人工知能学会研究会資料
SIG-FPAI-B403-08

# 文法圧縮に基づく自己索引のオンライン構築について

# Fully-Online Construction of Grammar-Based Self-Index

高畠嘉将 [1*]　田部井靖生 [2]　坂本比呂志 [1]
Yoshimasa Takabatake[1]　Yasuo Tabei [1,2]　Hiroshi Sakamoto[1]

[1] 九州工業大学大学院情報工学府
[1] Graduate School of Computer Science and Systems Engineering,
Kyushu Institute of Technology, Japan
[2] PRESTO さきがけ-独立行政法人科学技術振興機構
[2] PRESTO,Japan Science and Technology Agency

**Abstract:** Existing grammar-based self-indexes are efficient for highly repetitive texts. However, the construction space of existing self-indexes depends on input length. Thus, developing an online construction of grammar-based self-index executed on compressed space is important for highly repetitive and streaming texts. In this paper, we present a first online grammar-based self-index named *online ESP-index(OESP-index)*. OESP-index directly encodes an input text into a succinct representation of *straight line program(SLP)* in an online manner based on *fully online LCA(FOLCA)* techniques. The succinct representation of SLP is a wavelet tree and a bit array encoded by *dynamic range min/max tree*. OESP-index supports the pattern search of ESP-index by using such data structures. We experimentally show that the construction of OESP-index for real world texts is executed on compressed space.

## 1　はじめに

DNA シーケンサから出てくるゲノムや web にアップロードされるソースコードやドキュメントのような繰り返しの多いストリームテキストデータを効率的に扱うために圧縮しながら，圧縮したデータ上でパターン検索できるように索引化する技術であるオンラインで構築可能な自己索引が必要である．

繰り返しの多いテキストデータに対して効果的な圧縮手法としては文法圧縮が知られている．

自己索引とは圧縮したデータ自体が索引となっており，圧縮前のテキストに対して，キーワードの出現位置を求める *locating*，キーワードの出現回数を求める *counting*，圧縮前のテキストの $i$ 番目から $j$ 番目の文字列を復元する *extracting* の三つの操作を可能とする索引技術である．文法圧縮型の自己索引である SLP-index[2, 1] と ESP-index[6, 14] も提案されており，繰り返しの多いテキストに対して，高速検索が可能である．しかしながら，文法圧縮型を含めたすべての自己索引においては，少しずつデータを読みながら圧縮領域でストリームデータをオンラインで自己索引化する方法は提案されていない．

そこで本論文では，オンラインかつ圧縮領域で構築可能な文法圧縮型の自己索引 Online ESP-index(OESP-index) を提案する．OESP-index では生成規則は構文木と呼ばれる一つの木構造で表現し，FOLCA[8] の技術を用いて，テキストから直接構文木を符号化することで実現する．FOLCA との違う点は符号化に Wavelet tree を用いる点である．これにより ESP-index の検索に必要な操作を網羅することができる．オフラインの自己索引との比較を表 1,2 にまとめておく．

実験ではベンチマークテキストを用いて，実際に圧縮領域で構築可能であることを示した．各定理や補題の証明はスペースの関係で省略させてもらう．

*連絡先：九州工業大学
〒 820-8502　福岡県飯塚市川津 680-4
E-mail: takabatake@donald.ai.kyutech.ac.jp

## 2　準備

### 2.1　基本概念

文字列 $S$ の長さは $|S|$ と表記する．集合 $C$ の濃度も同様に $|C|$ と表記する．本論文では入力テキストを構成する記号の集合は $\Sigma$ とする $\Sigma$ によって，表現可能なすべての記号列の集合は $\Sigma^*$ と表記し，$\Sigma^i = \{\omega \in \Sigma^* | |\omega| = i\}$ とする．$\mathcal{X}$ は $\Sigma \cap \mathcal{X} \neq$ を満たす変数の帰納的加算集

表 1: オフラインの文法圧縮に基づく自己索引との比較 (メモリとアルゴリズム). $N$ はテキストの長さ，$m$ はクエリーの長さ，$n$ は文法の変数の数，$\sigma$ は文字種類数，$z$ は LZ77 のフレーズの数，$d$ は LZ77 の入れ子の長さとする．$occ$ は，テキスト中のクエリーの出現回数であり，$occ_c$ は，クエリーの出現候補の数である．lg の底は 2 である．$\lg^*$ は lg を適用した繰り返しの回数であり，$\epsilon$ は (0,1) の実数である．

| | Construction space(bits) | Index size(bits) | Algorithm |
|---|---|---|---|
| LZ-index [9] | $O(N)$ | $z \lg N + 5z \lg N$ $-z \lg z + o(N) + O(z)$ | offline |
| Gagie et al. [4] | $O(N)$ | $2n \lg n + O(z \lg N$ $+z \lg z \lg \lg z)$ | offline |
| SLP-index [2, 1] | $O(N)$ | $n \lg N + O(n \lg n)$ | offline |
| ESP-index [14] | $O(N)$ | $n \lg N + n \lg n$ $+2n + o(n \lg n)$ | offline |
| OESP-index | $n \lg N + O(n \lg n)$ | $n \lg N + O(n \lg n)$ | online |

表 2: オフラインの文法圧縮に基づく自己索引との比較（計算時間).

| | Construction time | Searching time | Extraction time |
|---|---|---|---|
| LZ-index [9] | $O(N \lg \sigma)$ | $O(m^2 d + (m + occ)) \lg z$ | $O(md)$ |
| Gagie et al. [4] | $O(N)$ | $O(m^2 + (m + occ) \lg \lg N)$ | $O(m + \lg lgN)$ |
| SLP-index [2, 1] | $O(N)$ | $O((m^2 + h(m + occ)) \lg n)$ | $O((m + h) \lg n)$ |
| ESP-index [14] | $O(\frac{N}{\alpha})$ expected | $O((\lg \lg n)(m + occ_c \lg m \lg N) \lg^* N)$ | $O((\lg \lg n)(m + \lg N))$ |
| OESP-index | $O(\frac{N}{\alpha} \lg n)$ expected | $O((\lg n)(\frac{m}{\alpha} + occ_c \lg m (\lg N)^2))$ | $O(\lg n(m + \lg N))$ |

合とする．このとき $\Sigma \cup \mathcal{X}$ からなる記号列を文字列と呼ぶ．$\Sigma \cup \mathcal{X}$ からなる連続する 2 記号のペアを *digram*, 連続する 3 記号を *trigram* と呼ぶ．文字列 $\omega = xyz$ において，$x$ を接頭辞，$z$ を接尾辞と呼ぶ．また，$x, y, z$ はすべて $\omega$ の部分文字列と呼ばれる．$\omega$ の $i$ 番目の記号は $\omega[i](1 \leq i \leq |\omega|)$ と表記する．$w[i, j]1 \leq i \leq j \leq |\omega|$ は $\omega$ の $i$ 番目から $j$ 番目までの文字列である．$\lg n$ は $\log_2 n$ を表す．$N$ は入力テキストの長さである．

## 2.2 文法圧縮

ここで**文脈自由文法** *(CFG)* は $G = (\Sigma, V, D, Z_S)$ の四つの記号の組によって表現する．ここで $V$ は $\mathcal{X}$ の有限部分集合であり，$D$ は $V \times (V \cup \Sigma)^*$ からなる生成規則の部分集合である．最後の $Z_S \in V$ は開始記号である．本論文では $V \cup \Sigma$ の全順序を想定する．$G$ によって $X_S$ から導出される $\Sigma^*$ 上の文字列の集合は $L(G)$ と表記する．全ての $X \in \mathcal{X}$ に対して，$X \to \gamma \in D$ と $|L(G)| = 1$ であるという条件を満たすとき CFG $G$ が *admissible* であるという．テキスト $S$ を導出する admissible な $G$ は $S$ の文法圧縮と呼ばれる．$G$ のサイズ $|G|$ は全ての生成規則の右側の文字列の長さとする．文法圧縮の問題は次の様に一般化されている:

**定義 1. 文法圧縮** 入力として $\omega \in \Sigma^*$ が与えられて，$\omega$ を導出する出来る限りサイズの小さい admissible な $G$ を生成せよ．

$S(D) \in \Sigma^*$ は文字列 $S \in (\Sigma \cup V)^*$ から $D$ によって導出される文字列とする．例えば，$S = YaY, D = \{X \to ab, Y \to cX\}\Sigma = \{a, b, c\}$ のとき，$S(D) = cabacab$ である．$|X|$ および $|X(D)|$ は $X \in V$ から $D$ によって，導出される文字列の長さである．

本論文では全ての生成規則 $X \to \gamma$ の $|\gamma| = 2$ と仮定する．すべての $n$ 個の変数を持つ文法圧縮は高々$2n$ 個の変数を持つこの制限された CFG に変換可能であるので，この仮定は妥当である．

$G$ の構文木は根付き順序二分木によって表現可能である．このとき，構文木は内部ノードは $V$ からなる変数によって名前付けされ，*yields*(葉の記号列) が $S$ となっている．構文木上ですべての内部ノード $Z \in V$ が生成規則 $Z \to XY$ に対応し，左の子が $X$，右の子が $Y$ によってそれぞれ名前付けされている．根として $X_i$ を持つ構文木の部分木の高さを *height*$(X_i)$ とする．本論文ではこの CFG を $SLP$ とする．

**定義 2.** *Karpinski-Rytter-Shinohara [5]* $SLP$ は生成規則が $X_k \to X_i X_j (X_i, X_j \in \Sigma \cup V, 1 \leq i, j \leq |V| + |\Sigma|)$ の形で表される $\Sigma \cup V$ 上の文法圧縮である．

ここまで述べてきた CFG は厳密な SLP の定義とは異なり，生成規則の右側が digram だけだが，この CFG と SLP は等価である．本論文上では表記の簡略化のためにここまで述べてきた CFG を SLP と表記する．

## 2.3 辞書と逆引き辞書

パターン検索と索引のオンライン構築を行う際にこれらの機能が必要である．辞書とは $X_k$ が与えられて，$X_k \to X_i X_j \in D$ のとき，$X_i X_j$ にアクセス可能な

データ構造である．一般には $n$ 個の生成規則を格納するのに $2n \lg n$bits の二本の配列を用いて，$O(1)$ 時間でアクセス可能なように実装される．逆引き辞書 $D^{-1}$ : $(\Sigma \cup \mathcal{X})^{\in} \to \mathcal{X}$ は digram から非終端記号への写像である．したがって，$D^{-1}(X_iX_j)$ は $X_k \to X_iX_j \in D$ のとき $X_k$ を返す．逆引き辞書は一般的にはハッシュテーブルで実装される．領域は $(2+\alpha)n \lg(n+\sigma)$bits であり，$O(1/\alpha)$ 期待時間で計算可能である．FOLCA[8] の技術と Wavelet Tree を用いて，これらの機能を圧縮領域で実装する．

### 2.4 Wavelet Tree(WT)

WT は文字列 $S$ に対して $access/rank/select$ の三つの関数をサポートする．$access(S,i)$ は $S[i]$ を返す．$rank_c(S,i)$ は $i$ 番目までに出現する文字 $c$ の数を返す．$select_c(S,i)$ は $S$ の $i$ 番目の $c$ の位置を返す．末尾にデータの挿入を許すためポインタ付きの WT[10] を用いて，末尾へのデータの挿入，$access/rank/select$ はそれぞれ $O(\lg \sigma)$ 時間 ($\sigma$ は文字種類数) で実行され，その領域は，$|S| \lg \sigma + o(|S| \lg \sigma) + O(\sigma \lg \sigma)$ で実装される．

## 3 簡易版 ESP-index

このセクションでは edit-sensitive parsing (ESP)[3] に基づく文法圧縮 [13]($GC\text{-}ESP$) と検索手法 (ESP-index) について述べる．

### 3.1 簡易版 edit-sensitive parsing

GC-ESP の基本的な考え方は以下の通りである．(i) 入力文字列 $S \in \Sigma^*$ が与えられて，(ii) 何度も出現する部分文字列を出来る限り同じ変数列によって置き換え，(iii) 置き換えられた文字列を入力文字列 $S$ として (i) から (iii) を入力文字列長が 1 になるまで繰り返すことで構文木をボトムアップに構築する．各繰り返しにおいて，GC-ESP は $S$ を $S = S_1S_2 \cdots S_\ell$ のような重複しない次の 3 タイプの部分文字列に分割する : (1) 文字の繰り返し;(2) 文字の繰り返しを含まない $\lceil \lg |S| \rceil$ より長い文字列; (3)(1) と (2) 以外．

GC-ESP では各繰り返しで分割された $S_i$ をさらに digram($XY$) か trigram($XYZ$) に分割し，生成規則の $Z \to XY$ を表す 2 木もしくは $W \to XV, V \to YZ$ を表す 2-2 木を構築する．このさらなる分割の方法は $S_i$ の種類によってことなる．$S_i$ が (1) と (3) の場合は以下のように左優先で 2 木を構築し，長さが偶数の場合はすべて 2 木で構築し，奇数の場合も同様に左優先で 2 木を構築し，最後の 3 文字だけ 2-2 木で構築する．$S_i$ が (2) の場合は alphabet reduction によって，$S_i = s_1s_2 \cdots s_\ell (2 \le |s_j| \le 3)$ のようにさらに分割し，長さが 2 のときは 2 木を構築し，長さが 3 のときは 2-2 木を構築する．分割され置き換えられた $S$ を新たな入力文字列 $S'$ として，この分割および木の構築を繰り返す．

**Alphabet reduction :** alphabet reduction は $S_i$ が (2) のとき $S_i$ を digram もしくは trigram に分割する．$S_i$ の各文字 $S_i[j]$ を二進数とみなす．$S_i[j]$ と $S_i[j-1]$ が初めて異なる最下位ビットを $p$ とし，$bit(p, S_i[j]) \in \{0,1\}$ を $S_i[j]$ の下位 $p$ 番目のビットとする．このとき，$L[j] = 2p + bit(p, S_i[j])$ を計算する．$L[j]$ を $S_i$ が (2) のときのすべての位置 $j$ に対して計算し，$L = L[1]L[2] \cdots L[|S_i|]$ を得る．$S$ は繰り返しを含まないので $L$ もまた type2 である．$L[j]$ が極大値（$L[j] > max\{L[j-1]L[j+1]\}$ ）のとき，もしくは $L[j]$ の隣に極大値がなく，$L[j]$ が極小値（$L[j] < min\{L[j-1]L[j+1]\}$) のとき，$S[j]$ を $landmark$ と呼ぶ．

$L$ は（2）であり，$L$ の文字種類数は高々 $\lg |S|$ であるので，$S_i$ が (2) であるとき，その長さは $\lg |S|$ 以上であるので，必ず一つは landmark を含む．すべての landmark が決められたところで $S_i[j]$ が landmark のとき，$S_i[j]S_i[j-1]$ を 2 木として $Z \to S_i[j]S_i[j-1]$ を生成する．それ以外の部分は，$S_i$ が (1) と (3) のときと同様に左優先で 2 木もしくは 2-2 木を構築する．

**補題 3.** $S = x\beta y\beta z$($x$, $y$, $z$ は $S$ の ESP のタイプ (1) から (3) のいずれかの部分文字列，$\beta(|\beta| \ge 2 \lg |S|)$ は $S$ のタイプ 2 の文字列) であるとき，$S$ 上の 2 つの $\beta$ における $\beta[\lg |S|, |\beta| - \lg |S|]$ の ESP による分割は一致する．

補題 3 における $\beta[k_1, k_2]$ を置き換えた文字列 $\beta[k_1, k_2]'$ が $2 \lg |S|$ 以上であれば，この分割の一致は $\beta[k_1, k_2]'$ において次のレベルの ESP の繰り返しでも継続される．

### 3.2 検索アルゴリズム

この章では簡易版 ESP-index の検索方法について述べる．パターン $P$ と入力テキスト $S$ が与えられたとき，$S$ 上の $P$ を探す問題である．

**定義 4.** *(Evidence)[7]* ESP によって構文木を構築されたパターン $P$ と入力テキスト $S$ の $P$ において，$P$ を符号化している非終端記号列を evidence と呼ぶ．

**定理 5.** 入力テキスト $S$ とパターン $P$ を辞書を共有して，ESP を用いた構文木を構築したとき，$P$ と $S$ 上の $P$ は $P$ より短い $O(\lg |S| \lg |P|)$ 個の evidence $Q$ によって符号化されている．

### 3.3 検索のための evidence の抽出

まず，$P$ を ESP を用いて，各タイプ (1) から (3) の部分文字列に分割する．先頭がタイプ（2）の場合は最初の landmark までの文字列 $q_x$ を $Q_1$ の末尾に追加する．末尾がタイプ（2）の場合は最後の landmark 以降の部分文字列 $q_y$ を $Q_2$ の先頭に追加する．先頭（末尾）がタイプ (3) の場合は先頭（末尾）の部分文字列 $q_x(q_y)$ を $Q_1(Q_2)$ の末尾 (先頭) に追加する．先頭がタイプ (1) の場合はそのタイプ (1) を葉に持つような完全二分木の非終端記号のリスト $l_x$ を $Q_1$ の末尾に追加．末尾がタイプ (1) の場合も同様にリスト $l_y$ を $Q_2$ の先頭に追加する．分割が一致した部分を $T_S$ を構築したときの逆引き辞書を用いて置き換えて，次のレベルでも同様の操作を繰り返し，分割の一致がなくなるまでこの操作を繰り返す．最後に $Q_1$ と $Q_2$ を連結して，evidence $Q = Q_1Q_2$ とする．

### 3.4 location, counting, extraction

**counting** ： evidence の中で，最も長い文字列を符号化している非終端記号 $x$ を抽出する．$x$ に対応する $T_S$ の構文木のノード $v$ にアクセスする．$v$ から構文木上の top-down 検索を用いて，他の evidence を探す．$v$ の親ノード ($parent(v)$) に移動し，親ノードに移動する際移動した枝が左の枝だった場合は，$parent(v)$ の右の子の最左子孫に $x$ の右隣りの evidence があるかどうか確認する．親ノードに移動する際移動した枝が右の枝だった場合は，逆に $parent(v)$ の左の子の最右子孫に $x$ の左隣りの evidence があるかどうか確認する．隣接する evidence が見つかった場合はこの操作を繰り返しすべての evidence を探す．この操作を構文木上のすべての $v$ に対して行う．この時すべての evidece を発見できた場合は，移動したノードの中で最も構文木上で高い位置にあるノード $v_a$ はパターンを符号化しているため，構文木上の $v_a$ をすべて数えあげえることで，パターンの出現回数をカウントすることができる．

**locating** ： 準備として，構文木の各ノードに対してそのノードが符号化している文字列の長さ（展開長）を保存しておく．次に counting と同様にして $v_a$ を発見する．$v_a$ から $T_S$ のルートへと移動する．このとき右の枝を通って，親ノードに移動するときは移動した親ノードの左の子の展開長保存しておき，その保存した左の子の展開長をすべて足し合わせることでパターンの出現位置を発見できる．これをすべての $v_a$ に対して行う．

**extraction:** locating と同様に各ノードの展開長を保存しておく．$T_S$ のルートから指定されたポジションが左の子の展開長とこれまで辿ってきた右の子と逆の左の子の展開長を足し合わせたものより大きい場合は右の子へそうでなければ左の子へ降りていき，先頭位置を発見したら，そこから指定した位置まで文字列を復元する．

したがって，ESP-index では evidence を発見するための逆引き辞書，検索のための各ノードへのアクセスと親子間のノードの移動が必要である．次のセクションではオンラインかつ圧縮領域で構文木を構築しながら，これらの操作をサポートする構文木の符号化を提案する．

## 4 Online ESP-index

### 4.1 Post-order SLP

Fully online LCA(FOLCA)[8] の技術を用いて，Online ESP-index(OESP-index) では同様に post-order SLP(POSLP) を構築し，検索可能なように直接符号化していく．Rytter[12] によって定義された構文解析木では深さ優先順で構文木を巡回し，二回以上出現する非終端記号以下の子孫が枝刈りされた順序木である．

**定義 6. (POSLP と POPPT)**[8] post-order partial parse tree(POPPT) は内部ノードが後置順で名前付けされた構文解析木である．post-order SLP(POSLP) は構文解析木が POPPT である SLP である．

$n$ 個の生成規則を持つ POSLP の POPPT 上でのノードの数は $2n+1$ 個（内部ノード $n$ 個，葉ノード $n+1$ 個）である．

FOLCA はオンラインで POSLP を構築する．このとき，自明な辞書を構築することなく，サイズが $n\lg(n+\sigma)+2n+o(n)$bits の構文木の簡潔表現を入力テキストから直接構築するため、圧縮領域で動作させることが可能である．また，コンパクトな逆引き辞書も提案している．OESP-index では，FOLCA における辞書の簡潔表現の葉のラベルの配列を Wavelet treeNavarro2012-cpm で符号化することで検索を可能とする．そのサイズは $2n + n\lg(n+\sigma) + O(n\lg(n+\sigma))$ である．

### 4.2 ESP-index のための辞書の符号化

POSLP を bit 列 $B$ と葉のラベルの配列 $L$ によって符号化する．後置順に POSLP の POPPT を辿るこのとき $B$ は葉ノードならば 0 内部ノードなら 1 をならべていき，最後に 1 をならべる．このとき $B$ は dynamic range min/max tree[11] で符号化する．構文木上の移動およびアクセスを $O(\lg n/\lg\lg n)$ 時間でサポートし，内部ノードの数 は $n$ 個なので $B$ のサイズは $2n+o(n)$bits である．

$L$ も後置順に葉のラベルをならべていく．$L$ は Wavelet tree [10] に符号化する．葉のラベルへのアクセスを $O(lgn)$ でサポートし，葉ノードの数は $n+1$，文字種類数は $n+\sigma$ なので，そのサイズは $O(n \lg n)$ で表現できる．

このときさらに後置順に内部ノードの展開長を並べた配列 $P$ で保持する．そのサイズは $n \lg N$ であり，$i$ 番目のノードへは $P[rank_0(B,i)]$ で各展開長へ $O(\lg n/\lg\lg n)$ 時間でアクセス可能である．

したがって，検索のための POSLP のサイズは以下の補題のようになる．

**補題 7.** 生成規則 $n$ 個の POSLP のサイズは，$n \lg N + O(n \lg n)$ であり，$O(\lg n)$ 時間で生成規則の右側へのアクセス，親への移動，ノード $i$ が $k$ 番目に出現するノードへの移動をサポートする．

### 4.3 逆引き辞書の動的圧縮法

FOLCA の逆引き辞書の圧縮法 [8] を用いる．そのサイズは，$\alpha n \lg(n+\sigma) + n(1+\lg(\alpha n))$ で $O(1/\alpha)$ 期待時間で一回のアクセスが可能である．

したがって，全体のメモリ使用量は以下の定理のようになる．

**定理 8.** 入力長が $N$ で，そこから生成される POSLP の生成規則数が $n$ のとき，辞書と逆引き辞書は $O(\frac{N \lg n}{\alpha})$ 期待時間で計算される．そのときの全体のメモリ使用量は，$n \lg N + O(n \lg n)$ bits である．

これにより簡易版 ESP-index の要件を満たすので，locating, countinig, extracting をそれぞれ以下のような計算時間でサポートする．

**定理 9.** 入力長 $N$，POSLP の生成規則数 $n$，パターン長 $m$，$occ_c$ は evidence の中で展開長が一番長いものとすると，locating と counting は $O(\lg n(\frac{m}{\alpha} + occ_c \lg m(\lg N)^2))$ 時間でサポートし，extracting は $O(\lg n(m + \lg N))$ 時間でサポートされる．couting の領域には辞書と逆引き辞書だけの $O(n \lg n)$ だけ必要であり，locating と extracting には加えて，$n \lg N$ の領域が必要である．

### 4.4 索引のオンライン構築

FOLCA のオンライン文法圧縮法と同じ方法 [8] で圧縮領域で動作する文法圧縮型オンライン構築を提案する．FOLCA における文法圧縮法は高さ $\lg N$，生成規則の数を $O(n^* \lg^2 N)$ 近似する．landmark かタイプ (1) に属するペアかを判定して，圧縮していく．

まず $k$ 個の queue$q_1, q_2, cdots q_k (k \geq \lg N)$ を用意する．各キューの長さは 5 に固定する．各キューは構文木のそれぞれの深さのために存在する．すべての $q_x[1], q_x[2] (1 \leq x \leq k)$ にはダミーコード $d$ をいれておく．$q_1$ に入力テキストを enqueue し，$q.size() = 4$ のときに $q_1[3]$ と $q_1[4]$ が同じ文字もしくは，$q_1[3]$ が landmark の場合は 2 木（$Z \rightarrow q_1[3]q_1[4]$）をつくり，$Z$ を $q_2$ に enqueue し，$q_1$ の先頭 2 文字 dequeue する．$q_1[3]$ と $q_1[4]$ が連続文字でもなく，$q_1[3]$ が landmark でもない場合はもう一文字 $q_1$ に enqueue し，2-2 木 ($W \rightarrow q_1[4]q_1[5], V \rightarrow q_1[3]W$) を作り，$q_2$ に $W$ を enqueue し，$q_1$ の先頭 3 文字を dequeue する．各深さでこれを繰り返し，入力テキストを導出する構文木を構築する．

**定理 10.** FOLCA によって生成される生成規則に対応する構文木は POPPT である．[8]

定理 10 により，入力テキスト $S \in \Sigma^*$ を導出する POPPT $T$ と等価な辞書を構築できる．$T$ のために本論文で示した索引構造のための符号化された辞書を用いることで，末尾に入力テキスト $S$ の末尾に新たな文字 $a \in \Sigma$ が追加されたときすなわち $Sa$ となったときも $O(\lg n)$ 時間で POPPT を再計算することができる．したがって，オンラインで文法圧縮に基づく自己索引を圧縮領域で構築することが可能である．

## 5 実験

実験にはメモリ 16GB の Intel(R) Core(TM) i7-2620M CPU(2.7GHz) のマシンを用いた．実験データとして `http://pizzachili.dcc.uchile.cl/repcorpus.html` より，英文データ einstein.en.txt(446MB)(einstein) を用いた．索引構築にかかる時間 (図 1) と全体のメモリ使用量（図 2）に関して実験を行った．

図 1 における索引構築にかかった全体の時間は 2415 秒であった．

図 2 における最終的なメモリ使用量は 22.79MB であった．圧縮領域で動作することを確認した．最終的なメモリ使用量の内訳を表 3 に示す．$D$ のメモリ使用量が最も多かった．

メモリ消費が一番多い $D$ は Wavelet tree で表現した $L$ のメモリ使用量が多いからである．これはポインタ付きの Wavelet tree を使用しているため，ポインタや予約済みの領域によりこの様にメモリ使用量が大きくなってしまっている．

表 3: 全体のメモリ使用量の詳細.

| | $D$(MB) | $P$(MB) | $H$(MB) |
|---|---|---|---|
| einstein | 15. 28 | 1.34 | 6.16 |

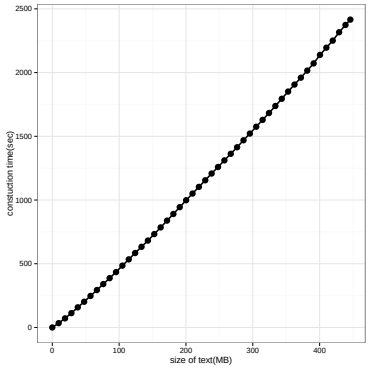


図 1: 索引構築時間 (einstein)

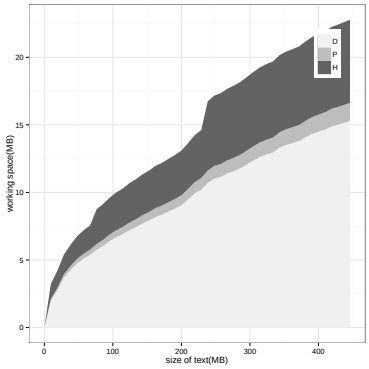


図 2: 全体のメモリ使用量(einstein). D，P，H はそれぞれ D:構文木の圧縮表現，P: 位置情報，H : ハッシュテーブルのメモリ使用量である.

## 6 まとめ

オンラインで構築可能な文法圧縮型の自己索引である OESP-index を提案した.

## 参考文献

[1] F. Claude and G. Navarro. Improved grammar-based compressed indexes. arXiv:1110.4493.

[2] F. Claude and G. Navarro. Self-indexed grammar-based compression. *Fundam. Inform.*, Vol. 111, No. 3, pp. 313–337, 2011.

[3] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Trans. Algor.*, Vol. 3, No. 1, p. Article 2, 2007.

[4] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, and S.J. Puglisi. A faster grammar-based self-index. In *LATA*, pp. 240–251, 2012.

[5] M. Karpinski, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nordic J. Comp.*, Vol. 4, No. 2, pp. 172–186, 1997.

[6] S. Maruyama, M. Nakahara, N. Kishiue, and H. Sakamoto. ESP-Index: A Compressed Index Based on Edit-Sensitive Parsing. *Journal of Discrete Algorithms*, Vol. 18, pp. 100–112, 2013.

[7] S. Maruyama, H. Sakamoto, and M. Takeda. An online algorithm for lightweight grammar-based compression. *Algorithms*, Vol. 5, No. 2, pp. 213–235, 2012.

[8] S. Maruyama, Y. Tabei, H. Sakamoto, and K. Sadakane. Fully-online grammar compression. In *SPIRE2013*, pp. 218–229, 2013.

[9] G. Navarro. Indexing text using the Ziv-Lempel tire. *J. Discrete Algorithms*, Vol. 2, No. 1, pp. 87–114, 2004.

[10] G. Navarro. Wavelet trees for all. In *CPM*, pp. 2–26, 2012.

[11] G. Navarro and K. Sadakane. Fully-functional static and dynamic succinct trees. *ACM Transactions on Algorithms*, 2012. Accepted. A preliminary version appeared in SODA 2010.

[12] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, Vol. 302, No. 1-3, pp. 211–222, 2003.

[13] H. Sakamoto, S. Maruyama, T. Kida, and S. Shimozono. A space-saving approximation algorithm for grammar-based compression. *IEICE trans. inf. syst.*, Vol. 92, No. 2, pp. 158–165, 2009.

[14] Y. Takabatake, Y. Tabei, and H. Sakamoto. Improved esp-index: A practical self-index for highly repetitive texts. In *SEA2014*, pp. 338–350, 2014.