LR表によるトライ構造辞書の共通辞圧縮に関する研究

センリ　李輝　田中穂積

東京工業大学大学院 情報理工学研究科
{lzhan,li,tanaka}@cs.titech.ac.jp

1 はじめに

近年、自然言語解析や音声認識における辞書の規模が拡大する傾向にある。辞書の圧縮にはトライ構造がよく使われている。トライ構造はキーの共通接頭部をマージすることができるため、メモリの節約ができる。しかしトライ構造には、キーの接尾部が共通であってもマージできないので、キーの数が多くなるとメモリを圧迫するという問題がある。接尾部を共有できるDAWG構造[2, 8]も辞書の圧縮に使われる。DAWGは接尾部を共有できるが、葉ノードと辞書項目が一対一に対応していないので、応用範囲は制限される。

これらの問題を解決することが可能な、LR表による辞書データ圧縮法を提案する。我々は辞書項目を一つのCFG規則として扱う。CFGの左辺が品詞で右辺の文字列中の各文字を終端記号として扱う。このような辞書項目から得られるLR表はトライ構造とみなすことができる。トライ構造の葉はreduce操作と一対一に対応し、トライ構造のノードを葉に向けてたどることは、LR表では辞書項目の文字を連続してシフトする操作に対応する。

本論文では、LR表を利用して、トライ構造と同様に接頭部をマージし、DAWGと同様に接尾部をマージした辞書項目の新しい表現法について述べる。ただしDAWGと異なりこの新しい表現法では、葉ノードと辞書項目とが一対一に対応する。

2 トライ構造とDAWG

トライ構造は、キー集合Kの各キーの共通接頭部をマージして得られる木構造である[5, 7]。トライ構造はキーを構成する文字ごとに入力キーと比較する。この特徴により文字列からなる辞書項目の探索によく使われる。トライ構造を実現する方法として配列、2進木、ダブル配列などが良く用いられる[7]。

K={nation,notion}に対するトライ構造を図1に示す。ノードに振られた番号は状態を表すが、この番号は3節の図3に示すLR表の状態に対応させてある。

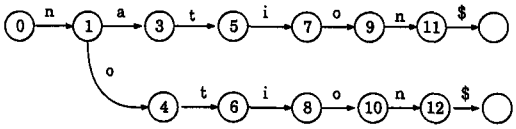

図1: トライ構造

図1から分かるように、トライ構造ではキーの共通接尾部 "tion" をマージできない。

トライ構造に対して、共通接尾部をマージした構造がDAWG (Directed Acyclic Word-Graph)[5, 2, 8]である。DAWGではキーの接頭部と接尾部の両方がマージされており記憶領域を節約できる。キー集合KのDAWGを図2に示す。

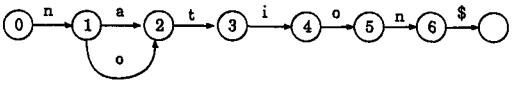

図2: DAWG構造

DAWGでは共通接尾部のマージにより探索成功時の状態（葉ノード）が複数のキーに対応しており、葉ノードとキーは一対一に対応しないので、たどったノードの履歴が必要となる。

3 LR表による辞書の圧縮

LR法を一般化したGLR法[9]は文脈自由文法(CFG)からLR表を構成し、LR表に書かれたアクションに従って解析を進める。辞書項目を一つのCFG規則として扱うことにより、LR表はキー集合からキーを探索する場合に用いるトライ構造としてみなすことができる。これを具体例により説明する。

集合Kに対応する辞書用CFG1の例を表1に示す。s, r, gはそれぞれshift, reduce, gotoアクションに対応する。CFG1から作成されるLR表を図3に示す。

表 1: 辞書用 CFG1

CFG1

```
0 START --> N
1 N --> n a t i o n
2 N --> n o t i o n
```

表 2: 辞書用 CFG2

CFG2

```
0 START --> N
1 N --> n a TION
2 N --> n o TION
3 TION --> t i o n
```

状態	n	a	o	$	t	i	N
0	s1						g2
1		s3	s4				
2				acc			
3					s5		
4					s6		
5						s7	
6						s8	
7			s9				
8			s10				
9	s11						
10	s12						
11				r1			
12				r2			

図 3: CFG 1 より作成した LR 表 1

状態	n	a	o	$	t	i	N	TION
0	s1						g2	
1		s4	s3					
2				acc				
3					s5			g6
4					s5			g7
5						s8		
6				r1				
7				r2				
8			s9					
9	s10							
10				r3				

図 4: CFG 2 より作成した LR 表 2

図 3の状態 0 で接頭部 "n" が共有され、状態 1 で "a" と "o" の先読みにより異なる状態 3 と 4 にシフトする。以下、この LR 表 1 に従って動作を行なうことが図 1に示すトライ構造をたどることに対応している。トライ構造と同様に共通接尾部 "tion" はマージされない。図 3に示す状態番号と図 1のノードの番号が対応させてあるので、このことを容易に理解できるだろう。辞書引き終了は LR 表 1 では reduce アクションとアクセプト (acc) が対応している。

4 接尾部のマージ

LR 表を使って接尾部をマージし、DAWG の問題点をも解決する新しい手法を提案する。そのために、辞書用 CFG の右辺の共通接尾部をまとめて、それを一つの非終端記号で置き換える。そして、この非終端記号を左辺とし、共通接尾部を右辺とする CFG 規則を最後に追加する。このようにして、表 1から表 2 に示す CFG を得ることができる。表 2の CFG から得られる LR 表を図 4 に示す。

図 4 を見ると状態 3 と 4 で "t" を shift した直後の状態が 5 となり、状態 5 から接尾部のマージが行なわれていることが分かる。これに対して、図 3では、状態 3 と 4 で "t" を shift した直後の状態がそれぞれ 5 と 6 であり、接尾部のマージが行なわれていない。

図 3と図 4 とを比べると、図 3 では状態 10 で行なう r3 と、非終端記号 "TION" に対する goto が増えていることが分かる。さらに、辞書引き終了は、図 3と同様に図 4でも、reduce と acc が対応していることが分かる。以上のことから、図 4では共通接尾部がマージされており、葉ノードと辞書項目が一対一に対応し、DAWG の問題点が解決され、記憶の節約がなされていることが分かる。どの程度の記憶の節約がなされるかについては 6節で詳しく検討する。

研究社英和中辞典 [6] から抽出した見出し語数 39,541 の辞書について、本節で述べた方法による接尾部のマージに関する評価実験を行なった。実験結果を表 3に示す。それによれば、トライ構造の LR 表と比べて、状態数で 26.3%、LR 表中のアクション数 (goto もアクションであるとみなす) で 9.1%の圧縮が可能であった。

5 goto アクションの置換

表 3に示した実験ではアクション数が 9.1%圧縮されている。圧縮率がさほど大きくならない理由は以下の通り。

表 3: 実験結果：本手法とトライの比較

	本手法	トライ	圧縮率 (%)
状態数	86176	117005	26.3
アクション数	142148	156545	9.1
shift 数	85974	117003	-
reduce 数	42073	39541	-
goto 数	14101	1	-

状態	n	a	o	$	t	i	N	TION
0	s1						g2	
1		s4	s3					
2				acc				
3					s5			sr1
4					s5			sr2
5						s6		
6			s7					
7	s8							
8				r3				

図 5: goto が置換された LR 表 3

表 4: 実験結果：本手法とトライの比較

	本手法	トライ	圧縮率 (%)
状態数	72075	117005	38.4
アクション数	121424	156545	22.1
shift 数	85974	117003	-
shift & reduce + reduce 数	42073	39541	-
goto 数	1	1	-

共通文字列を探索した後の reduce と goto アクションが必要になり、その分 LR 表中のアクション数が増える (図 4 の g6, g7)。

本手法では単語の共通接尾部をマージしているので、共通接尾部について reduce, goto を行なった直後に、さらに辞書引き終了のための reduce と goto を行なう。共通接尾部に対する goto とその直後に行なう reduce を一つのアクション、すなわち shift & reduce アクションで置き換えて LR 表中の状態数、アクション数を減らす方法がある [4, 1]。

例えば、"nation" を末尾まで探索し、先読みが "$" である時に図 4 の LR2 表では、r3 を行なってから、g7, r1, g2 を連続して実行する。この四つの連続したアクションの中で g7 と、その直後に実行する r1 を shift & reduce アクション sr1 としてまとめることができる。このような goto のアクション置換により、図 5 の LR 表 3 を得ることができる。図 4 と図 5 の LR 表を比較することにより、記憶のさらなる圧縮が得られていることがわかる。

表 3の実験に用いた辞書から、本手法を用いて LR 表をさらに圧縮した結果を表 4に示す。

goto と reduce を 一つの shift & reduce アクションで置き換えたので、状態数及び LR 表のエントリ数の圧縮率が 22.1%に向上していることが分かる。

6 最適な共通接尾部の選択

大きなキー集合に対しては最適な共通接尾部を自動的に抽出できることが望ましい。任意のキー集合に対して、自動的に最適な共通接尾部を抽出する方法を以下に示す。

まず、共通接尾部の抽出により、LR 表が圧縮可能な条件を明らかにする。変換前の LR 表を table1, 変換後の LR 表を table2 とする。LR 表中のアクションの数を action(table) で表し、変換により減少したアクションの数を merit とすると、merit は式 (1) を満足しなければならない。

$$merit = action(table1) - action(table2) > 0 \quad (1)$$

メリットを計算するための変数として共通接尾部の長さを L とし、共通接尾部の出現頻度を F とする。一つの共通接尾部に対して変換後と変換前とを比べると、アクション数は以下のように変化する。

- shift アクションの数: 共通接尾部の最初の文字以外が共有されるので、(L-1) × F + 1 だけ減る。ただし、共通接尾部を一回は解析しなくてはならないので、 L 増える。
- goto のアクションの数 : 変わらない。
- shift & reduce アクションの数 : 共通接尾部が出現する度に shift & reduce アクションを行なうので、F 増える。
- reduce アクションの数 : 各共通接尾部に対して reduce を一回行なうので、一増える。しかし、shift & reduce アクションを導入しているので、結局 F − 1 だけ減る。

以上、共通接尾部を非終端記号に変換する条件は式 (2) で表せる。

$$\begin{aligned} merit &= ((L-1) \times F + 1 - L) + (F-1) - F \\ &= L \times F - F - L > 0 \end{aligned} \quad (2)$$

式 (2) より式 (3) が得られる

$$F > L/(L-1) \quad (3)$$

式 (3) を共通文字列の長さを横軸、頻度を縦軸のグラフで表すと、図 6になる。

双曲線より右上の部分は merit > 0 の範囲である。図 6 からも分かるように、共通接尾部が長ければ長いほど、頻度が高ければ高いほど merit がある。

次に、二つの共通接尾部が重なっている場合を考える。例えば、キー集合 K={notion, nation, session, onion } では "tion" は "notion" と "nation" の共通接尾部になっており、 "tion" と重なる "ion" は "notion" "nation" "session" "onion" の共通接尾部になっている。この時、次の三つの場合が考えられる。

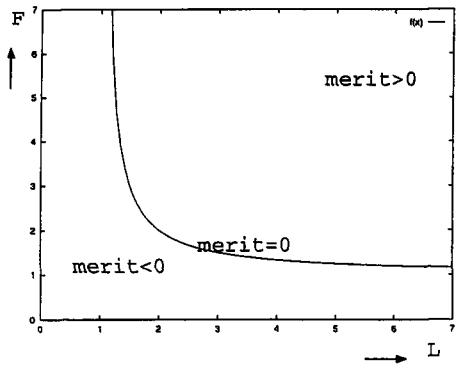

図 6: 最適な共通接尾部の選択

1. "tion" と "ion" を共通接尾部にする。
2. "tion" だけを共通接尾部にする。
3. "ion" だけを共通接尾部にする。

1 の方が 2 より圧縮率が大きいことは明らかである。1 と 3 の merit を比較するために、長い方の共通接尾部の長さと頻度をそれぞれ L1、F1 とし、短い方の共通接尾部の長さと頻度をそれぞれ L2、F2 とする (上記の例では L1 = 4, L2 = 3, F1 = 2, F2 = 4)。明らかに F2 > F1 である。$\Delta L = L1 - L2$ とし、短い共通接尾部のみを考える場合 (上記 3 の場合) には、長い共通接尾部の頭の ΔL 分が共有されなくなるので、1 の場合と比べて merit が $\Delta L \times F1$ だけ減る。一方、長い共通接尾部にあたる非終端記号を作る必要がなくなるので、この非終端記号を解析するための規則が不必要となり、merit は L1 だけ増える。

従って、$(L1 - \Delta L \times F1) \geq 0$ であれば、短い共通接尾部のみを取り上げ、$(L1 - \Delta L \times F1) < 0$ の場合には二つの共通接尾部をともに取り上げる。例えば、"tion" と "ion" の場合 $\Delta L = 1$ であるので、$4 - 2 > 0$ となり、"ion" だけを共通接尾部として取り上げる。

以上は二つの共通接尾部が重なっている場合であるが、これを n 個の共通接尾部が重なっている場合に容易に一般化することができる。

7 おわりに

本論文は、LR 表を用いたトライ構造辞書の新しい圧縮法を提案するとともに、記憶圧縮の観点から、最適な共通接尾部を求める方法を提案した。本圧縮法は、DAWG の持つ問題点が解消されている。実験によりトライ構造辞書と比べて大幅な記憶の節約が可能であることを実証した。

本文中では、はっきり述べなかったが、本手法の別な利点として、辞書引きと構文解析とが、GLR という共通の枠組によって処理することが可能な点をあげることができる。辞書項目の記述を音素列にすれば、本手法を HMM-LR に基づく大規模語彙音声認識システムに容易に組み込むことができる。

本手法をそのまま用いると、辞書項目の削除、付加にたいして始めから LR 表を作り直さなければならないことが問題である。付加に対しては論文 [3] の方法が利用できる。しかし削除に対しては、今後の課題である。英語の辞書ではなく、日本語の辞書の場合にも問題があるかもしれない。ただし、辞書項目が音素の系列であれば、問題はないだろう。

謝辞

本研究に対し有益なコメントを頂きました東京工業大学の藤井敦氏に感謝致します。

参考文献

[1] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers—Principles, Techniques, and Tools—*. Addison-Wesley, 1985.

[2] Appel A.W.and Jacobson G.J. The world's fastest scrabble program. *Communications of the ACM*, Vol. 31, No. 5, pp. 572–578, 1988.

[3] Hideki Mima and Kazuaki Ando. Incremental generation of lr(1) parse tables. *Natural Language Processing Pacific Rim Symposium 95*, Vol. 1, No. 1, pp. 600–605, 1995.

[4] Chapman N.P. *LR Parsing Theory and Practice*. Cambridge Univ., 1987.

[5] Alered V.Atto, John E.Hopcroft, and Jeffry D.Ullman. *Data Structures and Algorithes*. Addison Wesley Publish Company, 1984.

[6] 研究社. 英和中辞典. 研究社, 1988.

[7] 青江順一. トライとその応用. 情報処理, Vol. 34, No. 2, February 1993. 連載講座.

[8] 青江順一, 森本勝士, 長谷美紀. トライ構造における共通接尾辞の圧縮アルゴリズム. 電子情報通信学会論文誌, Vol. J75-D-II, No. 4, pp. 770–779, April 1992.

[9] 田中穂積. 自然言語解析の基礎. 産業図書, 1989.