかし，パージングの効率を良くするには，よりタイトな下限値を求めることが重要になる．そのため，best() 関数を使って，現在の縮退チャート表の中で非終端記号のみを持つ最良の導出を求める[*1]．そして，そのスコアが現在の下限値を上回るとき，lb をそのスコアで更新する．これにより多くの不要な辺を解析途中で枝刈りすることが可能となる．

重要なこととして，この枝刈りを行う場合と行わない場合において，ある文を解析し終えるのにかかるイテレーション数はまったく同じであることを注意しておく．また，各イテレーションで選ばれる導出もまったく同じである．そのため，この枝刈りは各イテレーションにかかる処理を高速化することのみに寄与する操作である．

2.2 チャート表の展開手続き

Algorithm 1 の 11 行目において，現在のチャート表を展開する方法は様々に考えられる．本稿ではまず訓練コーパス上での頻度に基づいて非終端記号の順序を降順に決め，チャート表上の各セルでは出現頻度の高い記号から順に展開することを考える．縮退記号は先に述べたアルファベットの例のように倍々で非終端記号が展開されるように作っておく．そして，各イテレーションにおいて，記号を展開するセルの選択は次の「均一チャート展開法」と「最良導出チャート展開法」の 2 つの方法を考える．

2.2.1 均一チャート展開法

現在のチャート表におけるすべてのセルの非終端記号の数を 2 倍にする．この方法では非終端記号の数を N とするとき，イテレーションの最小回数は 1 回で，最大回数は $\lceil \log_2 N \rceil + 1$ 回となる．最小回数で解析が収束するとき，各セルの記号の数は 2 個であるため，計算時間は $O(n^3)$ である．ただし，本来のチャート表上での決定的なパージングにより下限値を初期化する場合，$O(n^3 N)$ の計算時間が必要である．一方，i 番目の反復では 2^{i-1} 個の非終端記号が各セルには存在する．そのため，

$$\sum_{i=1}^{\lceil \log_2 N \rceil + 1} n^3 8^{i-1} = n^3 \frac{1 - 8^{\lceil \log_2 N \rceil + 1}}{1 - 8} = \frac{8n^3 N^3 - n^3}{7} < \frac{8}{7} n^3 N^3$$

となり，IVP 法にかかる最悪の計算時間は $O(n^3 N^3)$ となる．これは通常の CKY 法と同じ計算時間であり，提案法が最悪の場合でも通常の CKY 法と同程度の時間で動作することが分かる．空間計算量について，CKY 法では $O(N^3)$ の規則に対するスコアを保持した行列が必要になる一方，提案法では $O((N + \lceil \log_2 N \rceil)^3)$ の規則を保持する必要がある．

[*1] 8〜10 行目の処理は 4 行目の Viterbi-inside() 関数を少し修正するだけで，それと同時に処理できるため，実際はそのような実装にしている．

2.2.2 最良導出チャート展開法

現在のチャート表上におけるビタビ終了導出が縮退記号を持つとき，その縮退記号が現れるセルだけ非終端記号の数を 2 倍にする．この戦略ではイテレーションの最大回数は $n^2 \cdot (\lceil \log_2 N \rceil + 1)$ となるため，理論上の理にかなった時間計算量を見積もることはもはや意味がない．しかしながら，我々の予備実験では，均一チャート展開法よりも最良導出チャート展開法の方が実際のパージングを格段に高速化できることが分かった．そのため，本稿では均一チャート展開法は用いず，この最良導出チャート展開法を使って実験を行う．

3. 階層型 IVP 法

近年，A*探索の良質な見積り値を効率的に計算するため，問題の階層構造を利用する手法（階層型 A*探索）がさかんに研究されている [4]．自然言語の統語解析においても，非終端記号を階層的に定義し，各階層の記号に基づく文法をそれぞれ構築しておくことがある（階層文法）．そして，階層文法における低次の文法を使った解析結果を高次の文法による解析の見積り値として段階的に利用する階層型 A*法の研究が行われた [17]．本稿の IVP 法でもこのような階層的な知識を利用することができれば，問題により適した探索を行うことができると考えられる．

元の文法 G が持つ非終端記号の集合を Σ とする．ここで Σ の記号を何らかの方法により $m+1$ 段階の階層的なクラスタとして定義し，各階層の記号の集合を $\Sigma_0, \ldots, \Sigma_m$ $(\Sigma_m = \Sigma)$ とする．ある $i \in [0 \ldots m-1]$ に対して，Σ_i の要素は i 階層の縮退記号と呼ぶ．また，ある $0 \le i \le j \le m$ に対して，階層的な記号の定義では Σ_i の要素から Σ_j の部分集合への写像 $\pi_{i \to j} : \Sigma_i \mapsto \mathfrak{P}(\Sigma_j)$ を考える．ここで $\mathfrak{P}$ は冪集合とする．また，$i = j$ のとき，$\pi_{i \to j}$ は入力となる要素をそのままシングルトンとして返す．この写像を使って，任意の $0 \le i, j, k \le m$ の階層の記号に対して（i, j, k のうち，少なくとも 1 つは m より小さい），縮退記号を含む文法規則の対数尤度パラメータは

$$\log q(\mathrm{X}_i \to \mathrm{X}_j\ \mathrm{X}_k) = \max_{\substack{\mathrm{A} \in \pi_{i \to m}(\mathrm{X}_i) \\ \mathrm{B} \in \pi_{j \to m}(\mathrm{X}_j) \\ \mathrm{C} \in \pi_{k \to m}(\mathrm{X}_k)}} \log q(\mathrm{A} \to \mathrm{B}\ \mathrm{C})$$

として求めることができる．$\mathrm{A} \to \mathrm{B}\ \mathrm{C}$ が存在しない場合，スコアは $-\infty$ として設定しておく．

本稿では文献 [1], [20] の定義を参考にして，付録 A.1 のような非終端記号（と品詞タグ）の階層的なクラスタを定義した．このような階層的な記号定義を使った IVP 法（階層型 IVP 法）は，0 階層のラベルからなる縮退チャート表

を初期化して始まり，元の非終端記号のみからなる終了導出が見つかるまでIVP法と同じように反復処理を繰り返す．チャート表上で展開するセルの選択は最良導出チャート展開法を用い，その終了導出がi階層の縮退記号を含むとき，その記号に対応する次の階層の記号集合をそのセルに展開する．

4. K-best拡張

文献[9]のLazyアルゴリズムはビタビ内側アルゴリズムを動作させた後，トップダウンにk-best導出木を探索する．K-best IVP法はIVP法の反復処理とLazyアルゴリズムを統合することで，とりわけLazyアルゴリズムのビタビ内側アルゴリズムを高速化することに狙いがある．Alogirhtm 2にK-best IVP法のアルゴリズムを示す．基本的な動作はIVP法と同じであるが，ビタビ終了導出が非終端記号のみからなるとき，K-best IVP法ではLazyアルゴリズムを動作させてk-best導出木を探索する．もし，すべてのk-best終了導出が縮約記号を1つも含まないとき，その結果を返して，アルゴリズムは終了する．

K-best IVP法でも不要な辺を枝刈りして，各反復イテレーションごとの処理を効率化する．まず，本来のチャート表上でビーム幅kのビーム探索に基づくパージングアルゴリズムを動かすことで，k個の終了導出を得る．そのとき，k番目の終了導出のスコアを下限値lbとして初期化する．枝刈りをより効率化するため，各イテレーションごとにlbを更新する．k-best() 関数を使って，現在のチャート上において，非終端記号のみからなる導出の中でk番目に最良である終了導出のスコアを求め，この値がlbを上回るとき，更新を行う．k-best() 関数の具体的な手続きは，Lazyアルゴリズムを非終端記号からなる辺だけを使って動作させることで行える[*2]．

Algorithm 2 K-best反復ビタビパージング

```
 1: lb ← beam(x, G, k) or lb ← −∞
 2: chart ← init-chart(x, G)
 3: for all i ∈ [1 . . . ] do
 4:   t̂_1 ← Viterbi-inside(chart)
 5:   if t̂_1 consists of non-terminals only then
 6:     [t̂_2, . . . , t̂_k] ← Lazy K-best(chart)
 7:     if All of [t̂_2, . . . , t̂_k] consist of non-terminals only
        then
 8:       return [t̂_1, t̂_2, . . . , t̂_k]
 9:     else
10:       t̂_1 ← getShrinkDerivTree([t̂_2, . . . , t̂_k])
11:     end if
12:   end if
13:   if lb < k-best(chart, k) then
14:     lb ← k-best(chart, k)
15:   end if
16:   expand-chart(chart, t̂_1, G)
17:   Viterbi-outside(chart)
18:   prune-chart(chart, lb)
19: end for
```

他の動作原理はIVP法とほとんど同様であるが，10行目のgetShrinkDerivTree() 関数では，k-best終了導出の中から縮退記号を含む終了導出のうちでスコアが最大のものを取得しており，それを16行目の最良導出チャート展開に利用している．アルゴリズムの正当性については，下限値lbがつねにk番目に最良の導出のスコアを下回るように設定されているため，明らかである．

5. 実験

5.1 実験設定

実験では英語Penn Treebankの Wall street journal部分のセクション02–21を訓練データ，セクション22の長さ1–35の文をテストデータとして用いた．訓練データから機能ラベル，空範疇タグ，フィラー表示を取り除いて，右分岐の2分化を施した後，チョムスキー標準形に従うPCFGを最尤推定で求めた．非終端記号の数は52（品詞の数は45）であった．このようなPCFGは比較的小規模な部類の文法ではあるが，通常のCKY法はこれよりも大きなサイズ（記号数が増える）になると効率的に動作しないため，提案法の有効性を示すためのベンチマークデータとしては十分である．解析時間の計測にはXeon 5600 3.33 GHzのCPUマシンを利用した．

5.2 1-bestパージングの実験

表1ではCKY法，IVP法，階層型IVP法を用いてテストデータを解析したときの結果を比較する．実験からは階層型IVP法がCKY法に比べて8–9倍程度高速に動作することが分かる．一方，通常のIVP法は短い文ではCKY法よりも高速に動作したが，テストデータ全体での平均解析速度は同程度であった．IVP法はCKY法と比較して，展開される辺数そのものは少ないが，収束までのイテレーション数がかかるうえ，特に長い文では展開される辺数も相応の数となり，1回辺りの計算コストが重くなるため，解析に時間がかかっている．**図2**にはテストデータの文長ごとの平均解析時間を示した．長い文に対して，IVP法の動作が不安定である一方，階層型IVP法は安定して動作していることが分かる．

表1からIVP法と階層型IVP法では展開される辺の数が平均で2倍以上異なっている．これは階層型IVP法で使われる縮退記号の方が問題により適したクラスタになっているためである．すなわち，イテレーションの初期に選

[*2] この際に使うビタビ内側スコアはViterbi-inside() 関数内で計算した非終端記号のみからなる導出のスコアを使っており，これは1-best時の下限値計算（best() 関数）と同様の手続きである．

表 1 CKY 法，IVP 法，および，階層型 IVP 法（HIVP）の比較：解析時に展開された辺数，枝刈りされた辺数，収束までのイテレーション数，解析時間を示している．最終行はそれぞれの結果に対する平均値を示す

Table 1 Main results for the CKY, IVP and HIVP methods: This table shows the number of edges, the number of the pruned edges and the number of iterations until the convergence. The last row denotes the mean values.

	CKY 法		IVP 法				HIVP 法			
文長	edges	time	edges	pruned edges	iters	time	edges	pruned edges	iters	time
20	10590	1.25	5290	4499	56	0.24	2864	2089	68	0.13
23	13938	1.76	5342	4447	60	0.06	2219	1462	41	0.06
22	12771	1.52	4516	3767	58	0.08	2204	1425	46	0.05
17	7701	0.72	3791	3237	42	0.05	1526	1119	32	0.03
28	20538	3.14	15801	13830	134	4.88	7306	5338	144	1.18
34	30141	5.44	15377	13150	134	2.74	6390	4634	98	0.49
⋮	⋮		⋮				⋮			
21	12801	1.77	7864	6764	74	1.71	3502	2456	70	0.21

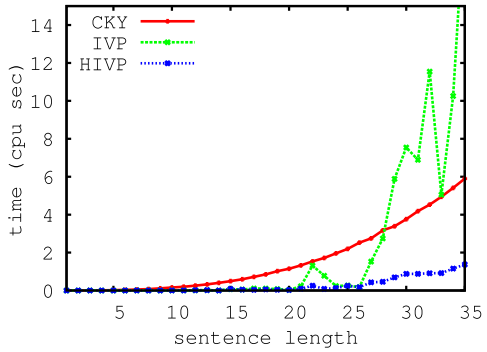

図 2 CKY 法，IVP 法，階層型 IVP 法（HIVP）に対するテストデータの文長ごとの解析時間

Fig. 2 The parsing time of the CKY, IVP and HIVP methods for sentences with various lengths.

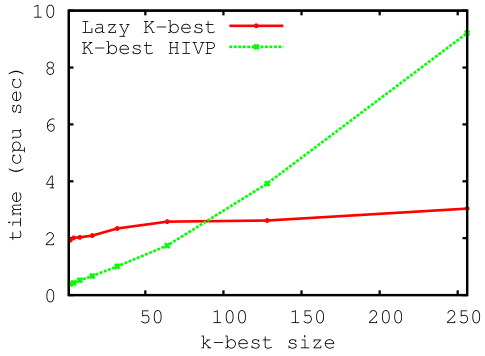

図 4 Lazy 法と K-best HIVP 法の平均解析時間：k は 2, 4, 8, 16, 32, 64, 128, 256 に設定してプロットした

Fig. 4 The average parsing time of the Lazy and K-best HIVP methods when setting k to 2, 4, 8, 16, 32, 64, 128, 256.

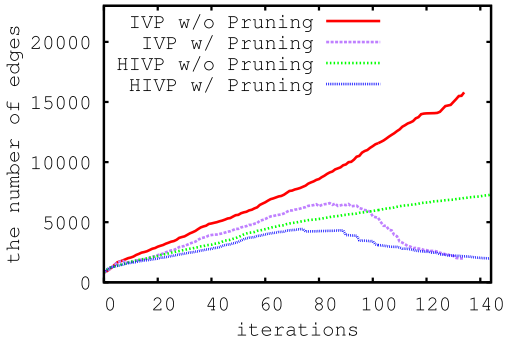

図 3 文長 28 の文に対する IVP 法，および，階層型 IVP 法（HIVP）における各イテレーションごとの辺数

Fig. 3 The number of edges in each iteration of the IVP and HIVP methods for a sentence with length 28.

ばれる最良の導出が本来の最良の導出とより良く対応がとれているため，イテレーションごとに展開される縮退記号の総数が減ったと考えられる．**図 3** では表 1 に示している文長 28 の文に対する IVP 法と階層型 IVP 法におけるイテレーションごとの辺の数を示した．図 3 から階層型 IVP 法では辺の数の増加を少なく抑えられていることが分かる．また，両手法ともイテレーションの中盤に辺の枝刈りが良く機能していることが分かる．

5.3 K-best パージングの実験

1-best パージングでの実験結果をふまえて，K-best パージングの実験では Lazy アルゴリズムと K-best 階層型 IVP 法（Kbest HIVP 法）の比較を行う．前述したように，Lazy アルゴリズムの主要な処理は最初に CKY 法を動作させる点である．K-best HIVP 法では 1-best の場合と同じく，その処理を減らすことが高速化のポイントになる．予備実験から，ビーム探索による下限値の設定は計算コストが非常に大きかったため，下限値の初期値は $-\infty$ に設定した．

図 4 では Lazy アルゴリズムと K-best HIVP 法のテストデータに対する解析時間の平均をプロットした．K-best HIVP 法は k を小さく設定したとき（$k \leq 64$ 程度），Lazy アルゴリズムより高速に動作した．しかし，k を大きく設定すると，K-best HIVP 法の解析速度は大幅に減速した．

表 2 K-best HIVP 法の実験結果
Table 2 Main results for the K-best HIVP method.

	$k=8$				$k=32$				$k=128$			
文長	edges	pruned edges	iters	time	edges	pruned edges	iters	time	edges	pruned edges	iters	time
20	2991	2079	75	0.33	3433	2149	103	0.72	3810	1997	144	2.74
23	2247	1160	44	0.16	2518	1215	65	0.40	3223	1321	108	1.21
22	2493	1441	63	0.13	2899	1427	100	0.69	3464	1041	146	2.37
17	1699	1169	45	0.07	1987	1096	67	0.32	2380	975	102	1.07
28	7654	5488	167	2.15	8312	5779	203	2.76	9062	5560	251	7.13
34	6390	4600	98	0.74	6577	4571	106	0.94	6996	4324	120	1.71
⋮		⋮				⋮				⋮		
21	3767	2515	84	0.52	4193	2548	108	1.01	4809	2479	145	3.92

表 3 Lazy 法（K-best CKY）の実験結果
Table 3 Results for the Lazy method.

	CKY 法		$k=8$	$k=32$	$k=128$
文長	edges	time	time	time	time
20	10590	1.25	2.34	2.48	2.94
23	13938	1.76	1.97	3.42	3.90
22	12771	1.52	1.76	2.10	2.59
17	7701	0.72	0.91	1.14	1.54
28	20538	3.14	3.75	4.22	5.00
34	30141	5.44	5.84	6.39	6.71
⋮	⋮		⋮	⋮	⋮
21	12801	1.77	2.01	2.34	2.61

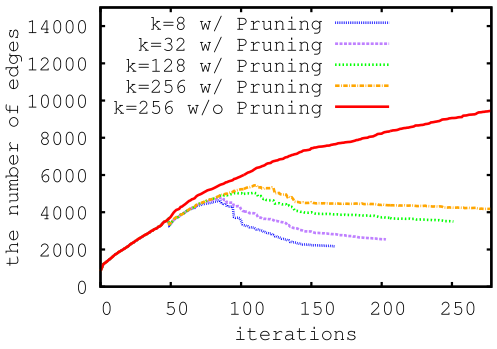

図 5 文長 28 の文に対する K-best HIVP 法の各イテレーションごとの辺数
Fig. 5 The number of edges in each iteration of the K-best HIVP method for a sentence with length 28.

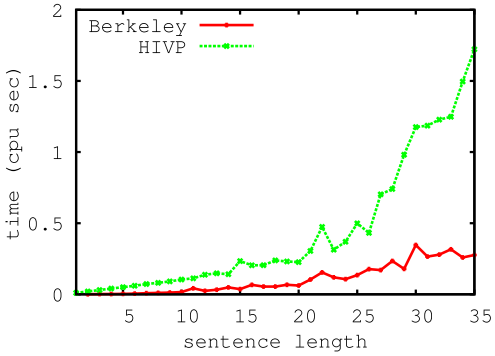

図 6 1-best HIVP 法と Berkeley Parser の速度比較
Fig. 6 The parsing time comparison between the HIVP and Berkeley methods.

表 2 には K-best HIVP 法において展開された辺数，枝刈りされた辺数，収束までのイテレーション数，解析時間を示した．また，**表 3** には Lazy 法による結果を示した．Lazy 法において展開される辺数は CKY 法とまったく同じである．K-best HIVP 法では k が大きくなる程，展開される辺数，および，イテレーション数が増加していることが分かる．**図 5** には文長 28 の文に対するイテレーションごとの辺の数を示した．k が大きな場合，展開される辺数とイテレーション数の増加が進む一方，辺の枝刈りがあまり行われていないため，大幅な減速につながったと考えられる．対して，Lazy 法では図 4 および表 3 から明らかなように，k のサイズを大きくしてもほぼ線形に動作しており，解析時間のボトルネックは CKY 法にあることが分かる．

ただし，自然言語処理で K-best の統語解析結果を使う場合，64-best 程度が一般には上限であるため [2], [5], [7]，提案法は十分に実用的であると考えられる．

5.4 Berkeley Parser との比較

Berkeley Parser [19] は多段探索に基づく枝刈りで階層的な潜在変数付き確率文脈自由文法（L-PCFGs）のパージングを高速化している．現在，このシステムは様々な言語処理に応用されており，HIVP 法を階層的な L-PCFGs に適用することは実用の面から興味深い課題である．ここでは付録 A.1 で定義した最終階層からさらに潜在変数付きの記号を展開することで記号の階層を定義し，HIVP 法の文法として用いた．**図 6** には 1-best HIVP 法と Berkeley Parser を用いてテストデータを解析したときの文長ごとの平均解析時間を示した．結果からは Berkeley Parser との解析速度にはまだ差があり，このことは実用上の課題といえる．ただし，Berkeley Parser は最適解を出力している保証はない．実際，最適解を求めることでテストデータ上で

F 値精度 89.9 が F 値精度 90.2 に上昇した．

6. 関連文献

文献 [23] では反復 CKY 法を提案している．このアルゴリズムはある閾値を使って，それを超える導出木が見つかるまで CKY 法を反復させる点で我々の IVP 法と類似している．しかし，IVP 法ではチャート表を徐々に展開していくことで，より良い上限，下限値を効率的に計算して更新しながら，枝刈りを行う点で異なる．また，文献 [23] では K-best 化について言及されていない．

文献 [14] では A*探索を構文解析に応用する方法が提案されている（A*法）．A*法ではオフライン，または，オンラインで本来の文法よりも簡単な文法を用いて計算される見積り値を利用することで解析を高速化する．文献 [17] では階層型 A*探索 [4] に基づいて，階層文法 [1], [19] を利用した A*法を提案している（階層型 A*法）．

系列デコーダに対して，文献 [12] では反復ビタビ法が提案されている．また，文献 [10] ではその手法を文献 [22] の後ろ向き K-best A*探索と統合することで，k-best の系列がとれるように拡張している．本稿で提案した IVP 法はこれらのアルゴリズムをパージング問題へ一般化したものと見なすことができる．また，系列デコーダの世界では階層型 A*法や本稿の階層型 IVP 法のように記号の階層構造を利用した手法についての議論はいまだ行われていない．

文献 [9] では効率的な K-best ビタビパージングアルゴリズムを提案している．本稿の K-best IVP 法は CKY 法を高速化するのと同じ要領で，この K-best アルゴリズムを高速化している．文献 [8] ではクヌースパージング法 [13], [15] を K-best パージングアルゴリズムへと拡張している．文献 [18] では階層文法を使うことで効率的に見積り値を計算して，K-best クヌースパージング法を K-best A*法へと拡張している．

7. まとめ

本稿では PCFGs に対する効率的，かつ，解の最適性を保証したパージングアルゴリズムを提案した．このアルゴリズムは動的計画法に基づいており，実装は容易である．今後の課題として，潜在変数付き PCFGs [3], [16], [19] のように文法サイズが大幅に増える場合における提案法の有効性をさらに検証していく必要がある．また，人手で記号の階層クラスタを定義できないような場合，パージングを高速化するようなクラスタを自動で取得する方法について考えることも重要な課題である．また，深層学習などの識別モデルを用いた文脈自由文法の解析に提案法を応用する方法も興味深い課題といえる．

参考文献

[1] Charniak, E., Johnson, M., Elsner, M., Austerweil, J., Ellis, D., Haxton, I., Hill, C., Shrivaths, R., Moore, J., Pozar, M., et al.: Multilevel coarse-to-fine PCFG parsing, *Proc. HLT-NAACL*, pp.168–175 (2006).

[2] Choe, D.K. and Charniak, E.: Parsing as Language Modeling, *Proc. EMNLP*, pp.2331–2336 (2016).

[3] Cohen, S.B., Stratos, K., Collins, M., Foster, D.P. and Ungar, L.: Spectral learning of latent-variable PCFGs, *Proc. ACL*, pp.223–231 (2012).

[4] Felzenszwalb, P.F. and McAllester, D.: The generalized A* architecture, *Journal of Artificial Intelligence Research*, Vol.29, pp.153–190 (2007).

[5] Hayashi, K., Kondo, S. and Matsumoto, Y.: Efficient stacked dependency parsing by forest reranking, *Trans. Association for Computational Linguistics*, Vol.1, pp.139–150 (2013).

[6] Hayashi, K. and Nagata, M.: K-best Iterative Viterbi Parsing, *Proc. EACL*, pp.305–310 (2017).

[7] Hayashi, K., Watanabe, T., Asahara, M. and Matsumoto, Y.: Third-order Variational Reranking on Packed-Shared Dependency Forests, *Proc. EMNLP*, pp.1479–1488 (2011).

[8] Huang, L.: K-best Knuth algorithm (2005), available from ⟨http://cis.upenn.edu/~lhuang3/knuth.pdf⟩.

[9] Huang, L. and Chiang, D.: Better K-best parsing, *Proc. IWPT*, pp.53–64 (2005).

[10] Huang, Z., Chang, Y., Long, B., Crespo, J.-F., Dong, A., Keerthi, S. and Wu, S.-L.: Iterative Viterbi A* algorithm for k-best sequential decoding, *Proc. ACL*, pp.611–619 (2012).

[11] Jurafsky, D. and Martin, J.H.: *Speech and Language Processing*, Prentice Hall (2000).

[12] Kaji, N., Fujiwara, Y., Yoshinaga, N. and Kitsuregawa, M.: Efficient staggered decoding for sequence labeling, *Proc. ACL*, pp.485–494 (2010).

[13] Klein, D. and Manning, C.D.: Parsing and hypergraphs, *Proc. IWPT*, pp.123–134 (2001).

[14] Klein, D. and Manning, C.D.: A* parsing: Fast exact Viterbi parse selection, *Proc. HLT-NAACL*, pp.119–126 (2003).

[15] Knuth, D.E.: A generalization of Dijkstra's algorithm, *Information Processing Letters*, Vol.6, No.1, pp.1–5 (1977).

[16] Matsuzaki, T., Miyao, Y. and Tsujii, J.: Probabilistic CFG with latent annotations, *Proc. ACL*, pp.75–82 (2005).

[17] Pauls, A. and Klein, D.: Hierarchical search for parsing, *Proc. HLT-NAACL*, pp.557–565 (2009).

[18] Pauls, A. and Klein, D.: K-best A* parsing, *Proc. ACL*, pp.958–966 (2009).

[19] Petrov, S., Barrett, L., Thibaux, R. and Klein, D.: Learning accurate, compact, and interpretable tree annotation, *Proc. COLING-ACL*, pp.433–440 (2006).

[20] Petrov, S., Das, D. and McDonald, R.: A universal part-of-speech tagset, arXiv preprint arXiv:1104.2086 (2011).

[21] Ratnaparkhi, A.: Learning to parse natural language with maximum entropy models, *Machine Learning*, Vol.34, No.1-3, pp.151–175 (1999).

[22] Soong, F.K. and Huang, E.-F.: A tree-trellis based fast search for finding the n-best sentence hypotheses in continuous speech recognition, *Proc. ICASSP*, pp.705–708 (1991).

[23] Tsuruoka, Y. and Tsujii, J.: Iterative CKY parsing for

probabilistic Context-free Grammars, *Proc. IJCNLP*, pp.52–60 (2004).

付　　録

A.1 非終端記号と品詞タグの階層的縮退記号

階層 0	階層 1	階層 2
HP	S_	S
		VP
		UCP
		SQ
		SBAR
		SBARQ
		SINV
	N_	NP
		NAC
		NX
		LST
		X
		UCP
		FRAG
MP	A_	ADJP
		QP
		CONJP
		ADVP
		INTJ
		PRN
		PRT
	P_	PP
		PRT
		RRC
		WHADJP
		WHADVP
		WHNP
		WHPP
Op_	ADJ_	JJ
		JJR
		JJS
	ADV_	RB
		RBR
		RBS
		WRB
	NOUN_	NN
		NNP
		NNPS
		NNS
	VERB_	MD
		VB
		VBD
		VBG
		VBN
		VBP
		VBZ
Cl_		IN
		CC
		CD
	DET_	DT
		EX
		PDT
		WDT
	PRON_	PRP
		PRP$
		WP
		WP$
	PRT_	POS
		RP
		TO
Ot_	X_	FW
		LS
		SYM
		UH
	PUNC_	#
		$
		”
		“
		-LRB-
		-RRB-
		,
		:
		.

林 克彦 （正会員）

2013 年奈良先端科学技術大学院大学情報科学研究科博士課程修了．同年 NTT コミュニケーション科学基礎研究所入所．自然言語処理の研究に従事．情報処理学会，人工知能学会，言語処理学会，ACL 各会員．

永田 昌明 （正会員）

1987 年京都大学大学院工学研究科修士課程修了．工学博士．同年 NTT 入社．自然言語処理の研究に従事．電子情報通信学会，人工知能学会，言語処理学会，ACL 各会員．