Dynamic Grammars with Lookahead Composition for WFST-based Speech Recognition

Josef R. Novak[1], *Nobuaki Minematsu*[1], *Keikichi Hirose*[1]

[1]Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
{novakj,mine,hirose}@gavo.t.u-tokyo.ac.jp

Abstract

Automatic Speech Recognition (ASR) applications often employ a mixture of static and dynamic grammar components, and can thus benefit from the ability to efficiently modify the system vocabulary and other parameters in an on-line mode. This paper presents a novel, generic approach to dynamic grammar handling in the context of the Weighted Finite-State Transducer (WFST) paradigm. The method relies on a straightforward extension of the lexicon and underlying grammar components, and leverages the ideas of on-the-fly composition and delayed construction to efficiently generate the recognition search space on-the-fly. The alternative partitioning of component models that this approach implies can also result in significant storage savings. In contrast to previous works in this area, the proposed method relies only on generic WFST operations and the context-dependency, lexicon and grammar components that form the basis of standard ASR cascades.

Index Terms: WFST, ASR, Dynamic vocabulary, Spoken dialog systems

1. Introduction

Automatic Speech Recognition (ASR) applications often utilize a mixture of static and dynamic grammar components and can thus benefit form the ability to efficiently switch or manipulate these components. In the context of systems based on the Weighted Finite-State Transducer (WFST) paradigm however, correctly splicing together the component grammars at runtime can be complicated. In particular some solution must be found to correctly and efficiently enforce cross-word context-dependency constraints for any dynamic components despite the fact that this information may not be known ahead of time. In this paper we present a new solution to this problem which leverages the idea of lookahead composition [1] and straightforward extensions to the lexicon and grammar components. This contrasts with the approach previously proposed in [2], which relies on specialized "enforcer" FSTs to correctly generate cross-word context dependency information when splicing dynamic component grammars.

The remainder of the paper is structured as follows. Section 2 provides background. Section 3 describes the proposed solution in detail. Section 5 concludes the paper and discusses future directions for the current work.

2. Speech Recognition with Weighted Finite-State Transducers

The WFST paradigm has gained considerable popularity in the ASR and spoken language processing communities in recent years [3, 4, 5, 6]. This approach typically relies on an integrated recognition cascade, which is constructed by first transforming the component knowledge sources - acoustic model, pronunciation lexicon and grammar - into equivalent finite-state representations, and then iteratively composing and optimizing them. In the current work we focus on a three-component cascade comprising a context-dependency WFST C that maps context-dependent triphone sequences to monophones, a pronunciation lexicon L that maps monophone sequences to words, and a grammar G, usually represented as an acceptor, that models word sequences.

The WFST paradigm provides a wide range of optimization techniques which can be applied to arbitrary transducers or acceptors in order to eliminate ambiguity, minimize the total number of states or transitions, push weights, synchronize labels, and remove epsilon transitions. There are many possible integration combinations given the set of component models and variety of optimization algorithms, each of which implies a different trade-off between competing factors of Word Accuracy (WACC), Real-Time Factor (RTF), build-time, storage requirements and RAM. In the present work we start with the following baseline construction:

$$\pi(C \circ det(L \circ G)), \tag{1}$$

where π refers to auxiliary symbol removal, det refers to $determinization$ and $\circ$ refers to standard composition while the C, L, and G components are as described above.

2.1. On-the-fly Composition

The static construction described by Eq. 1, and in particular the $det(L \circ G)$ operation, can be quite costly, especially in the case of large language models. In response to this concern, several approaches to on-the-fly composition have been proposed in recent years [1, 4, 5]. These alternatives avoid the often costly construction of the full, static cascade by performing composition at runtime as part of the decoding process. This can also result in a significant memory savings during decoding because only a small part of the full search space is ever expanded for any individual utterance [1]. In [1] the authors proposed a series of generalized filters that greatly speed up the composition process by preventing the creation of dead-end states and preprocessing weights, labels and epsilon transitions. This approach has been implemented in OpenFst [7] and is the one we adopt for this work. When using lookahead composition for speech recognition applications, the component WFSTs - C, L and G - remain the same, but an alternative partitioning is utilized:

$$\pi(C \circ det(L)).G, \tag{2}$$

where we use "." to refer to lookahead composition. In the above case the $(C \circ det(L))$ and G components are precompiled, but the lookahead composition operation is performed at runtime, as part of the decoding process. Because the full cascade need not be statically expanded, a significant savings in both storage space, build-time and on-line memory requirements can be achieved, with only minor impact to decoding speed [1, 5]. The following section describes how on-the-fly composition can be leveraged to produce a novel, efficient and straightforward alternative approach to dynamic grammar management for WFST-based systems.

3. Dynamic grammars with Lookahead composition

Previous work in this area has focused on an approach where the top-level grammar as well as any component grammars are precompiled as much as possible into fully expanded $C \circ L \circ G$ cascades [2]. Cross-word context-dependency issues were then handled through the use of specialized *enforcement* transducers that correctly enforced constraints, either through the use of on-line composition, or through modifications to the decoder core [2]. The series of operations is described in [2] as $E \circ splice(C'LG, D)$ where E represents the *enforcement* FST, C' represents a context-dependency transducer that has been factored into left and right components, and D represents the set of dynamic grammars.

The solution we propose instead focuses on the $C \circ det(L)$ and G lookahead partitioning described in Subsection 2.1. Correct context-dependency constraints for component grammars and dynamic vocabulary are enforced through straightforward extensions to the lexicon, and a reinterpretation of the grammar components as transducers as opposed to acceptors.

3.1. Modifying $C \circ det(L)$

The $C \circ det(L)$ component maps context-dependent triphone sequences to words. The closure operation is applied to L prior to composition with C which ensures that sequences of words can be handled. Under the proposed approach the pronunciation dictionary is augmented with entries for each of the individual monophones defined in the acoustic model. The pronunciation dictionary is then transformed in the standard way. When the augmented L is composed with C, the resulting $C \circ det(L)$ WFST encodes not just word sequences, but also transitions between words and monophones, and all possible context-dependent triphone information. Effectively it exposes the output side of the C transducer through the lexicon. The augmented $C \circ det(L)$ WFST is larger than the original, but the difference is not great, even in the case where positionalized monophones are utilized. Table 1 illustrates the differences between the $Proposed$ and $Standard$ constructions for both a 106k lexicon dervied from the Corpus of Spontaneous Japanese, and a smaller 8k lexicon from the Kyoto Tour dialog corpus. In both cases the underlying acoustic models contained 168 positionalized monophones. The *Proposed* construction results

Table 1: Cascade characteristics for optimized $C \circ det(L)$ components for 8k and 110k lexicons. The *Proposed* construction is augmented with positional monophones.

$C \circ det(L)$	***Arcs***	***States***	***Size / lkhd size***
Stand. (8k)	65,956	34,152	1.2M / 2.4M
Prop. (8k)	386,493	43,236	6.4M / 7.2M
Stand. (106k)	222,019	72,346	4.2M / 6.2M
Prop. (106k)	930,539	438,660	19M / 31M

in a significantly larger $C \circ det(L)$ component. In the *Standard* construction only those positional monophones that actually occur in the lexicon need be considered, however in the *Proposed* construction these unseen contexts must also be expanded as depicted in Figure 1. The relative cost of computing the lookahead information is also increased, due both the larger phoneme inventory and the increased complexity of the cross-word context-dependency information. If the underlying acoustic model does not employ positional monophones the relative increase in size and complexity is greatly reduced. Although the $Proposed\ C \circ det(L)$ construction is notably larger, we note that one copy of this component can potentially be shared among many top-level grammars without modification.

3.2. Modifying G

In a standard WFST cascade the grammar or G element is usually represented as a Weighted Finite-State Acceptor (WFSA) instead of a WFST. This reflects the fact that statistical N-gram models as well as expert grammars typically model word sequences and do not transduce anything. In the proposed method however, we opt to explicitly treat G as a synchronized WFST with matching input and output labels. Each of the sub-grammar components is then represented as a WFST mapping monophone sequences to words. In the case where a sub-grammar component is simply a word or list of words, it is sufficient to generate a new lexicon transducer, L_n. If a word from a sub-grammar is already covered in the base lexicon, then a word-level representation will suffice. If the sub-grammar is itself an N-gram model or regular-expression based expert grammar, then the component takes the form $L_n \circ G_m$. Thus the same tools used to create the base L and G can be reused to generate sub-grammar components $L_n \circ G_m$ and no specialized FSTs are required. Each sub-grammar can be pre-compiled and optimized without consideration of internal or external context with one exception. In the case where a sub-grammar component itself contains non-terminal symbols, the non-terminal symbols should be placed only on the output side of the component and synchronized with the epsilon symbol on the input side. During replacement nonterminal symbols with input epsilon arcs are treated as acceptors. The replacement operation replaces both the input *and* the output. This guarantees both that subsequent replacement operations will be successful, and that the output symbol table for the $C \circ det(L)$ component can be shared by any top-level grammar or sub-grammar. Figure 2 depicts a top-level grammar with an unresolved non-terminal corresponding to a simple telephone grammar. Figure 3 depicts a simple example of an $L \circ G$ sub-grammar representing a possible resolution of the non-terminal ϵ:<name> symbol in Figure 2.

3.3. Symbol management

One final concern is proper management of the unified output symbol table. One solution is to use a global namespace but, as described in [2], this is may not work if sub-grammars are shared, or if the lexicon is generated dynamically. The authors from [2] employ an alternative solution using a "shim" layer which maps one namespace to another. In our approach, the top-level symbol table is compiled at run-time in a recursive manner, reflecting the structure of the underlying components. As each sub-grammar is loaded, its output symbol table is compared against the component it is being bound into. Missing symbols are added to the parent table and the sub-grammar is relabeled. The process terminates once all non-terminals in the top-level grammar have been resolved.

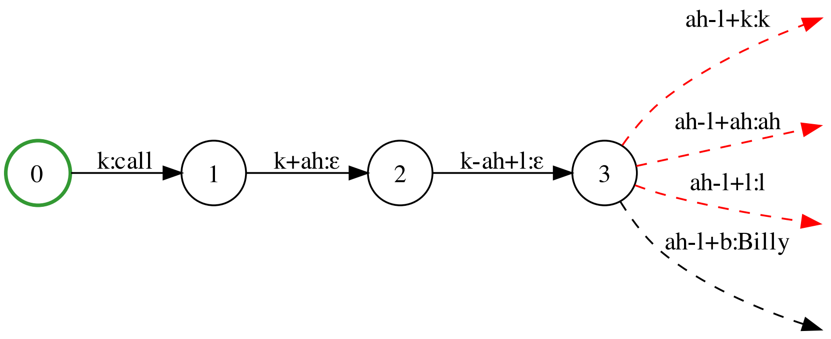

Figure 1: A partial representation of the *Proposed* alternative $C \circ det(L)$ construction corresponding to a simple telephone application. Monophone entries in the lexicon ensure that all possible crossword contexts are supported, and make it possible to add new vocabulary items as needed. Default vocabulary entries function as normal. In practice the *closure* operation is applied to the CL component, ensuring support for arbitrary word sequences and recursive subgrammars.

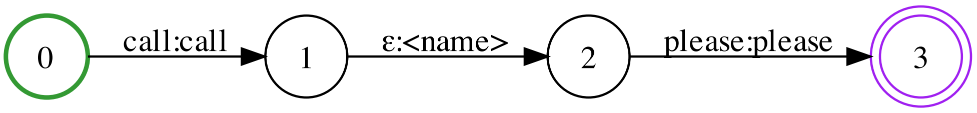

Figure 2: Explicit WFST representation of top-level grammar for a simplistic telephone application.

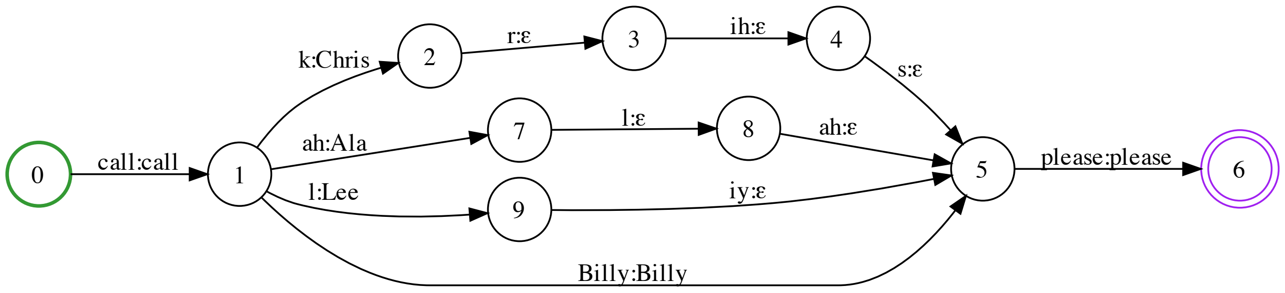

Figure 3: Explicit WFST representation of top-level G component, following replacement with a sub-grammar for the regular expression $/Chris|Ala|Lee|Billy/$. After replacement of the "ϵ:<name>" non-terminal, the input side is a mixture of word and monophone labels. The name "Billy" is covered in the baseline dictionary, thus in this case there is no need to provide an explicit monophone pronunciation.

3.4. Putting it together

The underlying decoding process is unchanged to that used for either static decoding or lookahead composition. We have implemented the proposed approach in an OpenFst [7] based WFST decoder based on Juicer [8] which we are currently developing. All of the component WFST operations which the proposed approach relies upon: replacement, arc sorting, and composition admit a lazy implementation where the result is computed on-demand. These delayed operations are likewise supported by OpenFst. Thus, once the modified $C \circ det(L)$ component, the top-level grammar, G and any sub-grammars are prepared and compiled it is sufficient to construct a cascaded, delayed transducer,

$$(C \circ det(L)) \,.\, arcs_i(repl(G, g_1, ..., g_n)) \qquad (3)$$

where $arcs_i$ refers to input arc sorting - a requirement for lookahead composition, and *repl* refers to replacement, which may be recursive depending on the structure of the grammar. This WFST can then be expanded on-the-fly during recognition, thereby also eliminating any need to statically perform the replacement operation and providing a considerable memory savings, particularly in the case of larger top-level grammars with numerous dynamic word classes.

4. Experiments

We first evaluated the proposed approach using a modified version of the WSJ [9] si_dt_s2-$20k$ task including 204 test sentences. In addition to an open-vocabulary 2-gram LM, and the standard 20k lexicon, we generated a single dynamic word class for a list of 159 OOV and other words. The dynamic word class was then bound into the top-level N-gram model and used to replace the unknown word token. We evaluated the WACC vs. RTF performance of the proposed method in two different ways. In the first instance all replacement operations were performed statically prior to decoding, while in the second the replacement operations were also carried out on-the-fly. Both scenarios used

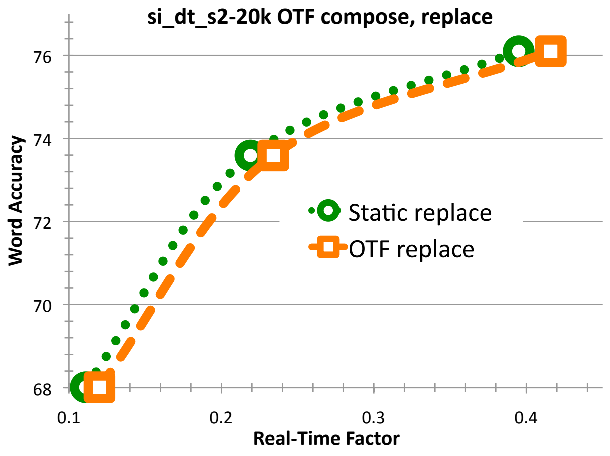

Figure 4: WACC vs. RTF comparison of static compilation of G versus dynamic replacement.

lookahead composition to compose the $C \circ det(L)$ and G components on-the-fly. The results of the experiments are depicted in Figure 4, and are in-line with previous results for this particular test set [10]. The results show that the proposed approach is reasonably efficient applied to a mid-size top-level grammar. Dynamic replacement imposes an additional RTF penalty on top of dynamic composition, but this also appears to be acceptable.

The preceding experiments illustrate the reasonable efficiency of the approach in a simple configuration, however spoken dialog systems often consist of a smaller top-level grammar and a wider variety of sub-grammars. We have recently begun to apply the proposed approach to such a system focusing on the tour guide domain. In this system a smaller top-level bigram LM with a 8242 word vocabulary is augmented with three dynamic classes including landmarks, people and organization names - totaling to 8681 unique words. The storage savings that the proposed approach leads to are decent, especially for larger grammars. Table 2 depicts the cascade storage characteristics for a larger 3-gram version of the tour guide application. The 3-gram version resulted in a larger RTF penalty, probably due to the greatly increased number of dynamic elements, as well the naive pruning approach currently employed by our decoder, but still ran at faster than real-time.

Table 2: Storage requirements in MB for a trigram version of the tour guide application using 3 dynamic word classes.

Cascade type	***Size***
Static $(C \circ det(L)).G$	174MB
$C \circ det(L)$, static G	63.6MB
$C \circ det(L)$, dynamic $G, g_1, ..., g_n$	20.7MB

5. Conclusions and Future work

In this paper we have introduced a novel WFST-based approach to dynamic grammar management that leverages on-the-fly lookahead composition and a re-casting of the $C \circ det(L)$ and G component FSTs. We have shown that the approach is both flexible and fairly efficient, and affords a straightforward implementation.

Further experiments are still needed, and In future we plan to further investigate in more detail the impact that larger dynamic vocabulary components, as well as the number of components have on decoding RTF, particularly in the case of larger N-gram style top level grammars. It may be possible to further improve performance in this case by utilizing subword units other than monophones. We are also interested in the potential of applying the proposed approach to Out of Vocabulary (OOV) modeling.

6. References

[1] C. Allauzen, M. Riley, and J. Schalkwyk, "Filters for efficient composition of weighted finite-state transducers," in *CIAA*, 2010.

[2] J. Schalkwyk, L. Hetherington, and E. Story, "Speech recognition with dynamic grammars using finite-state transducers," in *Eurospeech*, 2003, pp. 1969–1972.

[3] M. Mohri, "Finite-state transducers in language and speech processing," in *Computational Linguistics*, 1997, vol. 23, issue 2.

[4] D. Caseiro and I. Trancoso, "Using dynamic wfst composition for recognizing broadcast news," in *ICSLP*, 2002, pp. 1301–1304.

[5] T. Hori, C. Hori, and Y. Minami, "Fast on-the-fly composition for weighted finite-state transducers in 1.8 million-word vocabulary continuous speech recognition," in *Interspeech*, 2004, pp. 289–292.

[6] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite-state transducers," in *Springer Handbook of Speech Processing*, 2008, pp. 1–31.

[7] C. Allauzen and M. Riley, "Openfst: A general and efficient weighted finite-state transducer library," in *tutorial, SLT*, 2010.

[8] D. Moore, V. Valtchev, Magimai M., O. Vepa, O. Cheng, and T. Hain, "Juicer: A weighted finite state transducer speech decoder," in *InterSpeech*, 2005, pp. 241–244.

[9] D. Paul and J. Baker, "The design for the wall street journal-based csr corpus," in *ICSLP*, 1992, pp. 357–362.

[10] J. Novak, P. Dixon, and F. Furui, "An empirical comparison of the t3, juicer, hdecode and sphinx3 decoders," in *InterSpeech 2010*, 2010, pp. 1890–1893.