2008－SLP－73 (6)
2008／10／24

WFST 音声認識デコーダの高機能化とその応用
—— on-the-fly 合成法の検討 ——

大西 翼† ディクソン ポール† 岩野 公司† 古井 貞熙†

† 東京工業大学大学院 情報理工学研究科 計算工学専攻
〒 152–8552 東京都目黒区大岡山 2–12–1–W8–77
Email: {oonishi, dixonp, iwano, furui}@furui.cs.titech.ac.jp

WFST を利用した音声認識では，メモリ消費量の削減や柔軟性の向上のために，認識時に動的に WFST を合成する on-the-fly 合成の技術が重要となる．過去に，on-the-fly 合成の高速化手法として，合成と同時に最適化処理を行う手法が提案されているが，合成元の WFST に構造制約が必要であるため，任意の構造を持つ WFST に利用することができないという問題があった．そこで，本論文では filter と呼ばれる WFST を合成時に利用することで，構造制約なく WFST の最適化処理を行う手法を提案する．本手法を実装したデコーダによる性能評価実験では，従来手法と比べて，さらに効率的な探索ネットワークが合成可能であることが確認され，本手法の有効性が示された．

Improvements and evaluations of on-the-fly WFST composition in speech recognition.

Tasuku Oonishi, Paul R. Dixon, Koji Iwano, and Sadaoki Furui

Department of Computer Science, Tokyo Institute of Technology
2-12-1-W8-77 Ookayama, Meguro-ku, Tokyo, 152-8552 Japan
Email: {oonishi, dixonp, iwano, furui}@furui.cs.titech.ac.jp

In the Weighted Finite State Transducer (WFST) framework for speech recognition, we can reduce memory usage and increase flexibility by using on-the-fly composition which generates the search network dynamically during decoding. Methods have also been proposed for optimizing composition WFSTs on-the-fly, however, these techniques place restrictions on the structure of the component WFSTs. We propose extended on-the-fly optimization techniques which can operate on WFSTs of arbitrary structure by utilizing a filter composition. The evaluations illustrate the proposed technique is able to generate more efficient WFSTs.

1 はじめに

我々は，柔軟で高性能な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 [1]．WFST 音声認識では，認識で利用する全てのモデルを 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効率化され，高性能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には，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 [2]．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．

一方，on-the-fly 合成では，状態を部分的に合成するため，WFST 中の全ての状態の情報を利用した最適化演算を行うことが難しい．この問題を解決するため，Caseiro らの on-the-fly 合成方式では，1) on-the-fly 合成の対象となる WFST 中の各状態に対し，そこから将来到達する状態の先読みを行い，2) 得られた情報を用いて，合成後された状態における将来のネットワーク構造や状態情報を推定することで，各種の最適化を実現している [3]．実際には，先読みで得られる状態情報だけでは合成後の WFST を精度よく推定することが難しいため，[3] では合成元の WFST の構造に制約を与えることで，推定の精度を向上させている．

一方，我々が開発している T^3 decoder では，Caseiro らの構造制約を取り除き，任意の構造を持つ WFST の on-the-fly 合成手法を実装している．我々の手法では合成時に「filter[4]」と呼ばれる特殊な WFST を併せて合成することで，合成後のネットワークの曖昧性を解消し，各種最適化の実現を可能にしている．これまでに本デコーダにおいて，各種最適化処理が有効に機能し，認識性能が向上することを確認している [5]．しかし，提案する on-the-fly 合成方式自体の詳細な説明と，他手法との比較による性能評価を行っていなかった．そこで，本論文では，提案する on-the-fly 方式の説明と，Caseiro の手法との認識性能の比較評価を行い，提案手法の有効性を示す．

なお，任意の構造の WFST を扱うことのできる on-

the-fly 合成方式としては，Cheng らの手法 [6] もある．この手法では，仮説探索部に最適化処理を組み込み，擬似的に WFST の最適化を行っている．このため，WFST の最適化処理と仮設探索とを切り離して実装することが難しい．一方，我々のアプローチは探索ネットワークの生成部で最適化が完結していることから，探索部を独立したコンポーネントとして扱うことが可能であり，システムの柔軟性が大きいという利点を有している．

2 WFST を利用した音声認識

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．さらに，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 4 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 のある状態 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 3 つが考えられる．これらの状態への状態遷移は，以下の規則を満たした場合に生成される．

(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，$((q_l, q_r), (q_l^{'}, q_r^{'}), i_l, o_r, w_l + w_r)$ 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．

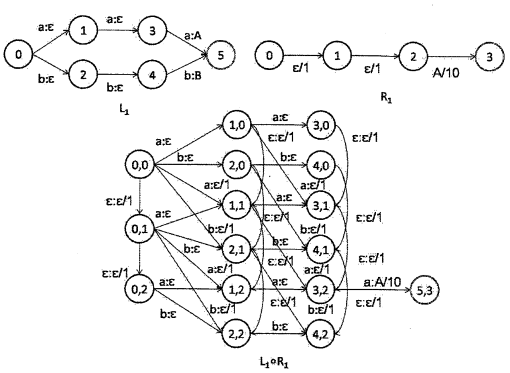

図 1: dead-end 状態が生成される例

4 on-the-fly 合成時の最適化演算

Caseiro ら [3] 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．本節では，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

4.1 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」であり，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．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 節に挙げた 3 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．

4.1.1 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，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[1]の入力シンボル集合との共通集合を調べてい

[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

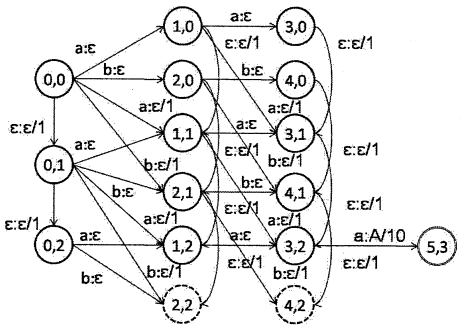

図 2: 多くの無駄な状態が合成される例

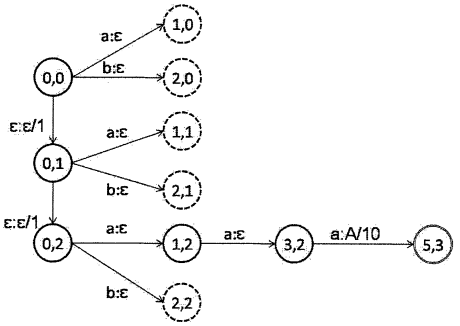

図 3: Caseiro の手法 [3] により合成された WFST

る．それが存在しない場合には，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，共通集合が無いことから，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，さら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）．

Caseiro らの手法では，1) ϵ 入力による状態遷移の生成（3 節の (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 節で述べた (1)(3) の合成規則のみを適用し，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，合成する WFST L が以下の構造制約を満たしていないと，正しい WFST を合成することができない [3].

(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

4.1.2 提案手法

我々は，任意の構造をもつ WFST に適用することができる dead-end 状態の早期回避手法として，「filter[4]」と呼ばれる特殊な WFST を併せて合成することで，ϵ 入力により生成される状態遷移を制御する方法を提案する．

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

R は動的に変更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．

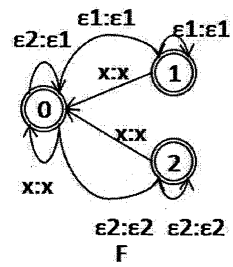

図 4: filter を表す WFST

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．) 図 4 に filter (F) を示す．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，さらに 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，$L' \circ F \circ R'$ という合成を行うことで，冗長なパスの除去が行われる．filter 中の状態 1 は R の ϵ 入力によって状態が生成されたことを，状態 2 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 と 2 の間に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 入力（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 出力（ϵ 入力）による状態遷移が生成されなくなる．これにより，冗長な状態遷移の生成を防ぐことができる．

dead-end の回避処理は，この状態遷移の生成が制限された 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．

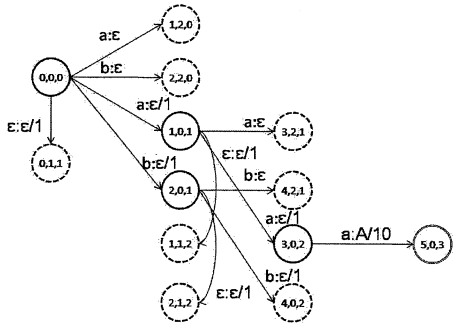

図 5: filter を用いた dead-end 処理により合成された WFST

- $q_f = 1$ のときには，以降に L の ϵ 出力 による状態遷移が発生しない．したがって，この時点で q_l からの状態遷移で非 ϵ シンボルの 出力を伴うものが存在しなければ，dead-end と判断できる．
- $q_f = 2$ のときには，以降に R の 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マッチングにより，dead-end を判定できる．
- $q_f = 0$ で，かつ，q_r に ϵ 入力 による状態遷移が存在しない場合には，次は L の ϵ 出力 による状態遷移のみが生成可能となり，それ以降では R の ϵ 入力 による状態遷移が発生しないことが保証される．したがって，$q_f = 2$ と同様の dead-end の判定処理を行う．

filter を用いて，図 1 の WFST の dead-end 状態の回避処理を行った結果は，図 5 のようになる．この手法は，合成対象となる WFST の構造に制約はないが，図 3 の結果と比較すると，全ての dead-end 状態の生成が回避できない（状態 (2,1) などの dead-end は回避されていない）ため，最適化の効率が若干劣る．

4.2 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 [3].

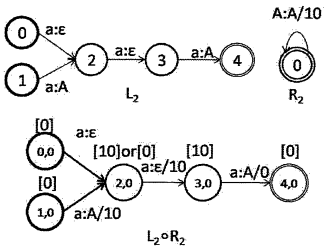

図 6: dynamic pushing の処理で先読みスコアが曖昧になる WFST の例

4.2.1 Caseiro らの手法

Caseiro らの手法 [3] 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，出来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 6 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決定されない．この様な問題は，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．[3] 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 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，先読みスコアの曖昧性を回避している．

4.2.2 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 ϵ 入力により合成された状態及びシンボルマッチにより合成された状態は，それぞれ別の状態として合成される．したがって，上述した基準を全ての状態について適用しても，先読みスコアが曖昧になる状態が発生しない．図 7 に，filter を用いた dynamic pushing により，合成される WFST を示す．先の WFST で先読みスコアが曖昧となった状態 (2,0) が，(2,2,0)，(2,0,0) のように区別されており，すべての状態で先読みスコアが一意に決定されていることが分かる．

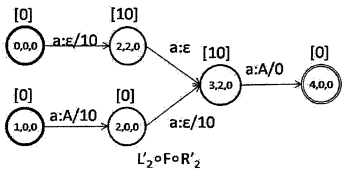

図 7: filter を用いて dynamic pushing の処理を行った WFST の例

5 実験

提案した on-the-fly 合成の最適化処理（dead-end 状態の回避処理，dynamic pushing）の効果を検証するため，日本語話し言葉コーパス (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．

5.1 想定する認識タスク

on-the-fly 合成には，個々のモデルの切り替え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 を動的に切り替えるかは，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替えることが可能な，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) を個別に切り替える」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，

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

と表すことができる．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おり，on-the-fly で L, G を合成していることを表している．

この合成を行うにあたり，1) 先に $(H \circ C) \circ L$ の on-the-fly 合成を行い，さらに G の合成を行う方式と，2) 先に $L \circ G$ の on-the-fly 合成を行い，生成された WFST を $(H \circ C)$ の右から合成する方式，の 2 通りが考えられる．提案する最適化手法では，事前に先読みシンボル集合を求める必要があるため，1) の方式だと，on-the-fly で生成された WFST $(H \circ C) \circ L$ について先読みシンボル集合を求めなければならず，処理が難しい．一方，2) では L と $(H \circ C)$ の先読みシンボル集合を事前に求めることが可能で，それぞれを一段階目，二段階目の on-the-fly 合成に利用すればよい．そこで，本実験でもこの 2) の方式で合成を行っている．

なお，比較対象となる Caseiro らの方法 [3] 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，第一段階の $L \circ G$ の合成にのみ適用した．$(H \circ C)$ 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，最適化を行わない合成演算を適用した．

5.2 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，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 10ms, 分析窓幅 25ms の MFCC12 次元+Δ MFCC12 次元+ΔΔ MFCC12 次元+Δ対数パワー+ΔΔ対数パワーの計 38 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，2.4GHz CPU (Intel Core2 Duo)，2GB メモリの計算機を使用した．

5.3 実験結果

提案する最適化手法と，Caseiro らの最適化手法 [3] 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 1,2 に示す．表 1 が，一段目の $L \circ G$ の合成を行ったときであり，表 2 が，二段目の $(H \circ C) \circ L \circ G$ の合成を行ったときである．

表 1 の $L \circ G$ を合成した場合については，提案手法の方が Caseiro の手法と比べて若干の生成された状態数・状態遷移数が多いことが分かる．これは，節 4.1.2 でも述べたとおり，提案手法では，合成対象となる WFST の構造に制約はないが，全ての dead-end 状態の生成を完全に回避できないことに起因している．なお提案手法で生成された WFST 上に残っている，最終状態に到達しない状態・状態遷移はそれぞれ全体のうちの 1.8%, 0.7% であった．一方，表 2 の WFST を合成した場合には，Caseiro 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わかる．

図 8 に，それぞれの手法で on-the-fly 合成した場合の認識時間と認識性能の関係図を示す．縦軸が単語正解精度，横軸が認識時間を表す．なお，図の static が，すべての WFST を事前に合成して最適化を行った場合，caseiro が Caseiro らの提案手法で on-the-fly 合成した場合，proposed が提案手法で on-the-fly 合成した場合を表す．この結果より，提案手法を用いた場合の方が，Caseiro の手法に比べ，わずかに認識性能が改善されていることがわかる．図 9 は，提案手法，Caseiro の手法のそれぞれを用いた場合の，on-the-fly 合成時の演算のオーバヘッドの影響を取り除いた場合の認識性能の比較を示す．具体的には，それぞれの手法を適用して事前に合成・最適化した WFST を利用して認識性能の比較を行っており，これにより生成される探索ネットワークの効率を純粋に比較することができる．この結果を見ると，提案手法の方がより効率的な探索ネットワークが合成されていることがわかる．なお，図 8 と比較して，提案手法と Caseiro の手法の性能差が大きくなっていることから，提案手法の方が合成演算のオーバヘッドが大きいことがわかる．

表 1: WFST $L \circ G$ 合成時の状態数・状態遷移数の違い

Method	#states	#arcs
Caseiro[3]	1,040,267	2,251,670
Proposed	1,217,744	2,865,283

表 2: WFST $(H \circ C) \circ L \circ G$ 合成時の状態数・状態遷移数の違い

Method	#states	#arcs
Caseiro[3]	25,357,146	34,054,721
Proposed	5,342,672	1,153,358

6 まとめ

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替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，効率的な探索ネットワークの生成が可能であり，高い認識性能を有す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，さらなる認識性能の向上を目指す必要がある．また，より高度なモデルを利用した頑健な音声認識の可能性を検討していく予定である．

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．

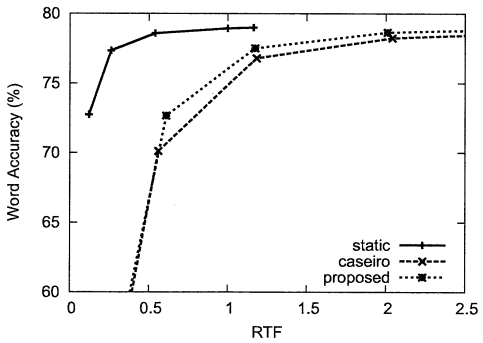

図 8: on-the-fly 合成時の認識性能の比較

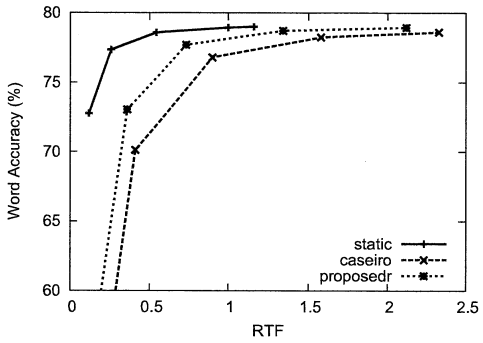

図 9: 合成演算のオーバヘッドを除いた場合の認識性能の比較

参考文献

[1] Paul R. Dixon, Diamantino A. Caseiro, Tasuku Oonishi, and Sadaoki Furui. The TITECH large vocabulary WFST speech recognition system. *Proc. IEEE Workshop on ASRU*, pp. 443–448, 2007.

[2] Hans J.G.A. Dolfing and I.Lee Hetherington. Incremental language models for speech recognition using finite-state transducers. *Proc. IEEE Workshop on ASRU*, pp. 194–197, 2001.

[3] Diamantino A. Caseiro and Isabel Trancoso. A specialized on-the-fly algorithm for lexicon and language model composition. *IEEE Transactions on Audio, Speech, and Language Processing*, Vol. 14, No. 4, pp. 1281–1291, 2006.

[4] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, Vol. 16, No. 1, pp. 69–88, 2002.

[5] Tasuku Oonishi, Paul R. Dixon, Koji Iwano, and Sadaoki Furui. Implementation and evaluation of fast on-the-fly composition algorithms. *Proc. Interspeech*, pp. 2110–2113, 2008.

[6] Octavian Cheng, John Dines, and Mathew Magimai Doss. A generalized dynamic composition algorithm of weighted finite state transducers for large vocabulary speech recognition. *Proc. ICASSP*, pp. 345–348, 2007.