文法圧縮における逆引き辞書の省スペース化

Space-Efficient Reverse Dictionary for Grammar Compression

福永祥平[1*]　坂本比呂志[1]
Shouhei Fukunaga[1]　Hiroshi Sakamoto[1]

[1] 九州工業大学情報工学部
[1] Kyushu Institute of Tecnology School of Computer Science and Systems Engineering

Abstract: We propose an efficient method for reducing the memory space of grammar compression. The memory space is mainly occupied by the dictionary and its reverse function called reverse dictionary. Usually, the reverse dictionary is implemented by a hash table. We apply a new data structure called DRMTRD to the reverse dictionary and reduce the hash talble.

1 はじめに

本研究では，動的一次元領域探索を用いて省作業領域で構築可能な文法圧縮のための逆引き辞書 [3] を実装し，従来のハッシュテーブルを用いた逆引き辞書との比較を行った．文法圧縮とは，データの共通部分を文脈自由文法 (CFG) の生成規則として表現することにより，データの冗長性を取り除いて圧縮する方法である．現在ではスマートフォンやソーシャルネットワークサービスなどの普及によりビッグデータ化が進み，そしてビッグデータの活用が求められている．しかし，ビッグデータをそのまま扱うと非常に大きなコストが必要となる．そこで文法圧縮を用いることで，より低コストでビッグデータを扱うことが可能となる．

文法圧縮の構築において，辞書と逆引き辞書という 2 つのデータ構造が必要であり，辞書とは X_i が与えられて，それに対応する生成規則 $X_i \to X_jX_k$ の右辺 X_jX_k へのアクセスをサポートするデータ構造であり，CFG の標準形の 1 つである *Straight-Line Program*(SLP) の情報理論的下限と漸近的に一致するデータ構造が Fully-online LCA(FOLCA)[1] で提案されている．一方で，逆引き辞書は X_jX_k が与えられて，生成規則 $X_i \to X_jX_k$ が構築中の SLP に存在すれば X_i を返すデータ構造であり (RE-PAIR[2] において文字ペアの頻度表のためのハッシュテーブルがこれに相当する)，こちらは作業領域が大きくなる原因となっている．

高畠らが提案した手法 Dynamic Range Max Tree for Reverse Dictionary(DRMTRD)[3] は一次元領域探索を用い，省作業領域の逆引き辞書の構築を可能としている．この手法は逆引き辞書を $n+n\lg n-o(n)$ ビットで構築し，逆引き辞書の探索 $D^{-1}(X_jX_k)$ を $O(\frac{\lg^2 n}{\lg\lg n})$ 時間で可能とするが，実データの実験では明確にされていない．そこで本研究では，DRMTRD を実装し，静的な逆引き辞書の省スペース化を実現し，従来のハッシュテーブルを用いた逆引き辞書のメモリ使用量と逆引き辞書の探索 $D^{-1}(X_jX_k)$ の比較実験を行った．

以下，2 章でこの論文で使う基本概念ついて説明し，3 章で動的一次元領域探索を用いた逆引き辞書 DRMTRD について説明し，4 章で実験について説明し，5 章でまとめと今後の課題を述べる．2 章と 3 章の DRMTRD に必要な説明は動的一次元領域探索を用いた省作業領域で構築可能な文法圧縮のための逆引き辞書 [3] から引用した．

*連絡先：九州工業大学情報工学部知能情報工学科
〒 820-0067 福岡県飯塚市川津 680-4
E-mail: s_fukunaga@donald.ai.kyutech.ac.jp

2 準備

2.1 基本概念

集合 C の要素数は $|C|$ とする．Σ を有限アルファベットの集合とし，$\sigma = |\Sigma|$ とする．入力文字列は S とし，$N = |S|$ とする．文字列 R の長さを $|R|$ と表記する．変数の帰納的可算集合 χ は $\Sigma \cap \chi = \emptyset$ とする．$\lg = \log 2$ である．

2.2 文法圧縮

文法圧縮とは CFG を構築することによって圧縮する方法である．CFG の標準形の 1 つである SLP は SLP $G = \{V,\Sigma,P,X_s\}$ で表される．ここで，$V \subseteq \chi$，$P = \subseteq (V \times (\Sigma \cup V)^2)$，$X_s \in V$ は SLP の開始記号であり，生成規則の形は $X_i \to X_jX_k (X_i \in V, i < j, k)$ に制限される．$n = |P|$ とし，各 X_i は非終端記号と呼ぶ．

2.3 辞書と逆引き辞書

文法圧縮を行うためには一般的に辞書 D を逆引き辞書 D^{-1} の 2 つのデータ構造が必要である．

文法圧縮の P を表現するデータ構造を辞書 D といい，関数 $D(X_i)$ は $X_i \to X_jX_k \in P$ のとき X_jX_k を返し，それが以外のときは $X_\infty X_\infty (X_\infty \notin V)$ を返す．生成規則数を n とすると，一般的に D は $2n \lg n$ ビットの二重配列で表現され，関数 $D(X_i)$ を $O(1)$ で返す．データの追加，削除が行われない静的な辞書 D に関しては $Improved\ ESP\text{-}index(ESP\text{-}index\text{-}I)$[5] において，領域が $2n + n \lg n + o(n \lg n)$ ビットという SLP の表現長の情報理論的下限である $2n + \lg n! + o(n)$ に近いサイズで関数 $D(X_i)$ を $O(\lg \lg n)$ で返す方法が提案されている．データの追加，削除が行われる動的な場合に関しては FOLCA で辞書 D を文字種類数を σ をとすると $2n + n \lg(n + \sigma) + o(n)$ ビットという SLP の表現長の情報理論的下限に漸近的に一致するサイズで $O(\frac{\lg n}{\lg \lg n})$ 時間で末尾追加とアクセスする方法が提案されている．

逆引き辞書 D^{-1} は X_jX_k が与えられて，$X_i \to X_jX_k \in P$ のとき X_i を返すデータ構造である．この逆引き辞書の関数は $D^{-1}(X_jX_k)$ と表現する．逆引き辞書の一般的なデータ構造はチェイン法によるハッシュテーブルであり，loadfactor を $\alpha \in (0, 1]$ とすると，$n(1+\alpha) \lg(n+\sigma)$ ビットで表現でき， $D^{-1}(X_jX_k)$ を $O(\frac{1}{\alpha})$ で返す．静的な場合には ESP-index-I で，辞書 D と同じデータ構造を用いて，$D^{-1}(X_jX_k)$ を $O(\lg \lg n)$ で返す方法が提案されている．FOLCA ではハッシュテーブルをリストに登録される値の単調増加性を用いて圧縮しており，その領域は $\alpha n \lg(n + \sigma) + n(1 + \lg \alpha n)$ ビットであり，$D^{-1}(X_jX_k)$ を $O(\frac{1}{\alpha})$ 期待時間で返す．

逆引き辞書は辞書と比べ作業領域が大きく，本実験では FOLCA で辞書を作成した後、その辞書を利用して DRMTRD を実装し，逆引き辞書の省スペース化を行った．

2.4 一次元領域探索

DRMTRD は逆引き辞書を動的一次元探索で表現しているので，一元領域探索について述べる．一次元領域探索 $Is(Q, x_i, x_j)$ と集合 $M = \{0, 1, 2, \cdots, m-1\}$, $Q \subseteq M$ と区間 $[x_i, x_j](1 < x_i < x_j < m)$ が与えられて，区間 $[x_i, x_j]$ にある Q の集合に属する要素を返す．動的一次元領域探索とは $Is(Q, x_i, x_j)$ を行えるようにしつつ Q への新たな正の整数の追加および削除が可能である．

新たなデータの追加が行われない静的な場合の一般的な手法では Q の要素をソートし，二分探索で区間 $[x_i, x_j]$ の要素を探す場合，解の個数を k 個とすると $|Q| \lg m$ ビットでの領域と $O(\lg |Q| + k)$ 時間で $Is(Q, x_i, x_j)$ を行うことが可能である．

動的な場合の一般的な手法では平衡二分木を用いて，$|Q| \lg m + 2|Q| \lg |Q|$ ビットの領域と $O(\lg |Q| + k)$ 時間で構築することが可能で，これの挿入 (insert)· 削除 (delete) は最悪 $O(\lg |Q|)$ 時間である．

したがって，辞書 D があるときに，生成規則 $X_i \to X_jX_k$ を X_jX_k の辞書式順序で昇順 ($X_{j_1}X_{k_1} > X_{j_2}X_{k_2}$ とは $X_{j_1} > X_{j_1}$ または $X_{j_1} = X_{j_1}$ のとき $X_{j_2} > X_{k_2}$ であること) で一次元領域探索を行えるようにしておけば，$D^{-1}(X_jX_k) = Is(D, X_jX_k, X_jX_k)$ とすることにより逆引き辞書を実装可能であるが，平衡二分木のような一般的な手法で行うと既存のハッシュテーブルを用いた逆引き辞書よりも多くの領域が必要となってしまう．そこで DRMTRD では動的区間最大最小木とほぼ同じデータ構造を用いることでハッシュテーブルを用いた逆引き辞書よりも省領域で実装を可能にしつつ，高速に $D^{-1}(X_jX_k) = Is(D, X_jX_k, X_jX_k)$ を計算可能としている．

2.5 Fully-online LCA

本論文で行った実験では辞書の作成に Fully-online LCA(FOLCA)[1] を利用しているので，FOLCA について説明する．FOLCA はオンラインで SLP の情報理論的下限に漸近的に一致するサイズの辞書 D を構築する．その辞書は succinct post order SLP(SPOSLP) と呼ばれる．post order SLP(POSLP) とは SLP を開始記号の根とする構文木を見たとき，各非終端記号を後置順にラベリングし，一度出現した内部ノードはその子孫を枝刈りした SLP 表現である．SPOSLP は POSLP を後置順に辿り，葉ノードなら 0，内部ノードなら 1 を並べたビット列ビット配列 B と POSLP の葉ノードのラベルを後置順に並べた配列 L を用意する．B は動的区間最大最小木 [4] によって符号化されている．符号化された B と L を用いると様々なな操作が可能となり [1]，$D(X_i)$ は $O(\frac{\lg n}{\lg \lg n})$ 時間で実行できる．

3 DRMTRD

この章では，DRMTRD のデータ構造を述べ，逆引き辞書の探索 $D^{-1}(X_jX_k)$ のアルゴリズムについて述べる．

3.1 DRMTRD のデータ構造

DRMTRD は動的区間最大最小木 (DRmMT)[4] のデータ構造とほぼ同じデータ構造の動的区間最大木 (DRMT)[3] により実現する．DRmMT ではビット列

を扱っていたが，提案手法では生成規則を辞書式順序で昇順に並び替えた列を扱っている．

辞書 D が与えられていると仮定する．

1. 各生成規則の左辺 $X_i(i \in [1, n])$ を生成規則の右辺 X_jX_k の辞書式順序昇順で並び替えた X_{sort} を準備する．
2. X_{sort} を長さ $3\lg n \leq L < 6\lg n$ の k 個のブロック列 $Div = Div_1, Div_2, \cdots, Div_k$ に分割する．
3. Div_{2i-1}，Div_{2i} を子とする内部ノード In_i からなる列 $In = In_1, In_2, \cdots, In_{k/2}$ を構築する．
4. できるノードが 1 つになるまで，1 つ前の操作でできた内部ノードの列に対しても同様にノードの構築を繰り返す．

各内部ノードは子孫のもつ生成規則の右辺 X_jX_k が辞書順で最大の生成規則の左辺 X_{max} と左右の子へのポインタを格納し，葉のブロックにはそれぞれが保持する列の長さ $|Div_i|(1 < i \leq k)$ を保持する．

3.2 DRMTRD の領域計算量

定理 3.1. *DRMTRD* のサイズは $n + n\lg n - o(n)$ ビットある．

証明 1. 葉と内部ノードそれぞれが持っているデータ構造のサイズを確認し，全体のサイズを求める．葉ノードは生成規則の左辺 $X_i(i \in [1, n])$ からなる列とその列の長さ $|Div_i|(1 < i \leq k)$ を持っている．生成規則の左辺は *1* つあたり $\lg n$ ビットであり，生成規則は n 個あるので，$n\lg n$ ビットである．各葉ノードの $|Div_i|$ は *1* つあたり $\lg(6\lg n) = \lg\lg n + \lg 6$ ビットであり，葉ノードの数は $\frac{n}{3\lg n}$ なので，$(\lg\lg n + \lg 6) \times \frac{n}{3\lg n} = \frac{n(\lg\lg n + \lg 6)}{3\lg n}$ ビットである．したがって，葉ノードの合計サイズは，$n\lg n + \frac{n(\lg\lg n + \lg 6)}{3\lg n}$ ビットである．内部ノードはサイズ $\lg\frac{2n}{3\lg n} = \lg n - \lg\lg n + 1 - \lg 3$ ビットのポインタが *2* つあり，$\lg n$ ビットの X_{max} を持っているので内部ノード一つあたりのサイズは $3\lg n - 2\lg\lg n + 2 - 2\lg 3$ である．内部ノードの数は $\frac{n}{3\lg n}$ なので，内部ノードの合計サイズは $(3\lg n - 2\lg\lg n + 2 - 2\lg 3) \times \frac{n}{3\lg n} = n + \frac{2n(1 - \lg\lg n - \lg 3)}{3\lg n}$ ビットとなる．したがって，このデータ構造の合計サイズは $n\lg n + \frac{n(\lg\lg n + \lg 6)}{3\lg n} + n + \frac{2n(1 - \lg\lg n - \lg 3)}{3\lg n} = n + n\lg n + \frac{n(2 + \lg 6 - \lg\lg n - 2\lg 3)}{3\lg n} = n + n\lg n - o(n)$ ビットである． □

3.3 逆引き辞書の探索 $D^{-1}(X_jX_k)$ のアルゴリズム

$D^{-1}(X_jX_k)$ は根から X_jX_k が存在する可能性のある葉ノードへと辿り，その葉の中を二分探索する．

1. X_jX_k が辞書順で各ノードで左の子の $D(X_{max}) = X_lX_r$ より大きい場合は右の子へ，それ以外は左の子へ移動する．
2. 葉ノードに辿り着くまで，1 の操作を繰り返す．
3. 葉ノードの中を二分探索する．

3.4 逆引き辞書の探索 $D^{-1}(X_jX_k)$ の時間計算量

定理 3.2. $D(X_i)$ の計算時間を $O(d)$ とすると *DRMTRD* における $D^{-1}(X_jX_k)$ を $O(d\lg n)$ 時間，新しい生成規則の登録 $insert(X_i)$ を $O(d\lg n)$ 時間で実行可能である．

証明 2. $D^{-1}(X_jX_k)$ の時間計算量は木の根から葉への探索と葉ノードに格納されている生成規則の左辺 $X_i(1 < i < k)$ からなる列の二分探索の合計である．*1* 回の辞書の探索 $D(X_i)$ の時間は $O(d)$ 時間であり，木の高さは $\lg n$ であるので，木での探索は $O(d\lg n)$ 時間である．葉ノードが保持する列の長さは $O(\lg n)$ であるので，葉ノード内の二分探索は $O(\lg\lg n)$ 時間である．したがって，$D^{-1}(X_jX_k)$ の時間計算量は $O(d\lg n) + O(\lg\lg n) = O(d\lg n)$ 時間である． □

今回行った実験では辞書の構築に FOLCA[1] を利用したので，$O(d) = O(\frac{\lg n}{\lg\lg n})$ 時間である．したがって $D^{-1}(X_jX_k)$ の計算時間量は $O(\frac{\lg^2 n}{\lg\lg n})$ 時間となる．

4 実験

実験には，Pizza& Chili Corpus [1] で公開されているテキストデータを用いた．

入力データは表 1 に示す圧縮率が良く文字種類数が多いテキスト einstein，圧縮率が良く文字種類数が少ないテキスト cere，圧縮率が悪く文字種類数が多いテキスト SOURCES，圧縮率が悪く文字種類数が少ないテキスト PROTEINS の 4 種類のテキストデータ利用した．ここで，圧縮率が良いとは圧縮率が 5% 未満であり，悪いとは 45% 以上のものをいう．また文字種類数 σ が多いとは $\sigma \geq 139$ であり，少ないとは $\sigma \leq 27$ をいう．

環境は，OS:CentOS 5.11，Memory:144GB RAM，CPU:Intel Xeon E5504(2.00GHz，Quad Core)，Compiler:gcc4.1.2 である．

辞書の作成には FOLCA を利用し，オフラインで DRMTRD を実装した．また，X_{sort} を $5\lg n$ の固定長でブロック列に分割した．

[1] http://pizzachili.dcc.uchile.cl/texts.html

表 1: テキストデータ

テキスト	einstein	cere	SOURCES	PROTEINS
データサイズ (MB)	446	446	200	200
圧縮率 (%)	0.20	3.58	45.31	58.59
文字種類数	139	5	230	27
生成規則数	376,372	5,615,076	29,396,483	37,504,238

DRMTRD のメモリ使用量と $D^{-1}(X_jX_k)$ の 10 万回の実行時間を計測し，ハッシュテーブルを用いた逆引き辞書との性能比較を行った．ただし，$X_jX_k \subseteq (\Sigma \cup V)^2$ を 1 回ずつランダムに生成した．逆引き辞書のメモリ使用量を図 1，$D^{-1}(X_jX_k)$ の 10 万回実行時間を図 2 に示す．従来手法を original，DRMTRD を drmtrd とし，横軸は生成規則数である．

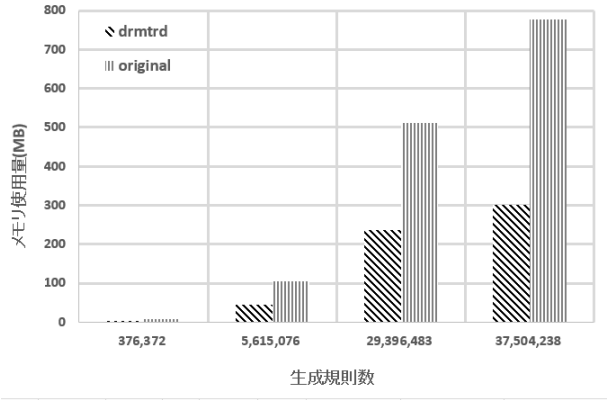

図 1: 逆引き辞書のメモリ使用量

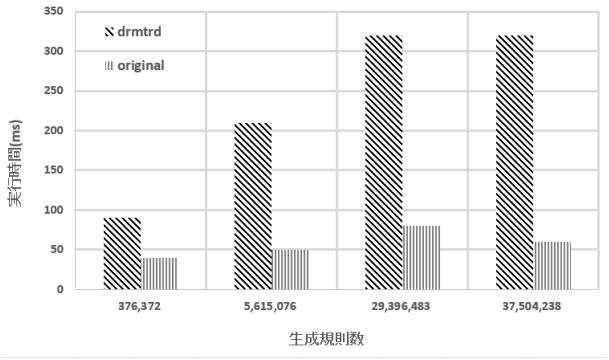

図 2: $D^{-1}(X_jX_k)$ の 10 万回の実行時間

図 1 から従来手法と比べると，50% ~ 60% のメモリの削減に成功していることが確認できる．よって生成規則数が多いデータほどより多くのメモリが削減できるといえる．図 2 から，従来手法は $O(\frac{1}{\alpha})$ 期待時間であり，loadfactor α は 0.5 で固定しているのでほぼ一定時間で $D^{-1}(X_jX_k)$ が可能であったが，DRMTRD では $O(\frac{\lg^2 n}{\lg\lg n})$ なので生成規則数が 30,000,000 までは実行時間の差が広がっているが，それ以降は実行時間の差は広がっていかないと考えられる．

5 まとめと今後の課題

本研究では，FOLCA で構築した辞書に対して一次元領域探索を用いた逆引き辞書 DRMTRD を実装して実験を行った．実験結果より，DRMTRD のメモリ使用量は生成規則数に依存しているため，テキストデータが大きく，圧縮率が悪いデータほどメモリの削減スペースは大きくなることがわかる．したがって逆引き辞書の省スペース化は達成できているといえる．一方で逆引き辞書の探索 $D^{-1}(X_jX_k)$ は理論通り生成規則数に依存しており，従来手法と比較すると実行時間は悪化していることが確認できた。

今後の課題として，今回の研究では生成規則の追加には対応していないので，$Insert(X_i)$ の機能を加えた逆引き辞書の実装や，$D^{-1}(X_jX_k)$ は従来手法と比較すると実行時間は悪化しているので $D^{-1}(X_jX_k)$ の高速化が挙げられる．

$D^{-1}(X_jX_k)$ の高速化の方法の一つとして，DRMTRD の内部ノードが保持する X_{max} を X_{max} の生成規則の右辺 X_lX_r に変える方法がある．

定理 5.1. X_{sort} を長さ $4\lg n \le L' < 8\lg n$ で分割することにより領域計算量を維持したまま $D^{-1}(X_jX_k)$ を $O(\lg n)$ 時間で行うことができる．

証明 3. 生成規則の左辺は *1* つあたり $\lg n$ ビットであり，生成規則は n 個あるので，$n\lg n$ ビットである．X_{sort} を長さ $4\lg n \le L' < 8\lg n$ で分割することにより各葉ノードの $|Div|$ は一つあたり $\lg(8\lg n) = \lg\lg n + 3$ ビットになり，葉ノードの数は $\frac{n}{4\lg n}$ となるので，$|Div_i|$ のすべての合計サイズは $(\lg\lg n + 3) \times \frac{n}{4\lg n} = \frac{n(\lg\lg n+3)}{4\lg n}$ ビットである．したがって，葉ノードの合計のサイズは $n\lg n + \frac{n(\lg\lg n+3)}{4\lg n}$ ビットである．内部ノードはサイズ $\lg\frac{2n}{4\lg n} = \lg n - \lg\lg n - 1$ ビットのポインタが *2* つとなり，$2\lg n$ ビットの X_lX_r を持っているので内部ノード一つあたりのサイズは $4\lg n - 2\lg\lg n - 2$ ビットである．内部ノードの数は $\frac{n}{4\lg n}$ なので，内部ノードの合計サイズは $(4\lg n - 2\lg\lg n - 2) \times \frac{n}{4\lg n} = n + \frac{2n(-\lg\lg n-1)}{4\lg n}$ ビットとなる．したがって，このデータ構造の合計サイズは $n\lg n + \frac{n(\lg\lg n+3)}{4\lg n} + n + \frac{2n(-\lg\lg n-1)}{4\lg n} = n + n\lg n + \frac{n(1-\lg\lg n)}{4\lg n} = n + n\lg n - o(n)$ ビットである．

木の高さ $O(\lg n)$ なので，木での探索は $O(\lg n)$ 時間である．葉ノードが保持する列の長さは $O(\lg n)$ であるので，葉ノード内の二分探索は $O(\lg\lg n)$ 時間であ

る．したがって，$D^{-1}(X_jX_k)$ の時間計算量は $O(\lg n)+O(\lg\lg n)=O(\lg n)$ 時間である． □

参考文献

[1] S. Maruyama, Y. Tabei, H.Sakamoto, and K. Sadakane. *Fully-online grammar compression*, In SPIRE, pages 218-229, 2013

[2] N. JLarsson and A. Moffat. *Offline Dictionary Based Compression*, *Proceedings of the IEEE*, 88(11):1722-1732, 2000

[3] Y. Takabateke and H. Sakamoto. *Space Efficient Reverse Phrase Dictionary for Grammar Compression*, 人工知能基本問題研究会 98, pages 42-47, 2015

[4] G. Navarrro and K. Sadakane. *Fully-Functional static and dynamic succinct trees*, *ACM Transactions on Algorithms*, Volume 10 Issue 3, Article No. 16, 2012

[5] Y. Takabatake, Y. Tabei, and H. Sakamoto. *Improved ESP-Index: A Practical Self-Index for Highly Repetitive Texts*, *In SEA*, pages 338-350, 2014

[6] 岸山直也. *Edit-Sensitive Parsing* による圧縮索引, 九州工業大学院情報工学府 情報科学専攻 知能情報工学分野 修士論文, 2011

[7] R. Raman, V. Raman and S. S. Rao. *Succinct indexable dictionaries with applications to encoding kary Trees and Multisets*, SODA,, pages 233-242, 2002