論文 / 著書情報
Article / Book Information

論題(和文)	WFST音声認識デコーダにおけるon-the-fly合成の最適化処理
Title(English)	
著者(和文)	大西 翼, ディクソン ポール, 岩野 公司, 古井貞熙
Authors(English)	大西 翼, ディクソン ポール, 岩野 公司, SADAOKI FURUI
出典(和文)	電子情報通信学会論文誌, Vol. J92-D, No. 7, pp. 1026-1035
Citation(English)	, Vol. J92-D, No. 7, pp. 1026-1035
発行日 / Pub. date	2009, 7
URL	http://search.ieice.org/
権利情報 / Copyright	本著作物の著作権は電子情報通信学会に帰属します。 Copyright (c) 2009 Institute of Electronics, Information and Communication Engineers.

論　文

WFST音声認識デコーダにおけるon-the-fly合成の最適化処理

大西　翼[†]　ディクソン　ポール[†]　岩野　公司[†*]　古井　貞熙[†]

Optimization of On-the-Fly Composition for WFST-Based Speech Recognition Decoders

Tasuku OONISHI[†], Paul DIXON[†], Koji IWANO[†*], and Sadaoki FURUI[†]

あらまし　重み付き有限状態トランスデューサ（WFST）を用いた音声認識は，柔軟で高性能な音声認識を実現できる反面，認識時のメモリ消費量やモデル変更時のオーバヘッドの増加などの問題がある．このため認識時にWFSTの合成演算を行うことで，動的に探索ネットワークを生成するon-the-fly合成のアプローチが提案されている．これにより，認識時のメモリ消費量の削減やモデル変更時のオーバヘッドの削減が可能となる一方で，認識速度が低下するという問題がある．過去にはon-the-fly合成時にネットワークの最適化処理を行うことで高速に音声認識を行う手法が提案されている．この手法では合成するWFSTの構造に制約を与えることによって最適化処理を実現しているため，すべてのWFSTで手法を利用することは困難であった．そこで，本論文では任意の構造のWFSTに対して，filterと呼ばれるWFSTを利用することで同様の最適化処理が実現可能であることを示す．本手法を実装したデコーダによる性能評価実験において，従来手法と比べてより効率的な探索が可能であることが確認され，本手法の有効性が示された．

キーワード　重み付き有限状態トランスデューサ，on-the-fly合成，最適化演算，音声認識デコーダ

1. まえがき

近年，重み付き有限状態トランスデューサ（WFST）を用いた音声認識のパラダイムが提唱され，従来のWFSTを利用しない音声認識デコーダに比べて，様々なモデルを容易に扱うことができる高速・高精度な音声認識デコーダが実現可能であることが報告されている[1],[2]．我々は，次世代の音声認識デコーダの実現を目指し，WFSTを利用した音声認識デコーダ「T^3（Tokyo Tech Transducer-based）decoder」の開発を行っている[3]．

WFST音声認識では，認識で利用するすべてのモデルをWFSTの形式で表現し，それらを合成演算により一つのWFSTに合成する．合成後のWFSTに様々な最適化演算を施すことで，探索ネットワークが最適化され，高速・高精度な音声認識を実現することができる．しかし，すべてのWFSTを一つに合成するため，合成されるWFSTが巨大化し，認識時に大きなメモリ量が必要となる．また，一部のモデルを変更する際でも，探索ネットワークを再構築するためのオーバヘッドが発生するといった問題がある．これらの問題を解決するために，認識時に合成演算を行うことで，動的に探索ネットワークを構築する「on-the-fly合成」が提案されている[4]．on-the-fly合成では，すべてのWFSTの状態を一度に生成するのではなく，探索に必要な状態だけを部分的に生成するため，探索時の消費メモリ量を大幅に削減することができる．また，変更したいモデルを認識時にon-the-flyで合成することで，探索ネットワークの再構築が不要となり，モデル変更に伴うオーバヘッドを削減することができる．

on-the-fly合成をデコーダに実装する方法として，大きく二つの方式が考えられる．一つは，動的に合成される部分的なWFSTを実際に生成して，それを直接参照して探索を進める方式[5]であり，もう一つは，合成結果のWFSTは生成せず，仮説を表すトークン中に合成元のWFSTの状態のポインタを保持しておき，探索時に仮説を展開する際，合成元のWFSTを参照しながら，合成演算を行った場合と同等のスコア

[†] 東京工業大学大学院情報理工学研究科計算工学専攻，東京都
Tokyo Institute of Technology, 2–12–1 Ookayama, Meguro-ku, Tokyo, 152–8552 Japan
[*] 現在，東京都市大学環境情報学部情報メディア学科

を与えるようにリスコアリング処理を行うことで探索を進める方式[6]〜[8]である．

前者の方式は，合成結果にあたる WFST を実際に生成していることから，後者の方式に比べると計算量やメモリ消費量が大きいという欠点がある．しかし，得られた合成結果の WFST に対して，更に別の演算を適用することで実現される高度な処理（例えば，多段にわたる on-the-fly 合成[9] や projection 演算の適用が必要なクラス言語モデルを利用したデコーディング[9] など）をデコーダに実装しようとする場合，導入したい演算の定義を探索ネットワークの生成を行うプログラムモジュールに組み込み，それらの演算を生成された WFST に適用するように処理の流れを変更するだけでよく，このような機能拡張を容易に行うことができる．所望の演算が既にモジュールとして作成され，存在している場合も多く，その場合には，大きな変更を加えることなくそのモジュールを組み込むことができるので，デコーダ開発者の負担が著しく小さい．

一方，後者の方式は，WFST 生成にかかわる計算コストを削減できる利点を有している．しかし，上述のような機能拡張を行う場合には，新たに導入する演算処理と等価となるようにリスコアリング処理のアルゴリズムを再構成し，それに従って探索に関するプログラムモジュールを書き換える必要がある．組み込みたい演算が既にモジュールとして存在している場合でも，その内容を解釈した上でリスコアリング処理のアルゴリズムに反映させる必要があり，開発者にとって負担が大きい．我々は，様々な機能を有する音声認識デコーダの開発を目指していることから，開発者にとって利点の大きい前者の Caseiro らの方式[5] に従って，on-the-fly 合成の実装を行っている．

on-the-fly 合成では，動的に合成される WFST が部分的なものであることから，WFST 全体の状態を考慮したネットワークの最適化処理を行うことが難しく，探索空間の最適化が不十分となって認識性能が劣化する可能性がある．この問題に対処するため，Caseiro らは，動的に無駄な状態の生成を回避する処理（dead-end 状態の回避処理）や動的 pushing 処理（dynamic pushing）など，on-the-fly 合成で実現可能な WFST の最適化処理手法を提案し，高速・高精度な音声認識を実現している[5]．しかし，Caseiro らの手法では合成元の WFST の構造に制約が必要であり，任意の構造をもつ WFST に対して適用できないという問題があった．

そこで本論文では，「filter [1], [10]」と呼ばれる WFST を on-the-fly 合成時に利用した，構造制約を必要としない，一般化された最適化処理手法（dead-end 状態の回避処理と dynamic pushing）の提案を行う．filter は本来，WFST 中に含まれる冗長な状態遷移（パス）を除去することを目的として提案されたものであるが，これまでに on-the-fly 合成における動的な最適化処理に利用された例はなかった．提案手法では，on-the-fly 合成時に filter の合成も併せて行うことによってパスの制御された WFST を動的に生成し，得られた WFST に対して，filter の状態番号を利用して，効率的に最適化処理（dead-end 状態の回避処理と dynamic pushing）を行う．filter 自体は任意の構造をもつ WFST と合成することが可能であるため，提案手法により，合成元の WFST の構造に制約のない動的な最適化処理が実現できる．提案手法を実装することにより，我々の開発している「T^3 decoder」は，「任意の構造をもつ WFST を対象とした最適化処理つきの on-the-fly 合成」が可能となり，汎用性の高い，高速・高精度の音声認識デコーダとなる．

本論文では，まず **2.** にて，WFST 音声認識の概要を述べる．次に，**3.** にて，WFST における合成演算の詳細について述べる．**4.** にて on-the-fly 合成の詳細について述べる．**5.** では，Caseiro らにより提案されている演算と本論文で提案する一般化手法について述べる．**6.** では，多段階の on-the-fly 合成について述べる．**7.** にて，Caseiro らの手法と本手法の比較実験を行い，本手法の有効性を示す．

2. WFST 音声認識

WFST を利用した音声認識デコーダでは，まず音響モデルや言語モデル，単語発音辞書などのモデルをそれぞれ個別に WFST の形式で表現する．それらの WFST に合成演算を施すことで，すべてのモデルの情報を組み込んだ一つの WFST を合成する．更に，合成された WFST に「決定化」や「最小化」「pushing」などの演算を施すことにより，探索ネットワークの最適化を行うことができる．デコーダは，事前に生成されたネットワークを探索することで音声認識を実現する．代表的な大語彙連続音声認識を例にすると，探索ネットワークは以下のような四つの WFST を合成することで構築される．

- H：HMM の状態から文脈依存音素への WFST
- C：文脈依存音素から文脈非依存音素への WFST

- L：文脈非依存音素から単語への WFST
- G：単語から単語 N-gram への WFST

合成演算を $\circ$ で表現すると，すべての WFST を合成した WFST は，以下のように表現される．

$$H \circ C \circ L \circ G \tag{1}$$

3. 合成演算

WFST L, R に，それぞれ x, y を入力したときの出力を y, z とする．このとき，合成演算により合成された WFST「$L \circ R$」は，x を入力すると z を出力する WFST となる．

合成演算により生成された状態は，L と R の状態 q_l, q_r を用いて (q_l, q_r) のように表現される．状態 q_l からの状態遷移を，開始状態，到達状態，入力シンボル，出力シンボル，出力重みのパラメータを用いて $(q_l, q_l^{'}, i_l, o_l, w_l)$ と表現し，状態 q_r の状態遷移を $(q_r, q_r^{'}, i_r, o_r, w_r)$ と表現する．この場合に，合成後の状態 (q_l, q_r) から遷移する先の状態としては $(q_l^{'}, q_r), (q_l, q_r^{'}), (q_l^{'}, q_r^{'})$ の三つが考えられる．これらの状態への状態遷移は，それぞれ以下の規則を満たした場合に生成される．なお，下記のシンボル ϵ は，シンボルを入力または出力せずに遷移を行う null 遷移を行うためのシンボルである．

（1） $o_l = \epsilon$ であれば，$((q_l, q_r), (q_l^{'}, q_r), i_l, \epsilon, w_l)$ を生成．この条件により合成される状態 $(q_l^{'}, q_r)$ を「ϵ 出力により合成される状態」と呼ぶ．

（2） $i_r = \epsilon$ であれば，$((q_l, q_r), (q_l, q_r^{'}), \epsilon, o_r, w_r)$ を生成．この条件により合成される状態 $(q_l, q_r^{'})$ を「ϵ 入力により合成される状態」と呼ぶ．

（3） $o_l = i_r$ であれば（ただし，シンボルが ϵ の場合を除く），$((q_l, q_r), (q_l^{'}, q_r^{'}), i_l, o_r, w_l + w_r)$ を生成．この条件により合成される状態 $(q_l^{'}, q_r^{'})$ を「シンボルマッチにより合成される状態」と呼ぶ．

4. on-the-fly 合成

合成演算において，ある状態 (q_l, q_r) から次に遷移する先の状態は，その状態に隣接する合成元の状態 $q_l, q_r, q_l^{'}, q_r^{'}$ の情報のみを利用して実行される．したがって，探索に必要な状態だけを部分的に合成する「on-the-fly 合成」[4] が可能となる．H, C, L の WFST を事前に合成し，G の WFST を on-the-fly 合成することを，以下のような表記で表す．

$$\{H \circ C \circ L\} \circ G \tag{2}$$

この場合には H, C, L を事前に合成した WFST $H \circ C \circ L$ 及び G に対して，決定化や pushing，最小化などの最適化処理が行われる．

5. on-the-fly 合成の最適化

前述のように，Caseiro ら [5] は，on-the-fly 合成時に利用可能な最適化処理として，無駄な状態の生成を回避する「dead-end 状態の回避処理」，重みの先読みを行う「dynamic pushing」を提案している．我々は，これらの最適化について，任意の構造の WFST に対して適用できる手法を提案し，デコーダへの実装を行っている．ここでは，それぞれの最適化手法について，まず先行研究における最適化手法を説明し，次に我々の最適化手法について述べる．

5.1 dead-end 状態の回避処理

dead-end 状態とは，「非最終状態であり，かつ次に遷移する状態が一つも存在しない状態」である．このような状態は，最終状態への最適パスの探索には不要となるため，探索ネットワーク上に存在しないことが望ましい．図 1 に，合成後の WFST 中に dead-end 状態が生成される例を示す．図中の状態遷移上の記号 $x : y/z$ は，それぞれ，入力シンボル，出力シンボル，出力重みを表している．出力重みが省略されている場

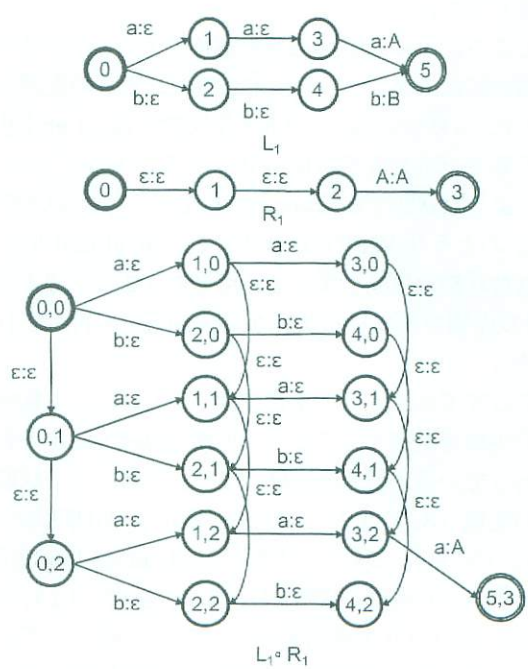

図 1 合成演算で dead-end 状態が生成される例
Fig. 1 An example showing dead-end states generated in WFST composition.

合は 0 であることを表す．ここでは，WFST L_1, R_1 から合成演算により $L_1 \circ R_1$ を合成しており，$L_1 \circ R_1$ 中の状態 (4,2) が dead-end 状態である．状態 (4,2) が dead-end となるのは，**3.** に挙げた三つの生成条件を満たす遷移先の状態が存在しないためである．

このような dead-end 状態の生成を on-the-fly 合成時に回避するためには，状態 (q_l, q_r) を合成する際に，L 中の状態 q_l から将来出力されるシンボルを事前に調べ，そのシンボルと R 中の入力シンボルとの照合を行う必要がある．

5.1.1 Caseiro らの手法

Caseiro らの on-the-fly 合成における dead-end 状態の回避処理では，L の状態 q_l から遷移するパスの中で，最初に出力されるシンボルの集合を先読みによって求め（先読みシンボル集合），そのシンボル集合と，q_r の直後の状態遷移(注1)の入力シンボル集合との積集合を調べている．積集合が空集合になる場合は，dead-end 状態に到達すると判定し，その状態の生成を回避している．このアルゴリズムにより，状態 (4,2) について判定を行うと，L_1 中の状態 4 の先読みシンボル集合は {B},R_1 中の状態 2 の直後の状態遷移の入力シンボルは {A} となり，積集合が空集合となることから dead-end と判定される．同様にして，状態 (2,2) のような，dead-end に至る無駄な状態の生成も回避することができる．

ところが，q_r の直後の状態遷移の入力シンボルが ϵ の場合には，更にその先に生じるシンボルの先読みは行われないため，(4,1) のような状態が dead-end 状態に至るかどうかを合成時に判定することができない．そのような状態で dead-end の判定が行われない場合，図 2 のような WFST が合成され，dead-end に至る無駄な状態の生成をほとんど回避することができない（図の点線で囲まれた状態が dead-end と判定された状態）．

そこで Caseiro らの手法では，ϵ 入力による状態遷移の生成を制御することで，早期の dead-end の判定を行っている．具体的には，(1) ϵ 入力による状態遷移の生成（**3.** の（2）の合成）を q_l が初期状態のときにのみ行い，これらの状態では dead-end の判定を行わず，(2) 残りの状態では，**3.** で述べた（1），(3）の合成規則のみを適用することで，上述したアルゴリズムによる dead-end の判定を行っている．この手法により，合成された WFST を図 3 に示す．この手法では，正しい WFST を合成するために，合成元の

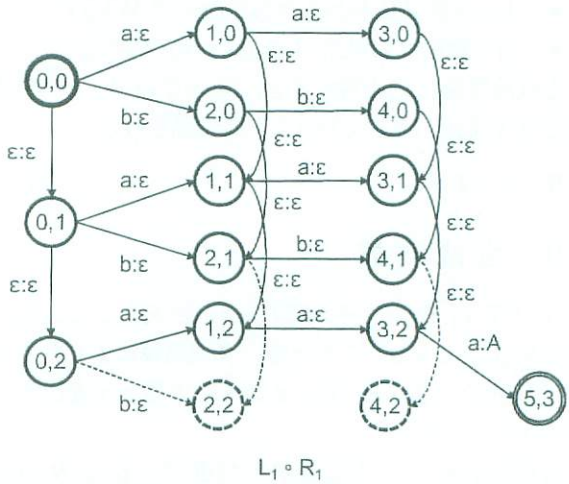

図 2　null 遷移のために多くの無駄な状態が合成される例
Fig. 2　An example showing useless states which occur because of the null transitions.

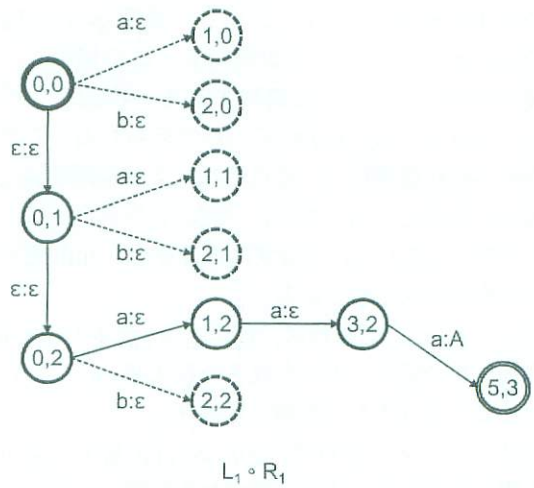

図 3　Caseiro らの手法 [5] により合成された WFST
Fig. 3　A WFST composed by the Caseiro's method [5].

WFST L が以下の構造制約を満たしている必要がある [5].

（1）　ループを構成する状態遷移は最終状態から初期状態に到達する状態遷移のみ

（2）　初期状態から最終状態に至るパスで非 ϵ の出力シンボルはただ一つ

図 1 の WFST もこの構造制約を満たしていることから，正しく dead-end 状態の回避処理が行われる．

(注1)：on-the-fly 合成では，WFST L は認識に先立って最適化や最終状態に至るシンボルの先読みを行うことができるが，WFST R は動的に合成される可能性があるため，このような事前の処理を行わないことが前提となっている．したがって，R については，直後の状態遷移の情報のみしか用いていない．

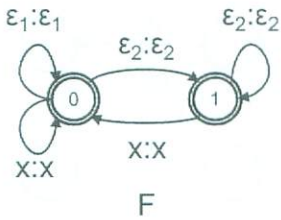

図 4 filter を表す WFST
Fig. 4 The filter WFST.

5.1.2 提案手法

我々は，「filter [1], [10]」と呼ばれる特殊な WFST を on-the-fly 合成時に利用することで，任意の構造をもつ WFST に適用可能な dead-end 状態の早期回避手法を提案する．

もともと filter を用いた合成演算は，L 中の ϵ 出力による状態遷移と，R 中の ϵ 入力による状態遷移により発生する，合成後の WFST 上の冗長なパスを除去するために利用されている．(例えば，図 1 では，初期状態から，シンボルマッチにより生成された最終状態 (5,3) に至るまでに，複数の冗長なパスが存在している．) 冗長なパスは，シンボルマッチによる状態遷移が発生する前に，ϵ 出力による状態遷移が発生した後で，ϵ 入力による状態遷移が発生する場合に生成される．このような遷移の発生を回避するために，図 4 に示す filter (F) が利用される．filter を用いた合成演算では，合成する WFST の L の出力シンボルの ϵ を ϵ_2 に，R の入力シンボルの ϵ を ϵ_1 に置き換え，更に L, R の各状態に ϵ_1, ϵ_2 の自己ループを追加した WFST をそれぞれ L', R' とし，$L' \circ F \circ R'$ という合成を行う．

filter 中の状態 0 は R の ϵ 入力またはシンボルマッチによって状態が生成されたことを，状態 1 は L の ϵ 出力によって状態が生成されたことを表している．シンボル x は，任意の非 ϵ シンボルを表していることから，シンボルマッチングにより状態が合成されると filter の状態は 0 に遷移する．ここで，filter の状態 1 からの状態遷移には ϵ_1 を受け付ける遷移が存在しないことから，$L' \circ F \circ R'$ により生成された WFST 中では，初期状態 (あるいはシンボルマッチングにより生成された状態) から，一度 ϵ 出力による状態遷移が生成されると，以降シンボルマッチングにより状態が生成されるまでは，ϵ 入力による状態遷移が生成されなくなる．このようなパスの合成を行うことで，冗長なパスの生成を防ぐことができる．

我々の提案する dead-end 状態の回避処理は，filter によって状態遷移が制御された WFST 上で行う．filter によって合成された状態は，L, F, R 中の状態番号を用いて (q_l, q_f, q_r) と表現される．この状態について，q_f に応じて，以下のように dead-end の判定を行う．

- $q_f = 1$ のときには，以降のパスでシンボルマッチによる状態遷移が発生する前に ϵ 入力による状態遷移が発生しない．したがって，q_l の先読みシンボル集合と，q_r の入力シンボル集合の積集合により dead-end の判定を行う．
- $q_f = 0$ で，かつ，q_r に ϵ 入力による状態遷移が存在しないときには，以降のパスでシンボルマッチによる状態遷移が発生する前に ϵ 入力による状態遷移が発生しない．したがって，$q_f = 1$ と同様の処理により dead-end の判定を行う．
- $q_f = 0$ で，かつ，q_r に ϵ 入力による状態遷移が存在するときには dead-end の判定を行わない．

filter の合成は，任意の構造をもつ WFST に対して適用が可能であり，特別な構造制約を必要としていない．提案手法では，このようにして得られた合成後の WFST に対して，filter の状態番号を利用して処理を行うため，合成元の WFST の構造制約を考慮することなく dead-end 状態の回避を行うことができる．

提案手法による dead-end 状態の回避の様子を，例で説明する．まず，図 1 に示した (Caseiro らが想定する構造制約を満たしている) WFST の合成に対して，提案手法による dead-end 状態の回避処理を行った結果を図 5 に示す．これから，Caseiro らの手法による結果と同様に，早期に dead-end 状態の生成が回避されていることが分かる．

次に，図 6 のような WFST L_2, R_2 を合成する場合を考える．L_2 は Caseiro らが想定している構造制約 (2) を満たしていない．Caseiro らの dead-end 状態の回避手法では，L の初期状態を対象とした合成を行う場合にのみ ϵ 入力による状態遷移の生成を許すように制限されているため，この例の WFST に適用すると，本来生成されるべき状態 (1,2) が生成されず，結果的に図 6 の下にあるような，最終状態に至るパスが存在しない WFST が生成されてしまう．一方，提案手法を用いた場合にはそのような問題は生じず，図 7 のように，dead-end 状態が回避された，正しい WFST が合成される．

なお，図 7 の状態 $(2, 0, 1)$ も dead-end 状態であるが，提案手法を適用した場合でも残っており，dead-end 状態の生成を完全に回避することができていない．

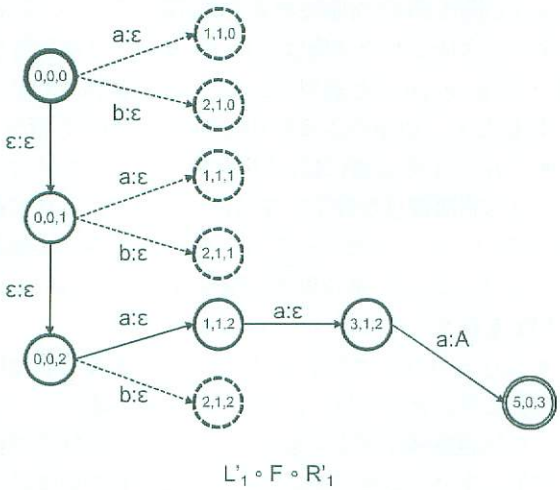

図 5 filter を用いた dead-end 状態の回避処理により合成された WFST ($L_1 \circ R_1$)

Fig. 5 WFST ($L_1 \circ R_1$) composed with dead-end state operation using a filter.

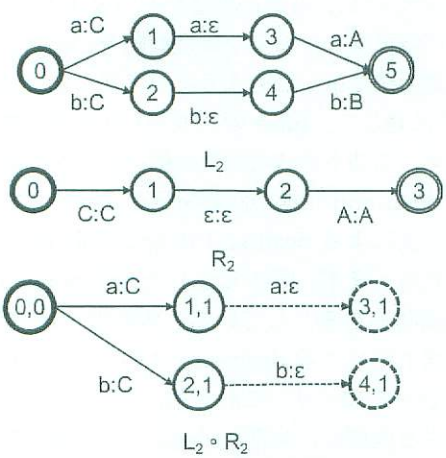

図 6 Caseiro らの dead-end 状態の回避手法 [5] を適用した場合に正しい WFST が合成されない例．L_2 は，Caseiro らの手法の構造制約を満たさない．

Fig. 6 An example of incorrect WFST composed by using the Caseiro's dead-end state avoiding operation [5]: L_2 does not satisfy the structural restrictions for the Caseiro's method.

これは，このような状態（提案手法における判定処理の 3 番目の条件に合致する状態）の生成時には，それより先の状態が dead-end であるかどうかを判断できず，判定を先延ばししているためである．

5.2 dynamic pushing

pushing とは，初期状態に近い状態に重みを再配置する演算である．これにより，WFST の重みを探索の早期に利用することが可能となるため，探索の効率化が実現できる．pushing 演算では，まず各状態に対して，その状態から最終状態への最短パス重みを「先読みスコア」として設定する．次に，各状態遷移の出力重みに，開始状態と到達状態の先読みスコアの差を加えることで，初期状態に重みを近づける操作を行う．しかし，on-the-fly 合成では，ある状態から最終状態への最短パス重みを求めることは難しい．そのため，on-the-fly 合成時の先読みスコアの決定手法とそれによる pushing 演算（dynamic pushing）が提案されている [5]．

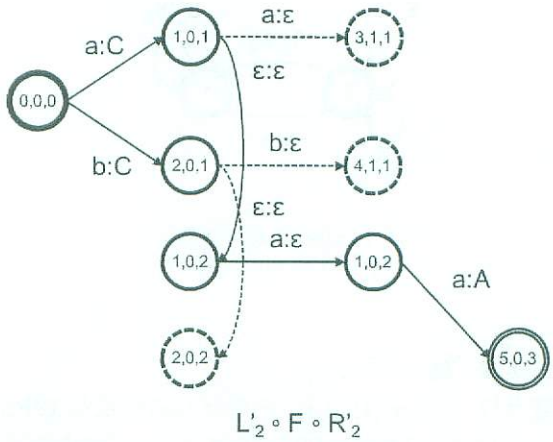

図 7 filter を用いた dead-end 状態の回避処理により正しく合成された WFST ($L_2 \circ R_2$)

Fig. 7 A WFST ($L_2 \circ R_2$) produced by correctly avoiding dead-end states using filter composition.

5.2.1 Caseiro らの手法

Caseiro らの手法 [5] では，ある状態 (q_l, q_r) の生成にあたり，q_r の直後の状態遷移の入力シンボルとのマッチングによって将来生成される状態遷移の出力重みを，できるだけ早い段階で利用することを目的としている．このため，ある状態 (q_l, q_r) の先読みスコアを以下の基準により決定する．

（1） ϵ 出力により合成された状態では，q_r の直後の状態遷移の中から，入力シンボルが q_l の先読みシンボル集合に含まれるものを選び，その中の最小の出力重みを先読みスコアとして設定する．

（2） それ以外の状態では，先読みスコアとして 0 を設定する．

しかし，この基準で先読みスコアを決定すると，先読みスコアが一意に決まらない状態が発生する．その例を図 8 に示す．図の各状態上の [ ] 内の数字は，先読みスコアを表す．このとき，図の状態 (2,0) では，(0,0) から遷移先状態を生成した場合には基準 (1) が適用され，(1,0) から遷移先状態を生成した場合には基準 (2) が適用されることから，先読みスコアが一意に

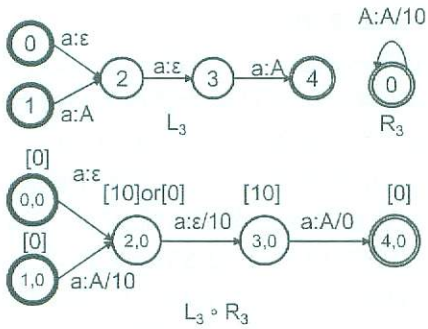

図 8 dynamic pushing の処理で先読みスコアがあいまいになる WFST の例
Fig. 8 An example of WFST composition in which the look ahead score cannot be uniquely determined.

決定されない．このような問題は，ϵ 出力により合成された状態とそれ以外の状態が同一の状態となる場合にのみ発生する．[5] では，このような問題が発生するのは，L の構造制約上，L の後半部分（各パスについて非 ϵ シンボルを出力してから最終状態に至るまで）の状態を合成に利用した場合に限定される．したがって，その部分を合成するときに，先読みスコアをすべて 0 とすることで，先読みスコアのあいまい性を回避している．

5.2.2 提案手法

filter を用いた合成演算では，ϵ 出力により合成された状態とそれ以外の状態は，それぞれ別の状態として合成される．したがって，それらの状態が同一の状態となることはないので，上述した基準をすべての状態について適用しても，先読みスコアがあいまいになる状態が発生しない．図 9 に，filter を用いた dynamic pushing により，合成される WFST を示す．先の WFST で先読みスコアがあいまいとなった状態 (2,0) が，(2,1,0), (2,0,0) のように区別されている．このように filter を合成した場合，生成される状態数は若干増加するが，すべての状態で先読みスコアを一意に決定することが可能となる．

なお，filter としては，本論文で取り上げた 2 状態の filter のほかに，図 10 の 3 状態の filter が提案されている [1]．この 3 状態の filter が 2 状態の filter と異なるのは，状態 0 の自己遷移（$\epsilon_2 : \epsilon_1$）により，ϵ 出力シンボルと ϵ 入力シンボルのマッチングで合成される状態（状態遷移）の生成を許している点である．この状態遷移の生成により，合成結果の WFST では，2 状態の filter を用いた場合に比べて，最終状態に至る最短経路のパス長がより短くなる．一方，合成後の WFST

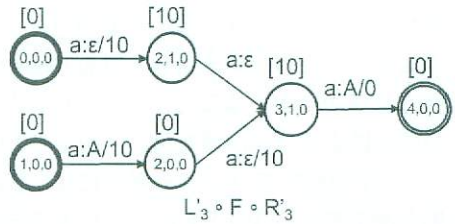

図 9 filter を用いて dynamic pushing の処理を行った WFST の例
Fig. 9 The WFST produced by dynamic pushing using filter composition.

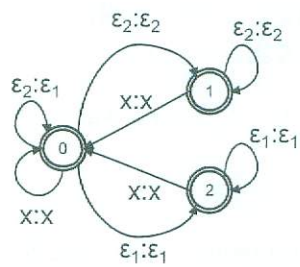

図 10 3 状態 filter
Fig. 10 The filter with 3 states.

の各状態において，最適化処理を即座に行うことができる「それ以降のパスで ϵ 入力による状態遷移が生起しないことが保証される状態（どちらの filter でも状態番号 1 が割り当てられる状態）」が，上記の状態（状態遷移）の生起を許すことで，2 状態の filter を用いる場合に比べて相対的に減ってしまい，最適化の効率が落ちることから，実時間で動作させる場合に認識性能が劣化する可能性がある．実際に，我々は 3 状態の filter を用いた場合の on-the-fly 合成の最適化処理の実装も行い，その性能評価を行っている [11]．その結果，2 状態 filter の方が良好な認識性能となったことから，本論文では 2 状態 filter を用いた場合のみについて説明を行うこととする．

6. 多段階の on-the-fly 合成

合成演算には結合法則が成り立つため以下の二つの WFST は等価となる．ここで X, Y, Z は，それぞれ WFST を表す．

$$(X \circ Y) \circ Z \tag{3}$$

$$X \circ (Y \circ Z) \tag{4}$$

我々の合成手法では，WFST L, R を合成する際には，事前に L の先読みシンボル集合を計算する必要がある．そのための順序で合成する場合には，$X, (X \circ Y)$ の二つの WFST について，先読みシンボル集合を求

める必要がある．しかしながら，WFST $(X \circ Y)$ は，動的に合成されるため，将来合成される状態の出力シンボル集合を事前に求めることができない．したがって，この順序による合成演算を実行することができない．一方，(4) の順序で合成する場合には，先読みシンボル集合を計算する WFST は X, Y となる．これらの WFST の先読みシンボル集合は，事前に計算することができるため合成演算を実行することができる．以上から，我々の方式では多段階の WFST X_1, X_2 $\cdots$, X_{n-1}, X_n の合成を行う場合には，以下の合成演算の順序により実現している．

$$X_1 \circ (X_2 \circ (\cdots \circ (X_{n-1} \circ X_n) \cdots)) \tag{5}$$

7. 実　　験

提案した on-the-fly 合成の最適化処理の効果を検証するため，日本語話し言葉コーパス（CSJ：Corpus of Spontaneous Japanese）を用いた，大語彙連続音声認識により性能評価実験を行う．

7.1 想定する認識タスク

on-the-fly 合成には，個々のモデルの切換が容易になるという利点がある．どのモデルの WFST を事前に用意し，どのモデルの WFST を動的に切り換えるかは，認識タスクに依存する．我々のデコーダでは，様々な利用要求に対処できるように，複数のモデルを個別に切り換えることが可能な，多段の on-the-fly 合成を実装している．この機能を用いて，「発音辞書 (L) と言語モデル (G) を個別に切り換える」タスクを応用例として想定し，実験を行う．この場合，合成される WFST を，

$$\{H \circ C\} \circ L \circ G \tag{6}$$

と表す．括弧は認識に先立って事前に合成・最適化を行っていることを意味している．この場合には，事前に合成した $H \circ C$ を L 及び G で on-the-fly 合成することを表している．

なお，比較対象となる Caseiro らの方法 [5] で on-the-fly 合成を行う場合には，ネットワークの構造に制約が必要である．ここでは，L のみがその構造制約を満たしているため，第 1 段階の $L \circ G$ の合成にのみ手法を適用した．$\{H \circ C\}$ と $L \circ G$ を合成する場合には，構造制約を満たしていないため，dead-end 状態の回避処理と dynamic pushing を行わない，通常の合成演算を適用した．

7.2 実験条件

学習用データとして，音響モデルには 967 学会講演，言語モデルには学会講演と模擬講演 2,682 講演を用いた．評価データには，CSJ のテストセット 1 の 10 講演を用いた．音響特徴量には，フレームシフト 10 ms，分析窓幅 25 ms の MFCC12 次元+ΔMFCC12 次元+ΔΔMFCC12 次元+対数パワー+Δ 対数パワー+ΔΔ 対数パワーの計 39 次元を用いた．音響モデルには，3,000 状態 32 混合の triphone HMM を用い，言語モデルには語彙サイズ 65,000 単語の trigram を用いた．デコーダには，我々が開発している T^3 decoder を使用した．実験には，2.4 GHz CPU（Intel Core2 Duo），2 GByte メモリの計算機を使用した．

7.3 実験結果

提案する最適化手法と，Caseiro らの最適化手法 [5] により on-the-fly 合成を行った際に生成される WFST の状態数 (#states) と状態遷移数 (#arcs) の比較を，表 1 に示す．表 1 から $L \circ G$ を合成した場合については，提案手法の方が Caseiro らの手法と比べて生成された状態数・状態遷移数が若干多いことが分かる．これは，filter を用いることによる状態数の増加に起因している．一方，$\{H \circ C\} \circ L \circ G$ の WFST を合成した場合には，Caseiro らの手法では最適化が行われず，状態数・状態遷移数ともに非常に大きいのに対し，提案手法では最適化の効果で状態数・状態遷移数ともに大きく削減されていることが分かる．

図 11 に提案手法，Caseiro らの手法を用いて on-the-fly 合成により音声認識を行った場合の性能を示す．縦軸が，単語正解精度，横軸が認識時間（real time factor；RTF）を表す．探索を行う際には，仮説数の上限による枝刈りと，仮説のゆう度差による枝刈り（ビームの幅）の二つのパラメータを用いているが，それぞれの手法における各点は，仮説数の上限による枝刈りのパラメータを固定し，ビームの幅を一定値で変化させることでプロットしている．図の「proposed」が提案手法を用いて on-the-fly 合成した場合，「caseiro」

表 1　WFST のパラメータ
Table 1　WFST parameters for the Caseiro's and proposed composition methods.

Network	Method	#states	#arcs
$L \circ G$	Caseiro	1,040,267	2,251,670
$L \circ G$	Proposed	1,198,351	2,409,754
$\{H \circ C\} \circ L \circ G$	Caseiro	25,357,146	34,054,721
$\{H \circ C\} \circ L \circ G$	Proposed	5,505,110	9,433,757

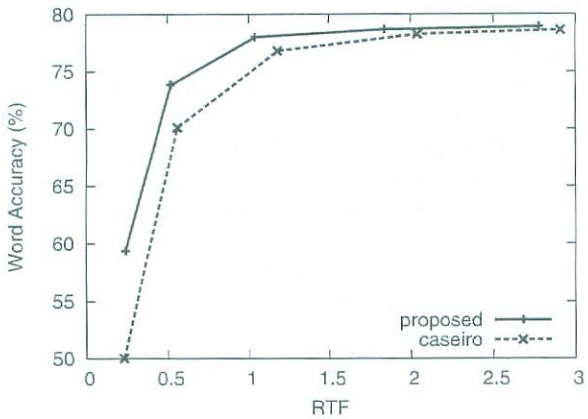

図 11 on-the-fly 合成時の認識性能の比較
Fig. 11 Recognition performance of the on-the-fly composition methods.

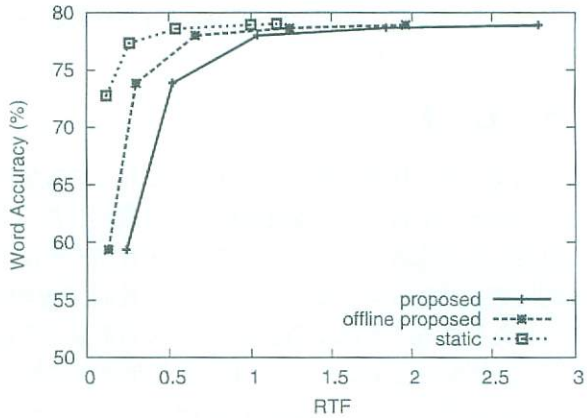

図 12 合成演算のオーバヘッドを除いた場合の認識性能の比較
Fig. 12 Recognition performance without including the CPU overhead of the on-the-fly composition.

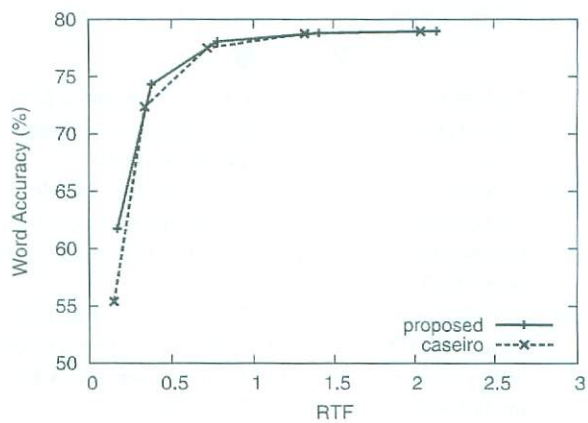

図 13 dynamic pushing の効果
Fig. 13 Performance of the dynamic pushing.

が Caseiro らの手法を用いて on-the-fly 合成した場合の結果を表す．図から，提案手法を用いることによって，Caseiro らの手法と比べ，短い時間でより高精度な認識結果を得られることが分かる．これは，出来上がる探索ネットワーク中で正解に到達しない無駄なパスがより削除され，保持する仮説中に正解が含まれやすくなり，同じ RTF に対する認識率が向上したためである．

次に，提案手法における，認識時の on-the-fly 合成処理に要するオーバヘッドの影響について調べる．そこで，このオーバヘッドの影響を単純に取り除いたときの認識性能を求めるため，提案手法の on-the-fly 合成をオフラインで事前に行って WFST を用意し，それを認識に利用したときの性能を求めた．更に比較として，すべての WFST を静的に合成し，それに最小化・決定化・pushing などの最適化を施した WFST を利用した（on-the-fly 合成を行わない）場合の認識性能も求めた．結果を図 12 に示す．図の「proposed」が提案する on-the-fly 合成をオンラインで行った場合，「offline proposed」が提案手法をオフラインで用いて事前に WFST を合成し認識に利用した場合，「static」が，静的に WFST を合成した後で最適化を行った WFST を認識に利用した場合の結果を表す．図の「proposed」と「offline proposed」の違いから，提案手法を認識時にオンラインで行った場合には，合成演算のオーバヘッドが認識時間の約 30〜45%を占めていることが分かる．また，「offline proposed」と「static」の違いを見ると，後者の方が WFST 全体を考慮した最適化が行われているため全体的に高い性能を示しており，特に RTF が 1.0 以下のときに単語正解精度の差が顕著に見られることが分かる．

最後に，図 13 に，提案手法と Caseiro らの手法における dynamic pushing の効果を評価した結果を示す．$L \circ G$ の合成のみに dynamic pushing の処理を行い，続く $(H \circ C)$ との合成では，通常の合成演算を用いて合成した．最後に，合成された WFST に対して，最終状態に到達しない無駄な状態の除去処理を行うことで，dead-end 状態が認識に与える影響を完全に除去した．図の proposed が提案手法により dynamic pushing の処理を行った場合，caseiro が Caseiro らの手法により dynamic pushing の処理を行った場合の結果を表す．図から提案手法を用いた場合，RTF が小さい領域で，若干の認識精度の向上が見られる．これは，先読みスコアのあいまい性の解消により，Caseiro らの手法に比べて，初期状態に，より重みを近づける

ことができ，早期に適切な仮説の枝刈りが行われたためであると考えられる．

8. むすび

本論文では，任意の構造の WFST に対して適用可能な on-the-fly 合成による最適化手法を提案した．大語彙連続音声認識において，発音辞書・言語モデルが動的に切り換わるような応用を想定した性能評価実験を行ったところ，従来の Caseiro らによる最適化手法よりも，効率的な探索が可能であることが確認できた．ただし提案手法では，合成演算時のオーバヘッドが大きいことが確かめられたので，今後はその削減により，更なる認識速度の向上を目指す必要がある．

謝辞 本研究は経産省「情報家電センサー・ヒューマンインターフェースデバイス活用技術開発・音声認識基盤技術」プロジェクトの支援により行った．

文　献

[1] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," Comput. Speech Lang., vol.16, no.1, pp.69–88, 2002.

[2] S. Kanthak, H. Ney, F. Fur, M. Riley, and M. Mohri, "A comparison of two LVR search optimization techniques," Proc. ICSLP, pp.1309–1312, Colorado, USA, Sept. 2002.

[3] P.R. Dixon, D.A. Caseiro, T. Oonishi, and S. Furui, "The TITECH large vocabulary WFST speech recognition system," Proc. IEEE Workshop on ASRU, pp.443–448, Kyoto, Japan, Dec. 2007.

[4] H.J.G.A. Dolfing and I.L. Hetherington, "Incremental language models for speech recognition using finite-state transducers," Proc. IEEE Workshop on ASRU, pp.194–197, Madonna di Campiglio, Italy, Dec. 2001.

[5] D.A. Caseiro and I. Trancoso, "A specialized on-the-fly algorithm for lexicon and language model composition," IEEE Trans. Audio, Speech, and Language Processing, vol.14, no.4, pp.1281–1291, July 2006.

[6] T. Hori, C. Hori, T. Minami, and A. Nakamura, "Efficient WFST-based one-pass decoding with on-the-fly hypothesis rescoring in extremely large vocabulary continuous speech recognition," IEEE Trans. Audio, Speech and Language Processing, vol.15, no.4, pp.1352–1365, 2007.

[7] O. Cheng, J. Dines, and M.M. Doss, "A generalized dynamic composition algorithm of weighted finite state transducers for large vocabulary speech recognition," Proc. ICASSP, pp.345–348, Honolulu, USA, April 2007.

[8] J. McDonough, E. Stoimenov, and D. Klakow, "An algorithm for fast composition of weighted finite-state transducers," Proc. ASRU, pp.461–466, Kyoto, Japan, Dec. 2007.

[9] T. Hori and A. Nakamura, "Generalized fast on-the-fly composition algorithm for WFST-based speech recognition," Proc. Interspeech, pp.557–560, 2005.

[10] F.C.N. Pereira and M.D. Riley, "Speech recognition by composition of weighted finite automata," in Finite-State Language Processing, pp.431–453, MIT Press, 1997.

[11] T. Oonishi, P. Dixon, K. Iwano, and S. Furui, "Generalization of specialized on-the-fly composition," Proc. ICASSP, 2009.

（平成 20 年 11 月 17 日受付，21 年 3 月 4 日再受付）

大西　翼

2006 東工大・工・情報工卒．2008 同大大学院修士課程了．現在，同大学院博士課程在籍．

ディクソン　ポール

2000 バーミンガム大・電工卒．2007 同大大学院博士課程了．現在，東京工業大学研究員．

岩野　公司　（正員）

1995 東大・工・電子情報卒．2000 同大大学院・工学系・情報工・博士課程了．同年東工大・大学院情報理工・計算工・助手．2007 同助教．2008 武蔵工大・環境情報・情報メディア・准教授．2009 東京都市大・環境情報・情報メディア・准教授，現在に至る．工博．音声認識，話者認識，音声合成，マルチメディア情報処理などの研究に従事．IEEE，ISCA，情報処理学会，日本音響学会各会員．

古井　貞熙　（正員：フェロー）

1968 東大・工・計数卒．1970 同大大学院修士課程了，NTT 研究所入社．ベル研究所客員研究員，NTT 基礎研究所第四研究室長，ヒューマンインタフェース研究所音声情報研究部長，古井特別研究室長を経て，現在，東京工業大学大学院情報理工学研究科計算工学専攻教授．工博．