002

TR-IT-0051

フレーム同期型 SSS-LR による連続音声認識
Frame Synchronous SSS-LR for Real Time Continuous Speech Recognition

門前 聖康
Monzen Seikou

Harald Singer

松永 昭一
Matsunaga Shouichi

1994.4.15

本研究では、フレーム同期型 SSS-LR による文節、および文音声認識について検討する。フレーム同期型 SSS-LR は、有限状態オートマトンにおける One-Pass サーチの手法を、文脈自由文法によって制御する連続音声認識に拡張したもので、同時に文脈依存型音韻 HMM である HMnet を用いて各時刻毎に音韻照合を駆動する音声認識手法である。

chart 法 Earley 法を用いて文脈自由文法を解析する One-Pass サーチは、文献 [4] [7] に提案されている。一方、文節音声認識において、拡張 LR 構文解析法に基づく HMM-LR が提案され、有効性が示されている。そこで本研究では、拡張 LR 構文解析法における、One-Pass サーチによる連続音声認識手法アルゴリズムを検討する。同時に HMnet を用いて音韻認識レベルでの認識精度の向上を目指す。

目次

第 1 章

はじめに

連続音声認識は、文節や文章を構成する単語列や音素列を一意化することが目標である。人間が普段なにげなく行なっているコミュニケーションのような、より円滑なインタフェースの実現において、連続音声認識技術は欠かすことのできない要素になっている。

近年、音声認識の研究において、自然な発話に対する認識・理解など、音声言語についての研究がより重要視されてきている。自然な発話に対する、音響処理系、言語解析系の構築などは、特に重要な課題である。より自然な発話では、音響的にも、言語的にも、バラエティーに富んだ音声が生成されるため、従来のような形式的な発話に対するアプローチに加えて、自然な発話の解釈系をモデル化する必要がある。そのモデル化の一環として、探索系の検討も重要である。

対話全体に関する様々な情報を認識のレベルに利用しようとする、より能動的な対話解析アプローチにおいては、各時点での最適な部分的情報を抽出する枠組として、発話の時間に並行して処理できる認識系が求められる。本研究では、フレーム同期型認識手法として、拡張 LR 構文解析法を用いた One Pass Viterbi による SSS-LR 連続音声認識について検討する。

本稿の構成は、先づ拡張 LR 構文解析法を用いた文脈自由文法制御の代表的文節認識手法である HMM-LR について考察する。次にフレーム同期型の認識に伴ういくつかの改訂要素と、認識アルゴリズムを述べる。認識実験による認識精度の評価を考察し、最後にまとめる。

第 2 章

フレーム同期型 SSS-LR 用いた連続音声認識

1　拡張 LR 構文解析法を用いた連続音声認識手法

1.1　拡張 LR 構文解析法

拡張 LR 構文解析法は、LR 級文法だけでなく、文脈自由文法を解析することができるため、広く用いられている統語解析手法である [5]。

文脈自由文法を予めコンパイルした LR テーブルを用いて解析するため、同じ文脈自由文法を扱う枠組である、チャート法やアーリー法に比べ、効率的に統語解析できるといった利点がある。

拡張 LR 構文解析法を利用する音声認識手法として HMM-LR が提案され、有用性が示されている [6]。

1.2　HMM-LR

HMM-LR は、HMM による音韻照合部と、LR パーザによる統語解析部とから構成される。

LR パーザでは、予め文脈自由文法をコンパイルした LR テーブルを参照して文法仮説を展開する。その後、各々の文法仮説から予測される音韻について、HMM による音素照合を駆動し評価する。これを、再びパーザによって文法仮説を展開することを繰り返す。最終的に最も尤度の高い文法仮説を認識結果とする。

確率文脈自由文法や確率 LR テーブルを用いる HMM-LR や、文章認識のための 2 段 LR パーザ、到達可能性機構による HMM-LR など、HMM-LR に関する様々な検討がなされている (例えば [1])。

また、HMM-LR に用いる音韻モデルとして、文脈依存型 HMM である HMnet を用いて音素照合する場合が SSS-LR であり、認識率の向上が示されている [2]。

HMM-LR の問題点

HMM-LR における問題点を考察する。

音素同期型の問題点

HMM-LR は、統語解析部によって音素照合を駆動する手法であるため、必然的に音素同期型の照合アルゴリズムになる。音素同期型の照合では、入力音声のフレーム数が異なる仮説同士を比較するための、フレーム数に対する尤度の正規化の必要がある。従来の HMM-LR ではフレーム当たりの平均尤度を用いているが、必ずしも精度の高い正規化とはいえない。

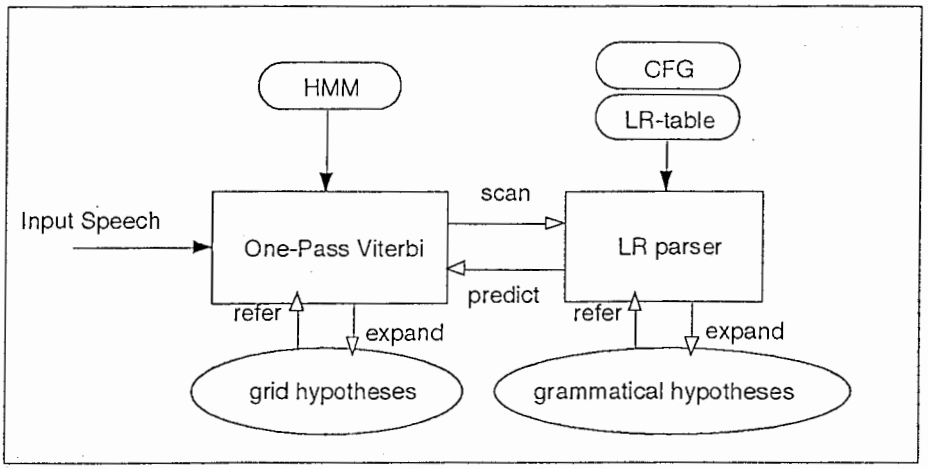

図 2.1: 認識系の構成図

スタックの併合について

複数の文法仮説が同時に存在する場合でも、以降の文法仮説が一致する仮説は一つにまとめあげることが出来る。以降の文法仮説が等しければ、以降の音響尤度も等しくなるため、その時点で尤度の高い文法仮説は、最終時点でも尤度が高くなる。従って、各時点での最も尤度の高い仮説のみを残すだけでよく、尤度の低い仮説はそれ以降展開する必要がない。

HMM-LR は、一つの LR スタックを一つのセルで表現するので、スタックの併合はセルの併合によって表現される。しかし、音素照合を音素毎に行なうため、長さの異なる音素列に対するセルの併合は難しい。現状ではスタックの併合は実現しておらず、音素照合を重複する。

アクションの競合について

LR テーブルのある状態において、一つの音素に対するアクションが複数定義されている場合をアクションの競合という。HMM-LR では、一つのセルで一つのスタックだけを表現するため、アクションの競合が生じた場合はセルを分割する。従って、同じ音素系列に対する複数のセルが存在する可能性がある。特に文音声認識のように、一つの音素系列に対して文法表現が極めて曖昧になり、アクションの競合が生じやすい場合、一つの音素系列に対して数多くのセルを作成する可能性が高いため、HMM-LR は文音声認識には不向きである。

2　One-Pass Viterbi HMM-LR 連続音声認識手法

One-Pass Viterbi に基づくフレーム同期型 HMM-LR の認識を図 2.1に示す。

2.1　探索部

One-Pass Viterbi

One-Pass サーチ (One-Pass DP, One-Stage DP) は、Ney [3] らによって提案されたサーチ方法で、入力音声の時間長 n に対し、照合に必要な経路展開の計算が $O(n)$ で済む効率的な

サーチ方法である。

入力音声の各フレームにおいて、照合を行なうと同時に音素や単語等の連鎖についての統語的制約を逐一適用していく。入力音声の最終フレームにおける文法仮説中の、最も尤度の高い仮説を認識結果として出力する。言語知識から得られる制約を、音素照合に対して動的に適用する事ができるため、全体の探索空間を大幅に削減できる。従ってラティスパージング等の手法に比べてオーバーヘッドが少なく、認識精度や処理量の点から、有効なサーチ方法である。

One-Pass サーチにおいて、HMM の Viterbi アルゴリズムを用いた照合の場合に、このアルゴリズムを One-Pass Viterbi と呼ぶ。

2.2 探索部と統語解析部の分離

音声認識では、認識の最終段階には One-Pass Viterbi により最適な経路を決定できるものの、探索途中においては一意に音素系列を決定できないため、同時に複数の文法仮説を保持しておく必要がある。しかし、生成可能な文法仮説の総数は組み合わせ的に爆発する。

また、探索部における音素照合では、音素系列が一意化されれば各音素系列に対する尤度も決定するが、統語解析部では、文法上の曖昧さから、一つの音素系列に対する文法仮説を一本化できず、複数の文法仮説を作成し得る。同じ音素系列である文法仮説に対して何度も音響照合を駆動するのでは、極めて冗長である。

従って音素照合における曖昧さと統語解析部における曖昧さを、各々に反映させないよう、分離する必要がある。

音素照合の曖昧さが、できるだけ統語解析部の負荷とならないために、動的に文法仮説を展開する。音素照合部での音素検出に際してのみ統語解析を駆動させ、探索の対象となりうる文法仮説のみを展開することで、生成する仮説数を極力小さく抑える。

また、統語解析部の曖昧さが、音素照合部の無駄を生じさせないために、同じ音素系列に対する文法仮説についてパッキングを行なう。音素系列を表現する音素履歴木 (2.3参照) を作成する段階で、同じ音素系列の文法仮説をひとまとめにパッキングする。音素照合部へは、展開可能な音素履歴、即ち接続可能な音素系列という情報だけを与えるようにする。

2.3 統語解析部

統語解析部は、状態ネットワークと音素履歴木から構成される。

状態ネットワーク

フレーム同期処理では、入力音声のフレーム毎に構文解析部を駆動する可能性があるため、様々な状況のスタックを同時に表現しなければならない。

例えば、入力音声のあるフレームで音素を検出し、文法仮説を展開する場合、その音素に対するアクションが reduce であったとすれば、LR の状態スタックから、状態を取り除く動作を行なう事になる。しかし、同じフレームにおいて、まだ音素を検出していない grid 候補も存在する可能性がある。この grid 仮説は、数フレーム後に音素を検出するための grid であるため、この時点では reduce を行なう前のスタックに対応している。従って、同時刻に、reduce を行なう前のスタックと、行なった後のスタックを保持しておかなければならない。各スタックに対して動作を実行する直前に、スタックのコピーを作成し、コピーに対して動作し、オリジナルのスタックを残す方法もあるが、極めて冗長であり、計算機的な負荷も大きい。効率化のために、LR の状態スタック構造をグラフ構造に展開する。以下このグラフを状態ネットワークと呼ぶ。

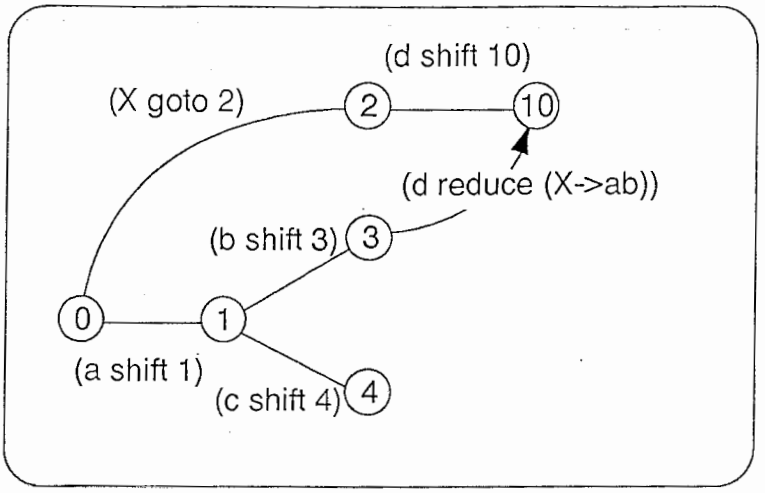

図 2.2: LR 状態ネットワーク

- 状態ネットワークの構成と統語解析

  図 2.2に、状態ネットワークの例を示す。状態ネットワークでは、スタック中の各状態をネットワーク上のノードで表す。LR 解析の初期状態は、ネットワークのルートノードに対応する。スタック構造はネットワークノード系列で表現される。即ち、ルートノードからそのノードへ至るノードの系列が、そのノードをスタックの最上位 (stack top) とするスタックを表現している。各ノード間を結ぶリンクは、リンク元のノードから、アクションを実行した際の遷移を意味する。以下このリンクをアクションリンクと呼ぶ。

  状態ネットワークを用いる構文解析例を図 2.3に示す。図 2.3の右側が状態ネットワーク、左側が各々のネットワークが対応しているスタック構造である。文法として、図 2.4が与えられたときの解析の流れは、

  1. 初期設定。
     ・空のスタックに初期状態 0 を push する。
     ・ネットワークではルートノード s_0 を作成する。
  2. 記号 a を走査する。
     ・スタックに対し $(a\ shift\ 1)$ を実行、状態 1 を push する。
     ・ネットワークでは、状態 1 に対応する新たなノードを作成し、ルートノードから、アクションリンク $(a\ shift\ 1)$ を張る。
  3. 記号 b と c を並列に走査する。
     ・スタックを分割し、$(b\ shift\ 3)$、$(c\ shift\ 4)$ を、各々のスタックに対して実行する。
     ・ネットワークでは、状態 3 と状態 4 に対応するノードを作成し、状態 1 のノードから、各々 $(b\ shift\ 3)$、$(c\ shift\ 4)$ のアクションリンクでつなぐ。
  4. 左側のスタックについて d を走査する (記号列 abd を走査)。
     ・左側のスタックに対し、$(d\ reduce\ (X->ab))$ を実行し、状態を 2 つ pop する。次に、X に関して $(X\ goto\ 2)$ を実行し、状態 2 を push、さらに、状態 2 において d に関して $(d\ shift\ 10)$ を実行し、状態 10 を push する。

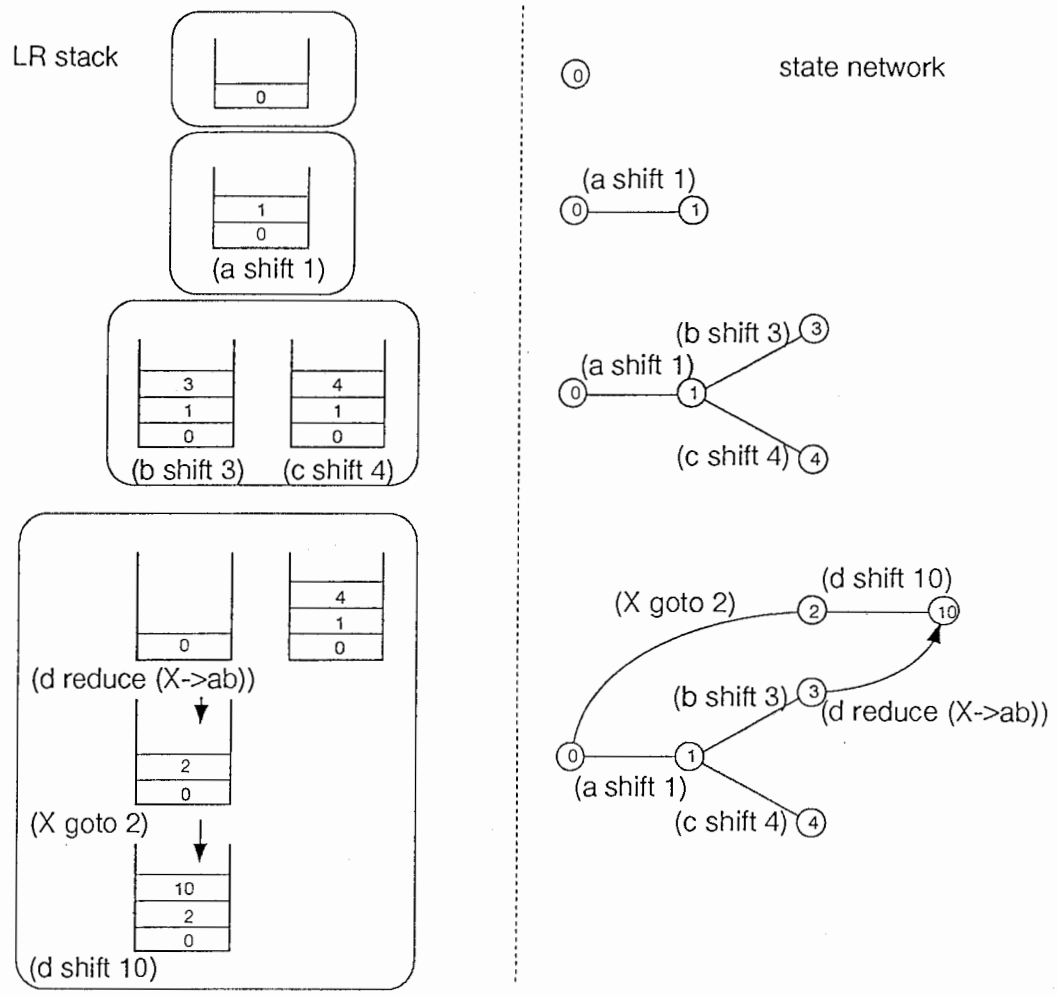

図 2.3: 状態ネットワークを用いた構文解析例

CFG rules

1 S' -> S
2 S -> XY
3 S -> Z
4 X -> ab
5 Y -> de
6 Z -> ac

LR table

	a b c d e	X Y
0	s1	g2
1	s3 s4	--------
2	s10	--------
3	r4	--------
4	-------	--------
⋮	⋮	⋮
10	--------	--------
⋮	⋮	⋮

図 2.4: 文脈自由文法と LR テーブル

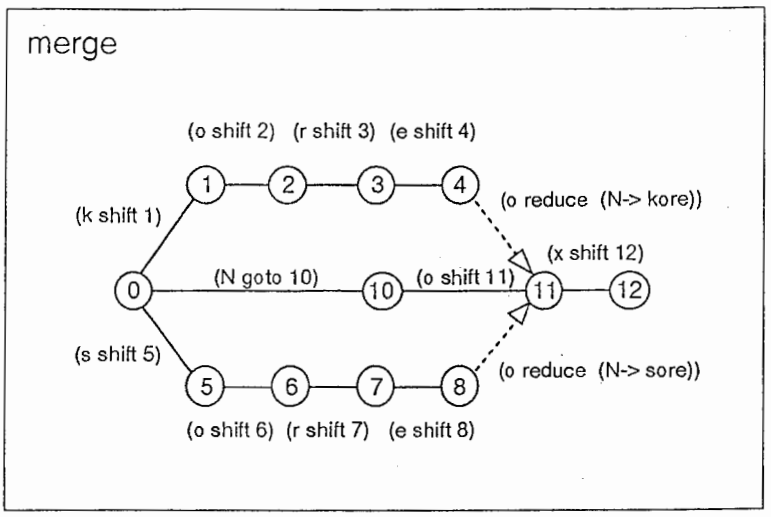

図 2.5: ネットワークの併合の例

・ネットワークでは、状態 3 のノードから、親のノード を 2 回辿り、そこ (状態 0 のノード) で X に関して状態 2 に対応するノードを作成し、$(X\ goto\ 2)$ のアクションリンクを張る。次に、状態 2 のノードから d に関して状態 10 に対応するノードを作成し、$(d\ shift\ 2)$ のアクションリンクを張る。最後に、 d に関する shift 動作で最終的に作成された状態 10 のノードに対して、この一連のアクションのもととなった状態 3 のノードから、$(d\ reduce\ (X->ab))$ のリンクを張る。

- スタックの併合

  状態ネットワークでは、スタックの併合 (merge) は、遷移先のノードの一本化で表現される。図 2.5は、記号列 'koreo' と'soreo' を併合した例で、'koreo' に対応するノード系列 $0-1-2-3-4=0-10-11-12$ が既に作成されており、'soreo' に対応するノード系列 $0-5-6-7-8=0-10-11-12$ では、 reduce 動作で辿った親ノードから goto でのアクションにおいて遷移する状態 10 のノードで併合される。

- アクションの競合

  状態ネットワークでは、アクションの競合 (conflict) は、一つのノードからの多重遷移で表現される。図 2.6では、'kore' に続く 'o' に対し、 2 つの reduce と 1 つの shift が競合した例で、 2 つの reduce に関して、ノード系列 $0-1-2-3-4=0-15-16$ とノード系列 $0-1-2-3-4=0-10-11$ が対応し、各々の reduce に関する状態 4 からのアクションリンクが張られる。 shift に対しては状態 4 のノードから状態 20 のノードへの shift のアクションリンクが張られる。

- ネットワークの不活性化

  状態ネットワークでは、一度行なったアクションを再度実行することはない。例えば図 2.5の併合の例では、'koreo' に対応するノード系列が既に作成されている場合、 'soreo' については、状態 10 に対する併合によって、自動的に状態 11 への遷移が行なわれる。従って、一度不活性化 (ネットワーク上に展開) されたアクションを繰り返すことはない。

- ネットワークの予備展開

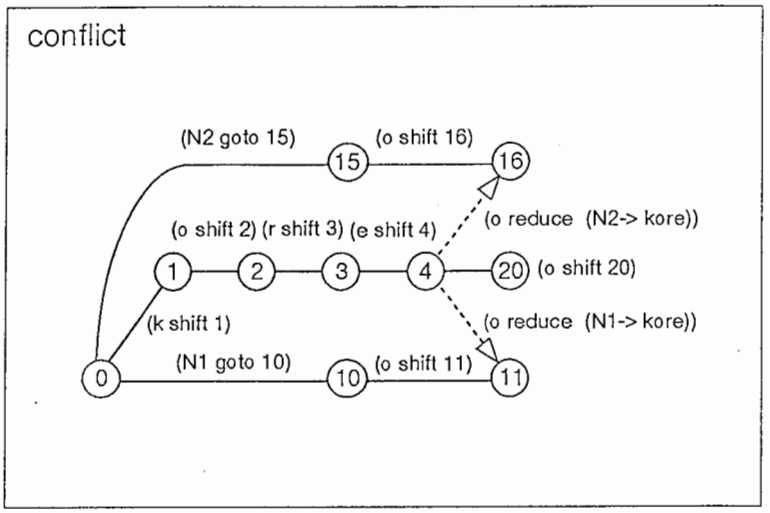

図 2.6: ネットワークの競合の例

状態ネットワークは、音素照合の音素検出に際して動的に展開されるものであるが、できるだけ展開処理が認識全体の負荷とならないために、予め、予備的にネットワークを展開しておくこともできる。認識の開始時においては、展開されるネットワークのノード数は多いが、認識が進むにつれ、展開されるノード数は小さくなると考えられる。従って、認識の開始部分のネットワークを、予め、ある程度展開しておけば、認識全体に対する負荷も小さくなると考えられる。

- グラフ構造の比較

  状態ネットワークはグラフ構造であり、他の文脈自由文法の解析手法におけるグラフ構造と、本質的に等価である。
  図 2.7に、下降型チャート法におけるグラフ構造と拡張 LR 構文解析法におけるグラフ構造を並べる。
  図から二つのグラフ構造が、事実上等価であることが分かる。しかし、拡張 LR 構文解析法の場合、バックトラックの必要がないためグラフ自体が簡略である。

音素履歴木

音素履歴木は、音素を木のノードに対応させた木構造のデータである。図 2.8に音素履歴木の例を示す。ルートノード s_0 から各ノードに至る経路が、一つの音素系列を表現する。

音素履歴木は、本来バックポインタとしての役割を持つデータであるものの、バックポインタとしての役割だけでなく、いくつかの有益な機能を持っている。

バックポインタとしての音素履歴木

音素履歴木は、本質的にバックポインタとしての役割を持つ。音素履歴木中のノードが決定すれば、そこまでの音素系列を一意に決定することができる。探索した全ての可能な音素系列を線形に保存する場合に比べて、効率的に音素履歴を保存できる。

また、ビームサーチの技法を用いて探索する場合、各時刻での grid 候補数はビーム幅を越えないので、全体でビーム幅の数だけの音素履歴を残していくだけで済むとも考えら

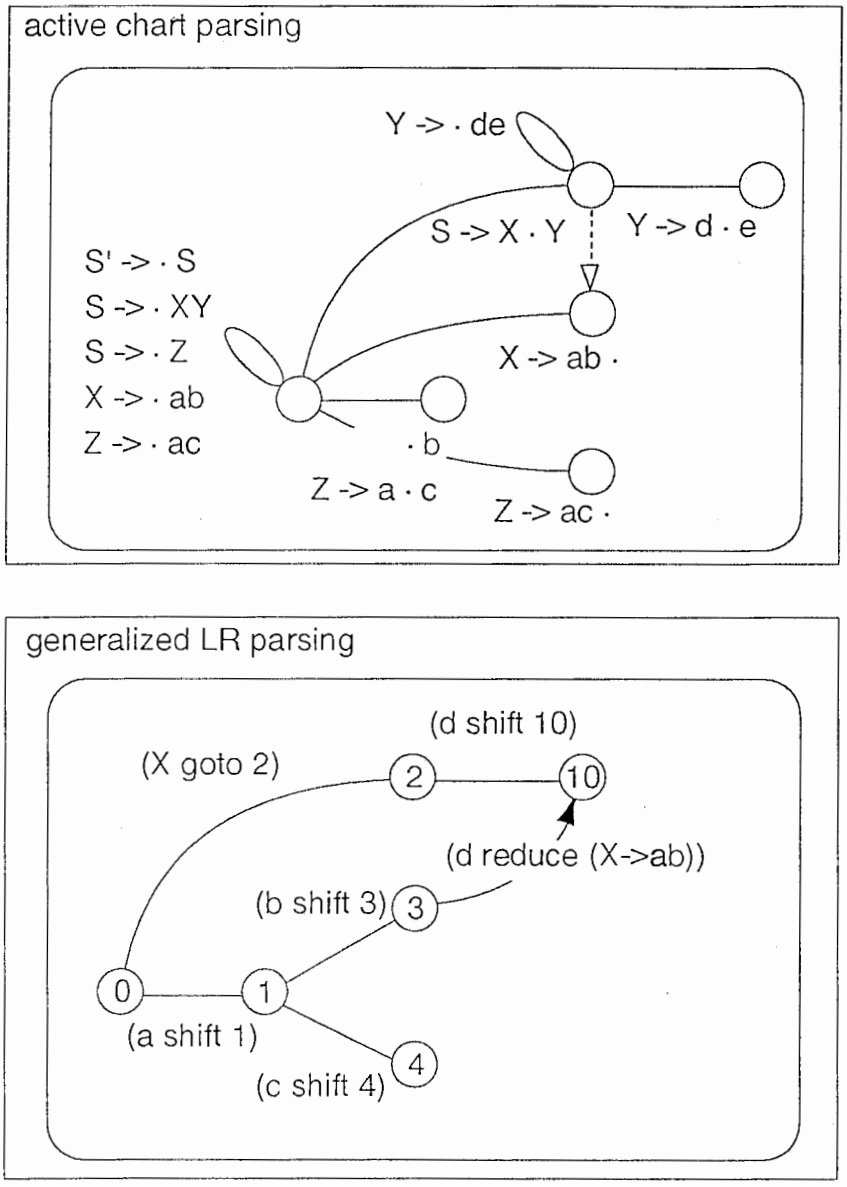

図 2.7: active chart parsing におけるグラフ構造との比較

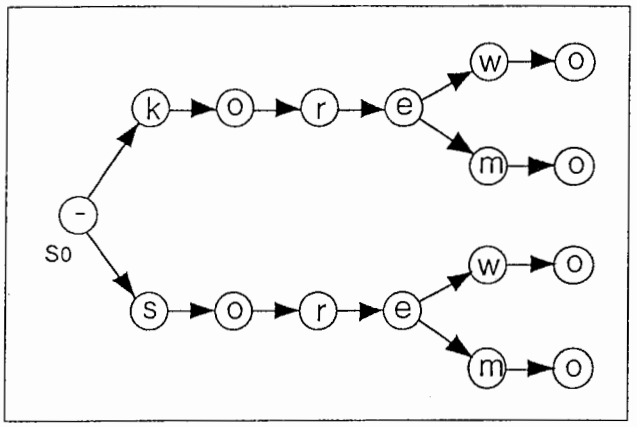

図 2.8: 音素履歴木の例

れる。しかし、ある時刻で grid 仮説に残っている各 grid の音素履歴は比較的重複しているものも多く、必ずしもビーム幅個の音素履歴を保持する必要がない場合もある。

音素履歴木では、こうした重複は生じない。

状態ネットワークの最小表現としての音素履歴木

音素照合において必要となる統語情報は、ある音素を走査した後の次に予測される音素は何か、即ち走査後の状態ネットワークノードからアクションが定義されている音素は何か、ということだけである。しかし、曖昧な文法仮説を表現する LR 状態ネットワークから、予測される音素を計算する場合には、複数の状態ネットワークノードに対して予測音素の和集合を計算しなければならない。

そこで、音素履歴木を展開する時に、状態ネットワークから得られる予測音素の和集合を計算し、音素履歴木上に保存しておく。この操作は、文法仮説の曖昧性を音素照合に影響させないためのパッキングに対応する。

音素照合では、予測音素を計算してある音素履歴木に対してアクセスするだけでよい。

音素履歴の比較の容易さ

音素照合において、同じ grid が存在する場合、尤度の高い grid のみを残すだけでよいため、grid 仮説中に同じ grid が存在するか否かの判定を行なう。

同じ grid である条件の一つが、音素履歴が等しいことである。grid 仮説の展開の度に grid を比較するため、音素履歴の比較も頻繁に行なわれる。従ってより簡単な比較方法で済ませたい。

音素履歴木では、対応する音素履歴のノードが等しいか否かを調べるだけでよいため、比較が容易である。

先読み処理

文脈依存型 HMM である HMnet を用いる場合、音素の前後環境を考慮しなければならない。当該の予測音素だけでなく、先行した音素、後続可能な音素について、予め知っておく必要がある。特に後続音素に関しては、LR 構文解析で予測される各音素について、さらにその予測音素に後続可能な音素まで予め計算しなければならない。

このため、構文解析において状態ネットワークを展開する際に、音素の予測だけでなく、各々の予測音素に対し、さらに構文解析を実行して後続可能な音素を調べる。後続音素を計算するための先読みにおいて、2通りのインプリメントが考えられる。

1. 各予測音素を走査したものとみなし、状態ネットワークを展開する。展開されたノードにおける予測音素を、現時点での後続音素とする。

2. 各予測音素について、 LR 状態ネットワークの展開はせず、単に次の構文解析を模倣するだけとする。

1 の場合、最終的に走査した音素に対しては、構文解析の重複が避けられるものの、最終的に走査しなかった音素に対しても、状態ネットワークを展開してしまうので、ネットワークの無駄が生じる。

2 の場合、構文解析を模倣した後、実際の音素の走査に際して、再び同様の解析処理を行なうため、計算量が増加する。しかし、最終的に走査されなかった音素に対してネットワークを展開することはないため、ネットワーク上の無駄はない。

本研究では、2 でインプリメントしている。

2.4　ビームサーチによる枝刈

統語制約として文脈自由文法を用いるため、全体の探索空間は事実上無限大の大きさになる。従って、可能な全ての仮説に対する総当り的な探索は非現実的であり、ビームサーチの技法による grid 仮説の枝刈を行なう。

- 枝刈の基準

  枝刈の目的は、計算量の低減である。認識結果として不要と思われる仮説を探索対象から取り除くことで、計算量の低減を図る。枝刈は、各仮説の尤度の高い順に任意個の仮説のみを今後の探索対象として残す場合と、各仮説の最も尤度の高い仮説の尤度から、しきい値以内の尤度の仮説のみを今後の探索対象として残す、2 通りがある。前者の方法では、全体の仮説から任意個の仮説を選び出すためのソートが必要であるが、探索全体を通して安定した計算量で済む。後者の方法では、ソートは不要であり、また、局所的に不明瞭な音声入力に対しては、保持する仮説数を増やし、逆に明瞭な音声に対しては、保持する仮説数を抑えるため、前者の方法に比べて柔軟である。

  前者の場合は、残す仮説の数をいくつにすればよいのか、後者の場合は、しきい値をいくらにすればよいのかが、一意に決定できない。多くは経験から適宜に決定している。

  本研究では、前者の方法で枝刈する。

- 仮説間の比較

  音素同期型 HMM-LR では、音素同期で音素照合を行なうため、文法仮説そのものに対して枝刈を行なう。各文法仮説毎の尤度を比較し、尤度の高い文法仮説だけを残す。各文法仮説は入力音声のフレーム毎に尤度を持っているものの、入力音声のフレーム数が重めば重むほど尤度は小さくなるため、入力音声の各フレームの長さに対する正規化が必要になる。 HMM-LR では、文法仮説の各フレームの尤度の中で、フレーム数に対する平均尤度が最も高い尤度を、その文法仮説の尤度としているが、必ずしも精度の高い正規化とはいえない。

フレーム同期型 HMM-LR のビームサーチでは、入力音声の各フレームにおいて grid を比較するため、全ての grid 仮説の入力フレームの長さは等しく、正規化の必要がない。

- N-best について

  音素同期型 HMM-LR では、文法仮説の併合を行なわないため、1 位の認識結果だけでなく、任意の順位の認識結果において最適性が保証される。

  フレーム同期型 HMM-LR では、文法仮説の併合に伴い、One Pass Viterbi において、尤度の高い grid のみを残す。従って、1 位については最適性が保証されるものの、2 位以下の結果における最適性は失われる。疑似的な N-best として、同じ文法仮説の grid に対して、音素系列の異なる複数の grid を尤度の高い順に任意個残す方法がある。

  本研究では、疑似 N-best に従い、複数の認識結果を出力する。

- 局所的枝刈りについて

  音素同期型 HMM-LR では、同様な文法仮説ばかりを数多く生成することがないように、一つの文法仮説から展開し得る仮説数に制限を与える。各仮説を展開する段階で、尤度の高い任意個の仮説だけを残し、その後広域的な枝刈によって、さらに全体の仮説を絞り込む。

  フレーム同期型の場合、音素検出に従って文法仮説を展開し、新たな grid を生成する時、全ての grid の尤度が等しいため、枝刈による絞り込みはできない。

  文法仮説の展開に伴う局所的な枝刈はできないものの、grid 仮説の展開に伴う多段階での局所的な枝刈ができる。新たな grid 仮説を展開する際に、同じ音素系列の grid の数を制限する、同じ文法仮説の grid の数を制限する、また、照合中の音素が同じ grid を制限する、さらに、照合中の音素 HMM の状態が同じ grid を制限するといった、多段階での枝刈が可能である。特に、文音声認識における文節間での仮説展開のように、瞬間的に仮説展開が爆発する場合には、有効な枝刈方法の一つと考えられる。但し、本研究では局所的な枝刈りは行なっていない。

第 3 章

データ構造と認識アルゴリズム

1　データ構造

全体の構造

フレーム同期型 SSS-LR の認識系は、HMM 部、One Pass Viterbi による探索部、LR パーザによる統語解析部とから構成される。
探索部では、HMnet を用いて、入力音声に対する音素照合を繰り返す。音素が検出された場合に、統語解析部が呼び出され、文法仮説を展開する。展開された文法仮説から予測されている音素について、再び入力音声に対して音素照合を行なう。

HMnet に関するデータ構造

HMM 部では、HMnet における '共有される状態' についてのデータと、'展開した状態' についてのデータを持つ。
'共有される状態' のデータ構造は、

- 適合する文脈
- 混合分布のインデクス
- 状態遷移確率

適合する文脈は、当該 HMnet 状態 を共有し得る文脈で、先行音素、中心音素、後続音素から成る。この文脈をもとに、'展開した状態' を構築する。
混合分布は、混合率を付加した単一ガウス分布の集合。
状態遷移確率は、自己ループの場合と、次状態遷移の場合の各々の状態遷移確率。

'展開した状態' は、'共有される状態' を適合する文脈に従って展開した left-to-right の HMM のことで、線形展開、木構造展開、未展開の 3 通りの展開方法がある。但し、本研究では線形展開だけを実験する。

'展開した状態' のデータ構造は、

- '共有される状態' のインデクス
- 後続する '展開した状態'

One Pass Viterbi に関するデータ構造

One Pass Viterbi では、grid が基本 unit になる。
grid のデータ構造は、

- 入力音声のフレーム番号
- 過去の音素系列 (音素履歴木のノード)
- 音素
- 音素 HMM の状態番号
- 尤度

grid は、入力音声、過去の音素系列、照合中の音素、照合中の音素 HMM の状態番号、の 4 次元で構成される空間の 1 点を表す。従って、各々の軸の位置についての情報を持つ。
尤度は、対数化した確率値である。
継続時間制御を導入する場合、継続時間長の軸に対する情報を加え、5 次元の空間における 1 点を表す形になる。
最終的な認識結果として、音素系列だけでなく、セグメントされたフレーム位置等のより詳細な情報を得るためには、音素履歴木の他にバックポインタが必要になる。

バックポインタのデータ構造は、

- セグメントしたフレーム番号
- 親のバックポインタ

LR 構文解析に関するデータ構造

LR 構文解析に関するデータは、音素履歴木と状態ネットワークである。
状態ネットワークは、ネットワークノードとアクションリンクの 2 種類のデータ構造で構成される。

ネットワークノードのデータ構造は、

- 親のネットワークノード
- LR 状態番号
- アクションリンクの集合

親のネットワークノードは、当該ノードを直接生成するもととなったノード。REDUCE アクションでの、ノードのトレースに用いる。
LR 状態番号は、LR テーブルを参照するときに用いる。
アクションリンクの集合は、当該ノードから動作可能な全てのアクションについてのリンク情報である。

アクションリンクのデータ構造は、

- 遷移先のネットワークノード
- アクションのタイプ
- 音素 (symbol)
- アクションの引数

遷移先のノードは、もとのノードから、当該アクションを実行して遷移するノード。
アクションのタイプは、SHIFT、REDUCE、GOTO、ACCEPT のいずれか。
音素 (symbol) は、当該アクションを実行するための要因となった音素 (symbol)。
アクションの引数は、アクションのタイプが SHIFT または GOTO の場合は次の状態番号、
アクションのタイプが REDUCE の場合は文法規則番号。

音素履歴木ノードのデータ構造は、

- 親の音素履歴木ノード
- 娘の音素履歴木ノード集合
- 対応する状態ネットワークノード集合
- 予測音素
- 後続音素

親の音素履歴木ノードは、認識終了時のトレースバックに用いる。
娘の音素履歴木ノードは、後続する各音素について展開した時のノードの集合。
対応する状態ネットワークノード集合は、音素照合により音素を検出した時に、LR パーサによって展開する状態ネットワークノードの集合。
予測音素は、対応する状態ネットワークノード集合の全てのノードにおける予測音素の和集合。
後続音素は、各予測音素に後続可能な音素の集合。

2 連続音声認識アルゴリズム

One Pass Viterbi アルゴリズム

One Pass Viterbi によるフレーム同期型 SSS-LR の認識アルゴリズムを以下に示す。

[記号の定義]

(i,j,n,s)	入力音声の第 i フレームにおける音素履歴木ノード s における, 予測文脈 (音素) n の HMM の第 j 状態の grid.
$f(i,j,n,s)$	grid (i,j,n,s) における累積尤度.
$c(i,j,k,n)$	grid (i,j,n,s) から $(i+1,k,n,s)$ へ遷移する際の尤度 $(= log(a^n_{jk}) + log(b^n_{jk}(i)))$.
$GHYP, GHYP'$	grid の集合 (grid 仮説).
$GHYP(s), GHYP'(s)$	音素履歴木ノード s についての grid 集合 $(= \{(i,j,n,t)\|t = s\})$.
$Pruning(hyp,num)$	grid 集合 hyp の中からスコアの高い順に num 個だけ残した grid 集合.
$LRparser(s,n)$	音素履歴木ノード s において, 文脈 (音素) n についてのアクションを実行した時に作成される音素履歴木ノード.
$PredictingContext(s)$	音素履歴木ノード s における予測文脈の集合.
Jn	文脈 n の HMM の最終状態.
$Transition(n,j)$	文脈 n の HMM の第 j 状態から遷移可能な HMM の状態の集合.
$TopN, TopB$	局所ビーム幅, 広域ビーム幅.
s_0	音素履歴木の root ノード.
$TraceBack(g)$	grid g についてトレースバックして得られる音素系列.
$ParseCompleted(s)$	音素履歴木ノード s が、文法的に完了しているか否か.
S	音素履歴木ノード集合

[One Pass Viterbi アルゴリズム]

1. $S = \{s_0\}$, $GHYP = \{(0, 1, n, s_0) | n \in PredictingContext(s_0)\}$.
2. $i = 0....I-1$ について, 3〜5を実行.
3. grid の展開
   (a) $GHYP' = \emptyset$
   (b) $s \in S$ について,
   - $GHYP(s) = \emptyset$.

   (c) $(i, j, n, s) \in GHYP$ について, 以下を実行.
   $k \in Transition(n, j)$ について, 以下を実行.
   i. $g = f(i, j, n, s) + c(i, j, k, n)$
   ii. k について、
   - $k < Jn$ ならば、
     - $(i+1, k, n, s) \notin GHYP(s)$ ならば、
       $GHYP(s) = GHYP(s) + (i+1, k, n, s)$
     - $(i+1, k, n, s) \in GHYP(s)$ かつ $g > f(i+1, k, n, s)$ ならば、
       $f(i+1, k, n, s) = g$

   - $k = Jn$ ならば、
   A. $s' = LRparsing(s, n)$.
   B. $s' \notin S$ ならば、$S = S + s'$.
   C. $m \in PredictingContext(s')$ について, 以下を実行.
     - $g = f(i, j, n, s) + c(i, j, k, n)$
     - $(i+1, k, m, s') \notin GHYP(s')$ ならば、
       $GHYP(s') = GHYP(s') + (i+1, k, m, s')$
     - $(i+1, k, m, s') \in GHYP(s')$ かつ $g > f(i+1, k, m, s')$ ならば、
       $f(i+1, k, m, s') = g$

4. $s \in S'$ について, 以下を実行.
   - $GHYP' = GHYP' + Pruning(GHYP(s), TopN)$.
5. $GHYP = Pruning(GHYP', TopB)$.
6. $best\_score = -\infty$, $best\_grid = \emptyset$.
7. $(I, Jn, n, s) \in GHYP$ について以下を実行.
   - $best\_score < f(I, Jn, n, s)$ かつ $ParseComplete(s)$ ならば、
     $best\_score = f(I, Jn, n, s)$
     $best\_grid = (I, Jn, n, s)$
8. $recognition\_result = TraceBack(best\_grid)$

LR 構文解析アルゴリズム

音素履歴木と LR 状態ネットワークを用いる構文解析アルゴリズムを以下に示す。

[記号の定義]

$DaughterStringNodeOf(s,m)$	音素履歴木ノード s についての文脈 m における音素履歴木の娘ノード.
$Status(s)$	音素履歴木ノード s についての活性値 (ACTIVE か PASSIVE か).
$NetworkNodeOf(s)$	音素履歴木ノード s が対応している状態ネットワークノード集合.
$ActionAt(st,m)$	LR の第 st 状態における文脈 m についてのアクションの集合.
$StateOf(p)$	状態ネットワークノード p の状態番号.
$AntecedentNodeOf(p,n)$	状態ネットワークノード p について, n 回親をたどった時の先祖ノード.
$ReachedNetworkNodeOf(p,a)$	状態ネットワークノード p について, アクション a を実行した際に到達するノード.
$LeavingNetworkNodeOf(a)$	アクション a を実行するための元のノード.
$ActionTypeOf(a)$	アクション a についてのアクションタイプ.
$RuleOf(a)$	アクション a が REDUCE である時の文法規則番号.
$LeftHandSymbolOf(r)$	文法規則 r の左辺記号.
$Length(r)$	文法規則 r の右辺記号の数.
$Sset, Rset, Aset$	SHIFT, REDUCE, ACCEPT アクションの各々の集合.

[LR 構文解析アルゴリズム]

1. given
   - string history node s
   - scanned context m
2. $s' = DaughterStringNodeOf(s, m)$
3. if ($Status(s')$ == PASSIVE) then
   return s'
4. initialization
   - $Sset = \emptyset$
   - $Rset = \emptyset$
   - $Aset = \emptyset$
5. for each $p \in NetworkNodeOf(s)$ do following.
   (a) for each $a \in ActionAt(StateOf(p), m)$ do following.
   - if ($ActionType(a)$ == SHIFT) then
     $Sset = Sset + a$
   - else if ($ActionType(a)$ == REDUCE) then
     $Rset = Rset + a$
   - else if ($ActionType(a)$ == ACCEPT) then
     $Aset = Aset + a$
6. for each $a \in Rset$ do 7. $\sim$ 8
7. $q = AntecedentNodeOf(LeavingNetworkNodeOf(a), Length(RuleOf(a)))$
8. for each $a' \in ActionAt(StateOf(q), LeftHandSymbolOf(RuleOf(a))$ do following.
   if ($ActionType(a')$ == $GOTO$) then
   - $q' = ReachedNetworkNodeOf(q, a')$
   - for each $a'' \in ActionAt(StateOf(q'), m)$ do following.
     (a) if ($ActionType(a'')$ == SHIFT) then
     $Sset = Sset + a''$
     (b) else if ($ActionType(a'')$ == REDUCE) then
     $Rset = Rset + a''$
     (c) else if ($ActionType(a'')$ == ACCEPT) then
     $Aset = Aset + a''$
9. for each $a \in \{Sset + Aset\}$ do following.
   - $q' = ReachedNetworkNodeOf(LeavingNetworkNodeOf(a), a)$
   - $NetworkNodeOf(s') = NetworkNodeOf(s') + q'$

10. $Status(s') = \mathrm{PASSIVE}$
11. $DaughterStringNodeOf(s, m) = s'$
12. return s'

先読み処理

音素履歴木及び LR 状態ネットワークについて、予測音素と、さらに各々の予測音素に後続する音素を計算するための先読み処理のアルゴリズムを以下に示す。

[記号の定義]

$PredictingContext(s)$	音素履歴木ノード s について予測されている文脈.
$NetworkNodeOf(s)$	音素履歴木ノード s が対応している LR 状態ネットワークノード集合.
$Phoneme(s)$	音素履歴木ノード s が対応している音素.
$ActionAt(st,m)$	LR の第 st 状態における文脈 m についての定義動作の集合.
$ActionType(a)$	アクション a についてのアクションタイプ.
$AntecedentNodeOf(p,n)$	LR 状態ネットワーク上のノード p について, n 回親をたどった時の先祖ノード.
$LeftHandSymbolOf(r)$	文法規則 r の左辺記号.
$Length(r)$	文法規則 r の右辺記号の数.
$DestinationStateNumber(a)$	アクション a が SHIFT または GOTO である場合の, 行き先の状態番号.
$RuleOf(a)$	アクション a が REDUCE である場合の, 文法規則.
$Scand$	仮のシフトアクションを行なうための候補 ((状態番号, 音素) の対) の集合.
$Rcand$	仮のリデュースアクションを行なうための候補 ((REDUCE するノードの親ノード, 文法規則, 音素), の 3 つ組) の集合.
N	音素数.

[先読みアルゴリズム]

1. given
   - string histroy node s
2. for each $p \in NetworkNodeOf(s)$
   - for each $n = 1....N$
     - for each $a \in ActionAt(StateOf(p), n)$
       (a) if $(ActionType(a) ==$ SHIFT) then
       $Scand = Scand + (DestinationStateNumber(a), n)$
       (b) if $(ActionType(a) ==$ REDUCE) then
       $Rcand = Rcand + (p, RuleOf(a), n)$
3. for each $(p, r, n) \in Rcand$
   (a) $q = AntecedentNodeOf(p, Length(r) - 1)$
   (b) for each $a \in ActionAt(StateOf(q), LeftHandSymbol(r))$
   if $(ActionType(a) ==$ GOTO) then
   - for each $a' \in ActionAt(DestinationStateNumber(a), n)$
     - if $(ActionType(a') ==$ SHIFT) then
     $Scand = Scand + (DestinationStateNumber(a'), n)$
     - if $(ActionType(a') ==$ REDUCE) then
     $Rcand = Rcand + (q, RuleNumber(a'), n)$
4. for each $(st, n) \in Scand$
   - for each $m = 1....N$
     - for each $a \in ActionAt(st, m)$
       * if $(ActionType(a) ==$ SHIFT
       or $ActionType(a) ==$ REDUCE
       or $ActionType(a) ==$ ACCEPT) then
       $PredictingContext(s) = PredictingContext(s) + (Phoneme(s), n, m)$

第 4 章

認識実験

1　実験条件

実験は、以下の条件のもとで行なった。

音声資料	国際会議に関する問い合わせタスク 12 対話, 701 文節.
音響分析	標本化 12kHz, フレーム周期 5 ms, ハミング窓 30 ms.
特徴量	1 〜 16 次 LPC ケプストラム, 1 〜 16 次 Δ LPC ケプストラム,log パワー, Δ log パワー.
HMnet	5240 単語の偶数番目の単語から学習, 状態数 600.
文脈自由文法	タスク依存型文節内文法 (mset), テストセット音素パープレキシティー 約 3.5.

2　特定話者文節音声認識実験

文節音声認識結果を表 4.1に示す。
表 4.1中の文節認識率を、図 4.1に示す。
図 4.1の横線は、 音素同期型 SSS-LR によるビーム幅 250 における文節認識結果である。
文節認識における消費 CPU タイムを図 4.2に示す。
図 4.2の横線は、ビーム幅 250 の音素同期型 SSS-LR で消費した CPU タイムと、リアルタイムである。

表 4.1: 各ビーム幅における文節認識率と消費 CPU タイム

ビーム幅	including			CPUtime
(個)	1	5	10	(ms)
50	84.7	87.7	87.7	800
100	90.7	94.6	94.7	1932
200	93.6	98.7	99.0	5064
400	93.9	99.3	99.7	13941
800	94.0	99.4	99.9	39117

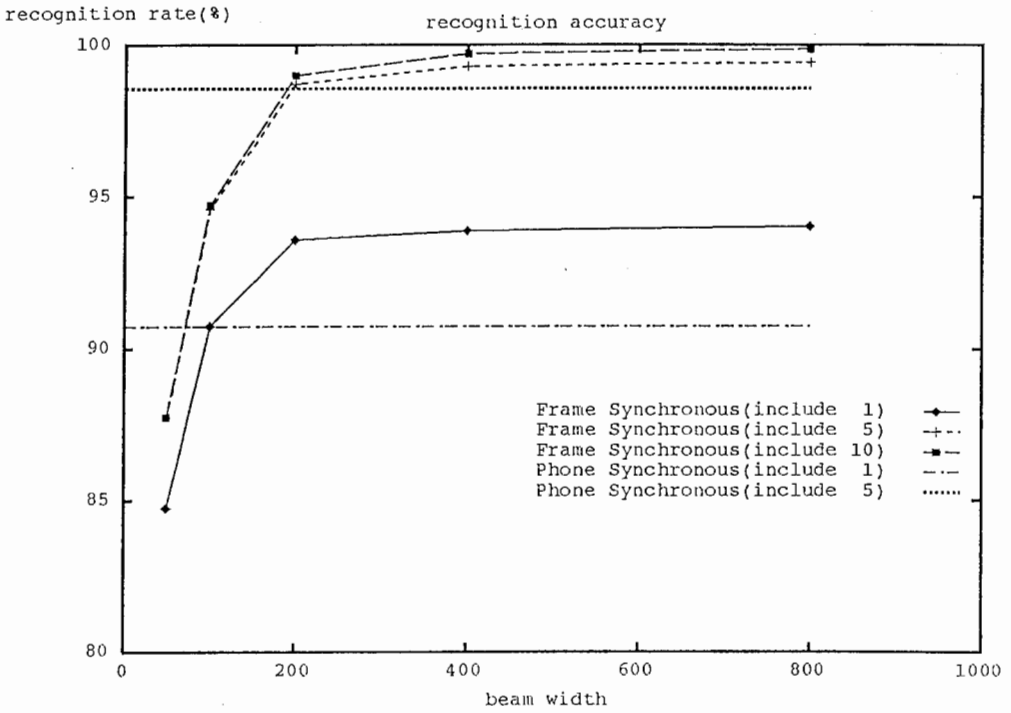

図 4.1: ビーム幅 vs. 文節認識率

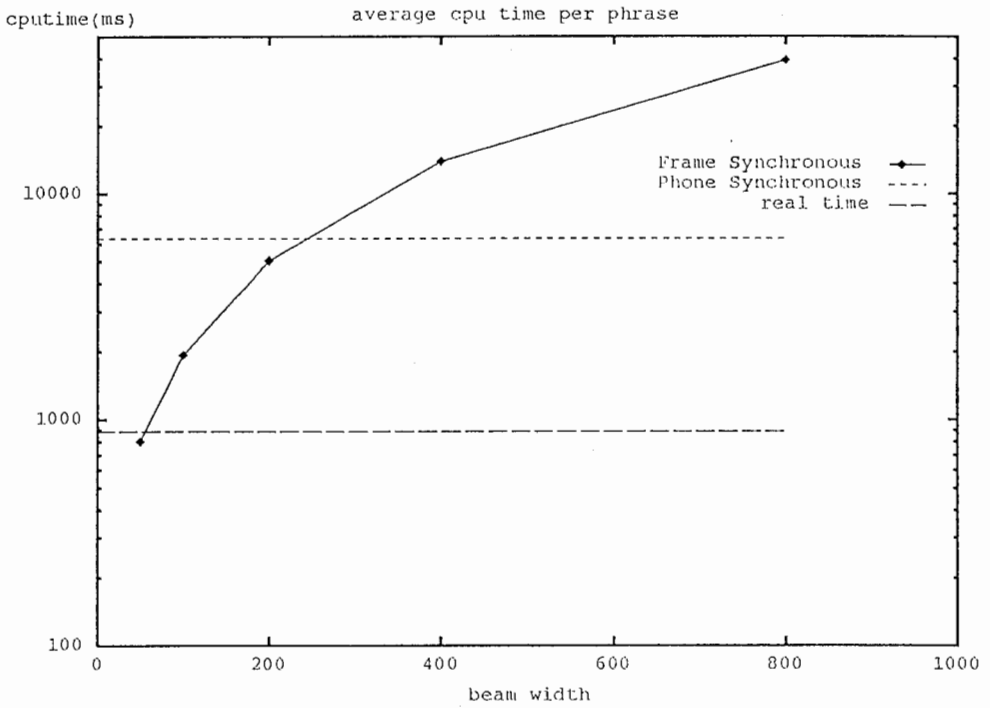

図 4.2: ビーム幅 vs. CPU タイム

3　特定話者文音声認識実験

特定話者文音声認識実験については、一部の実験しか行なっていない。従って全体の認識率及び CPU タイムを明示することはできない。実験結果の一例のみを紹介する。

1. 実験結果 1

正解文 = そちらは、会議事務局ですか。

(a) 誤認識例: 実験条件: ビーム幅 100、 20-best

```
>>> - - - - - - - - Result - - - - - - - - - - <<<
- - - - - - - candidate - - - - - - -
(1)notcompleted <= /-/-/s/o/ch/i/r/a/-/w/a/k/a/r/i/-/zh/j/u/u/sh/j/o/t/o/-/u/k/e/ts/u/k/e/g/a/(now.-)(next.-
aikjozhzudmgchngrshtssebqtwnph)
(2)notcompleted <= /-/-/s/o/ch/i/r/a/-/w/a/k/a/r/i/-/zh/j/u/u/sh/j/o/t/o/-/u/k/e/ts/u/k/e/g/(now.a)(next.-)
(3)notcompleted <= /-/-/s/o/ch/i/r/a/-/w/a/k/a/r/i/-/zh/j/u/u/sh/j/o/t/o/-/u/k/e/ts/u/k/e/g/a/-/(now.h)(next.u)
(4)notcompleted <= /-/-/s/o/ch/i/r/a/-/w/a/k/a/r/i/-/zh/j/u/u/sh/j/o/t/o/-/u/k/e/ts/u/k/e/g/a/-/(now.h)(next.o)
(5)notcompleted <= /-/-/s/o/ch/i/r/a/-/w/a/k/a/r/i/-/zh/j/u/u/sh/j/o/t/o/-/u/k/e/ts/u/k/e/g/a/-/(now.p)(next.-
aikjozhzudmgchngrshtssebqtwnph)
(6)notcompleted <= /-/-/s/o/ch/i/r/a/-/w/a/k/a/r/i/-/zh/j/u/u/sh/j/o/t/o/-/u/k/e/ts/u/k/e/g/a/-/(now.h)(next.ikzchshtssqtp
(7)notcompleted <= /-/-/s/o/ch/i/r/a/-/w/a/k/a/r/i/-/zh/j/u/u/sh/j/o/t/o/-/u/k/e/ts/u/k/e/g/a/-/(now.h)(next.a)
(8)notcompleted <= /-/-/s/o/ch/i/r/a/-/w/a/k/a/r/i/-/zh/j/u/u/sh/j/o/t/o/-/u/k/e/ts/u/k/e/g/a/-/(now.o)(next.ow)
(9)notcompleted <= /-/-/s/o/ch/i/r/a/-/w/a/k/a/r/i/-/zh/j/u/u/sh/j/o/t/o/-/u/k/e/ts/u/k/e/g/a/-/(now.ch)(next.-
aikjozhzudmgchngrshtssebqtwnph)
restgridatlasttime = 112
lastHMM - stategrid = 9
admissible(accept) = 0
- - - - - - - - - - - - - - - - - - - - - - - -
(0)ans > - - sochirawa - kaigizhimukjokudesuka - - -

recog > nonecandidate!!
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
NumofLRnetworknode : 431
Numofstringhistorynode : 205
NumofcreatedAction - link : 558
CPUtime = 2810, Elapsed = 3.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

(b) 正解認識例: 実験条件: ビーム幅 1600、 20-best

```
>>> - - - - - - - - Result - - - - - - - - - - <<<
- - - - - - - candidate - - - - - - -
(0)138548317(39.136log./frame) - / - /s/o/ch/i/r/a/w/a/ - /k/a/i/g/i/zh/i/m/u/k/j/o/k/u/d/e/s/u/k/a/ - / - / - /
(1)137060167(38.716log./frame) - / - /s/o/ch/i/r/a/w/a/ - /k/a/i/g/i/zh/i/m/u/k/j/o/k/u/d/e/s/u/g/a/ - / - / - /
restgridatlasttime = 1701
lastHMM - stategrid = 293
admissible(accept) = 2
- - - - - - - - - - - - - - - - - - - - - - - -
(0)ans > - - sochirawa - kaigizhimukjokudesuka - - -

recog > -/ - /s/o/ch/i/r/a/w/a/ - /k/a/i/g/i/zh/i/m/u/k/j/o/k/u/d/e/s/u/k/a/ - /
score > 138548317(13854.139005 : 39.13594log/perframe)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
NumofLRnetworknode : 5119
Numofstringhistorynode : 2867
NumofcreatedAction - link : 6061
CPUtime = 247280, Elapsed = 249.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

2. 実験結果 2

正解文 = 住所は、大阪市、北区、茶屋町、 23 です。

(a) 誤認識例: 実験条件: ビーム幅 200、 20-best

```
>>> - - - - - - - - Result - - - - - - - - - - <<<
- - - - - - - candidate - - - - - - -
(1)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /t/e/ts/(now.u)(next.zsph)
(2)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /t/e/ts/u/(now.z)(next. - aikjozhzudmgchngrshtssebqtwnph)
(3)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /(now.ts)(next. - aikjozhzudmgchngrshtssebqtwnph)
(4)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/u/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /t/e/(now.ts)(next. - aikjozhzudmgchngrshtssebqtwnph)
(5)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/u/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /(now.s)(next. - umgng)
(6)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/u/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /s/(now.u)(next.zsph)
(7)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /s/u/(now.z)(next. - aikjozhzudmgchngrshtssebqtwnph)
(8)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /t/e/ts/u/z/(now.u)(next.k)
(9)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /t/e/ts/u/z/u/(now.k)(next. - ikzhzdgchrshtsbqtnph)
(10)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /ts/(now.u)(next.u)
(11)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /s/u/z/(now.u)(next.k)
(12)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/ - /sh/i/(now.ts)(next. - aikjozhzudmgchngrshtssebqtwnph)
(13)notcompleted <= / - / - /zh/j/u/u/sh/j/o/w/a/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
```

```
/n/i/zh/j/u/u/s/a/ng/ − /s/u/z/u/(now.k)(next. − ikzhzdgchrshtsbqtnph)
(14)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /(now.ch)(next. − aikjozhzudmgchngrshtssebqtwnph)
(15)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /ch/(now.i)(next.ktsqtp)
(16)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /(now.s)(next.o)
(17)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /ch/i/(now.k)(next.a)
(18)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/(now.i)(next.k)
(19)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/(now.i)(next.aw)
(20)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/(now.i)(next.n)
(21)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /s/(now.o)(next.ow)
(22)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/(now.i)(next.g)
(23)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/(now.i)(next.chtsqtph)
(24)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /(now.s)(next.a)
(25)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/i/(now.k)(next.a)
(26)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/(now.i)(next.o)
(27)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/(now.i)(next.jm)
(28)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/(now.i)(next.dr)
(29)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /(now.s)(next.ijzhzrshtsep)
(30)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /(now.p)(next. − aikjozhzudmgchngrshtssebqtwnph)
(31)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /(now.sh)(next. − i)
(32)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/(now.i)(next.e)
(33)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /(now.k)(next. − ikzchshtsqtph)
(34)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /s/u/z/u/k/(now.i)(next.k)
(35)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /ts/u/(now.u)(next.ije)
(36)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /s/u/z/u/k/(now.i)(next.aw)
(37)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /s/u/z/u/k/(now.i)(next.chtsqtph)
(38)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /sh/i/ts/(now.u)(next.mn)
(39)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /sh/i/ts/(now.u)(next.r)
(40)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /s/u/z/u/k/i/(now.k)(next.a)
(41)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /s/u/z/u/k/(now.i)(next.jm)
(42)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/i/(now.d)(next.kjzhzdmgchngrshtssebqtwnph)
(43)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /s/(now.e)(next.gng)
(44)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /sh/i/ts/u/(now.r)(next.e)
(45)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /sh/(now.i)(next.ktsqtp)
(46)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /s/(now.a)(next.gng)
(47)notcompleted <= / − / − /zh/j/u/u/sh/j/o/w/a/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ − /ch/j/a/j/a/m/a/ch/i/ −
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ts/u/z/u/k/i/(now.n)(next.aoue)
restgridatlasttime = 172
lastHMM − stategrid = 47
admissible(accept) = 0
− − − − − − − − − − − − − − − − − − − − − − − −
(0)ans > − − zhjuushjowa − oosakashi − kitaku − chjajamachi − nizhjuusangdesu − − −

recog > nonecandidate!!
− − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − −
NumofLRnetworknode : 1332
Numofstringhistorynode : 650
NumofcreatedAction − link : 1619
CPUtime = 20190, Elapsed = 20.
− − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − − −
```

(b) 誤認識例: 実験条件: ビーム幅 3200、 20-best

```
>>> − − − − − − − −Result − − − − − − − − − − <<<
− − − − − − − −candidate − − − − − − − −
(0)555951715(44.438log./frame)−/−/zh/j/u/u/sh/j/o/w/a/−/i/q/t/e/−/o/o/s/a/k/a/sh/i/−/k/i/t/a/k/u/−/ch/j/a/j/a/m/a/ch/i/−
/n/i/zh/j/u/u/s/a/ng/d/e/s/u/ − / − / − /
(1)554968825(44.360log./frame) − / − /zh/j/u/u/sh/j/o/n/i/w/a/ − /i/q/t/e/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ −
/ch/j/a/j/a/m/a/ch/i/ − /n/i/zh/j/u/u/s/a/ng/d/e/s/u/ − / − / − /
(2)554839435(44.349log./frame)−/−/zh/j/u/u/sh/j/o/g/a/−/i/q/t/e/−/o/o/s/a/k/a/sh/i/−/k/i/t/a/k/u/−/ch/j/a/j/a/m/a/ch/i/−
/n/i/zh/j/u/u/s/a/ng/d/e/s/u/ − / − / − /
(3)554749233(44.342log./frame)−/−/zh/j/u/u/sh/j/o/j/a/−/i/q/t/e/−/o/o/s/a/k/a/sh/i/−/k/i/t/a/k/u/−/ch/j/a/j/a/m/a/ch/i/−
/n/i/zh/j/u/u/s/a/ng/d/e/s/u/ − / − / − /
(4)554656847(44.335log./frame)−/−/zh/j/u/u/sh/j/o/o/−/n/a/q/t/e/−/o/o/s/a/k/a/sh/i/−/k/i/t/a/k/u/−/ch/j/a/j/a/m/a/ch/i/−
/n/i/zh/j/u/u/s/a/ng/d/e/s/u/ − / − / − /
(5)554610916(44.331log./frame) − / − /zh/j/u/u/sh/j/o/w/a/ − /n/a/q/t/e/ − /o/o/s/a/k/a/sh/i/ − /k/i/t/a/k/u/ −
/ch/j/a/j/a/m/a/ch/i/ − /n/i/zh/j/u/u/s/a/ng/d/e/s/n/ − / − / − /
(6)553291329(44.226log./frame)−/−/zh/j/u/u/sh/j/o/w/a/−/i/q/t/e/−/o/o/s/a/k/a/sh/i/−/k/i/t/a/k/u/−/ch/j/a/j/a/m/a/ch/i/−
/n/i/zh/j/u/u/s/a/ng/ − /t/e/ng/d/e/s/u/ − / − / − /
```

```
(7)550558412(44.007log./frame)-/-/zh/j/u/u/sh/j/o/w/a/-/i/q/t/e/-/o/o/s/a/k/a/sh/i/-/k/i/t/a/k/u/-/ch/j/a/j/a/m/a/ch/i/-
/n/i/zh/j/u/u/s/a/ng/ - /t/e/ng/d/e/s/u/k/a/ - / - / - /
(8)550315402(43.988log./frame)-/-/zh/j/u/u/sh/j/o/w/a/-/i/q/t/e/-/o/o/s/a/k/a/sh/i/-/k/i/t/a/k/u/-/ch/j/a/j/a/m/a/ch/i/-
/n/i/zh/j/u/u/s/a/ng/ - /t/e/ng/d/e/s/u/n/e/ - / - / - /
restgridatlasttime = 2711
lastHMM - stategrid = 773
admissible(accept) = 9
- - - - - - - - - - - - - - - - - - - - - - - - -
(0)ans > - - zhjuushjowa - oosakashi - kitaku - chjajamachi - nizhjuusangdesu - - -

recog > -/ - /zh/j/u/u/sh/j/o/w/a/ - /i/q/t/e/ - /o/o/s/a/k/a/sh/i/ - /k/i/t/a/k/u/ - /ch/j/a/j/a/m/a/ch/i/ -
/n/i/zh/j/u/u/s/a/ng/d/e/s/u/ - /
score > 555951715(55592.391927 : 44.438378log/perframe)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
NumofLRnetworknode : 17794
Numofstringhistorynode : 10760
NumofcreatedAction - link : 21441
CPUtime = -2039564, Elapsed = 71054.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

4　考察

文節音声認識実験結果から、次のことがいえる。

1. 図 4.1より、ビーム幅を広げると、認識率は向上する。しかし、ビーム幅 200 程度で十分であり、それ以上広げてもさほど認識率の向上は望めない。
   ビーム幅 250 の音素同期型 SSS-LR に対し、フレーム同期型 SSS-LR での 1 位の認識率は、ビーム幅 100 が匹敵し、5 位までの場合ビーム幅 200 が匹敵する。

2. 図 4.2より、フレーム同期型ビームサーチの場合、ビーム幅を広げると消費する CPU タイムは極度に大きくなるものの、ビーム幅を 50 程度にすると、ほぼリアルタイムに並ぶ。
   ビーム幅 250 の音素同期型 SSS-LR に対し、フレーム同期型 SSS-LR が消費した CPU タイムが匹敵するのは、ビーム幅を 200 強に設定した場合である。

   以上の結果から、フレーム同期型 SSS-LR は、音素同期型 SSS-LR に比べ、認識パフォーマンスが高いといえる。

3. また、文節認識実験を通して以下のことに気がついた。

   N-best の N の値を大きくすれば、認識率の向上が見込まれると思われたが、ほとんどない。また、N の値を小さくすれば認識率は低下するものの、計算量が飛躍的に小さくなると考えられたが、試行実験の結果、逆に N の値がある程度大きい方が、実行時間が早いという結果が出た。
   これらは、スタックが併合された音素履歴の数が少なかったためで、特に実行時間については、N の値が小さい場合は枝刈の必要があるが、N の値が 10 程度で、ほとんど枝刈の必要がなくなる。従って、枝刈の処理の分だけ、数十 ms のオーダではあるが、N を大きくした方が早いという結果が出た。

文音声認識実験は、十分な実験は行なっていないが、実験を通して感じたことを述べる。

1. 文節音声認識に比べ、かなり大きなビーム幅を設定しなければ、正解認識が得られない。例え、一つの文章中の複数の文節を、個々に文節認識した場合に、ビーム幅 100 程度で十分であったものでも、文章として認識する場合、ビーム幅を 3200 まで広げなければ正解認識とはならない例もある。

2. 文認識の場合、文節間での仮説の展開数が大きいため、瞬間的に同様な grid ばかりがビーム内に残る。結果として、文末尾の助詞だけが異なるような、同様な認識結果ばかりを出力する。

3. 特に文認識において、ビーム幅が小さい場合、結果として出力可能な候補が一つもないこともある。

4. 音声フレームの早期の段階での認識誤りによって、文章全体の認識に影響を及ぼす場合がある。早期の認識誤りに引きづられ、文節の境界検出に失敗し、結果として、決定的な認識誤りを引き起こす。

第 5 章

むすび

本研究では、フレーム同期型 SSS-LR による連続音声認識について検討した。拡張 LR 構文解析法における、One-Pass サーチによる連続音声認識アルゴリズムについて検討し、同時に HMMnet を用いて認識精度の向上を図った。

認識実験を行ない、認識率と計算量について評価を行なった。

実験結果から、フレーム同期型 SSS-LR は、文節音声認識において、音素同期型 SSS-LR より、パフォーマンスが高いことが分かった。

文音声認識に対する今後の検討項目として、

- プログラムの高速化
  文音声認識において、フレーム同期型の場合、ビーム幅を 1600 から 3200 程度を用意しなければならない。しかし、現状のプログラムでは、ビーム幅を広げると極端に遅くなる。文認識の現実化のためにも、プログラムの改善が必要である。

- パラメータの調整
  本研究では、数種類のパラメータを固定して実験を行なった。今後は、いくつかのパラメータに自由をもたせ、厳密に実験を行なう必要がある。

- 継続時間制御の導入
  本研究では、文節初頭のポーズモデルを除いて継続時間制御を行なっていない。継続時間制御による認識率の向上と、処理量の増加の間のトレードオフにおいて、如何に妥協するかは難しい問題であるが、連続音声認識において、継続時間制御は極めて重要である。

- 文認識特有の問題
  文節認識に比べ、文認識で展開し得る仮説の数は膨大である。特に文節間では、瞬間的に仮説展開が爆発する。また、ポーズや冗長語を同時に扱うことを考えると、さらに複雑になる。こうした、文認識における問題にも対処する必要がある。

本稿では、音素同期型SSS-LR の欠点ばかりを述べてきたが、最後に、フレーム同期型 SSS-LR の欠点について述べる。

- left-to-right 探索の問題
  探索手順が入力音声に同期しているため、入力音声のある一部分での照合に問題が生じた場合、以降の探索に対して誤りをひきずる可能性がある。特に、入力音声の開始付近での照合の誤りは、認識全体に悪影響を及ぼす。音素同期の照合の場合は、可能な照合範囲に対して一律に照合するため、フレーム同期の照合に比べ比較的悪影響は少ない。

- 探索の終了時期の問題
  フレーム同期型の探索の場合、文法的な条件を一切考慮せずに、入力音声のフレームの終了によって探索が終了する。最終時刻で受理できる仮説が一つも存在しない場合は、認識結果を出力することができない。音素同期型の照合では、音素の数 (深さ) がある数に達した時点で探索を終了するが、文法的に受理できる仮説が残っていないとしても、さらに文法的に受理できる仮説が出現するまで、探索の深さを増やしていくことで、何らかの結果を必ず出力することができる。

謝辞

本実習に当たり、懇切丁寧な御指導と、激励をいただきました、Harald Singer 研究員、松永 昭一 主任研究員、匂坂 芳典 第一研究室室長、並びに ATR 音声翻訳通信研究所の皆様に感謝致します。また、本実習の機会を与えていただいた、山形大学 好田 正紀 教授、ATR 音声翻訳通信研究所 山崎 泰弘 社長に感謝致します。

参考文献

[1] K. Kita, T. Kawabata, and T. Hanazawa. Hmm speech recognition using stochastic language models. *The Journal of Acoustical Society of Japan*, Vol. 12, No. 3, pp. 99–105, 1991.

[2] A. Nagai, K. Kita, and S. Sagayama. Hmm-lr 法における音素文脈依存型 lr パーザの検討. 音講論, pp. 125–126, September 1990.

[3] H. Ney, D. Mergel, A. Noll, and A. Paeseler. A data-driven organization of the dynamic programming beam search for continuous speech recognition. In *Proc. ICASSP*, pp. 833–836, 1987.

[4] M. Okada. A unification-grammar-directed one-pass search algorithm for parsing spoken language. In *Proc. ICASSP*, pp. 721–724, 1991.

[5] M. Tomita. An efficient word lattice parsing algorithm for continuous speech recognition. *Proc. ICASSP*, pp. 1569–1572, 1986.

[6] 北研二, 川端豪, 斉藤博明. HMM 音韻認識と拡張 LR 構文解析法を用いた連続音声認識. 情処学論, pp. 472–478, March 1992.

[7] 中川聖一. 文脈自由文法のフレーム同期型構文解析法による連続音声認識. 信学技報, Vol. J70-D, No. 5, pp. 907–916, May 1987.