DynamicGrammarswithLookaheadCompositionforWFST-basedSpeechRecognition

JosefR.Novak1, NobuakiMinematsu1, KeikichiHirose1

1GraduateSchoolofInformationScienceandTechnology,TheUniversityofTokyo,Tokyo,Japan
{novakj,mine,hirose}@gavo.t.u-tokyo.ac.jp

Abstract
AutomaticSpeechRecognition(ASR)applicationsoftenemployamixtureofstaticanddynamicgrammarcomponents,and
canthusbenefitfromtheabilitytoefficientlyswitchormanipulate
thesecomponents.Inthecaseofsystemsthatemploythe
WeightedFinite-StateTransducer(WFST)paradigm,thefactthat
thisinformationmaynotbeknownaheadoftime.

In this paper we present a new solution to this problem which
thatthisapproachimpliescanalsoreesultinsignificantstorage
Incontrasttopreviousworksinthisarea,theproposedmethodreliesonlyongenericWFSToperationsand
dynamicallyconstructtoefficientlygenerate therecognitionsearchspace
on-the-fly. The alternative partitioning of model components
thatthisapproachimpliescanalsoreesultinsignificantstorage
In contrast to previous works in this area, the proposed method relies only on generic WFST operations and
dynamically constructs to efficiently generate the recognition search space
on-the-fly. The alternative partitioning of model components
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
dynamically constructs to efficiently generate the recognition search space
on-the-fly. The alternative partitioning of model components
that this approach implies can also result in significant storage
savings.

2.1. On-the-flyComposition
ThestaticconstructiondescribedbyEq.1,andinparticular the
det(L⊙G)operation,canbequitespecialy,especiallyinthe
caseoflargelanguagemodels.Inresponsetothisconcern,several
approaches to on-the-fly composition have been proposed inrecent
years [1, 4, 5]. Thesearchavoidstheoften costly
cost of constructing 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 fillers 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:

\[ \pi(C \circ \text{det}(L \circ G)), \]

where \( \pi \) refers to auxiliary symbol removal, \( \text{det} \) refers to
determinization and \( \circ \) refers to standard composition while the
C, L, and G components are as described above.

2.1. On-the-flyComposition
The static construction described by Eq. 1, and in particular the
det(L⊙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 fillers 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:

\[ \pi(C \circ \text{det}(L)) \circ G, \]

where we use “.” to refer to lookahead composition. In the
above case the \((C \circ \text{det}(L)) \circ G \) 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 $E_{ospice}(C', LG, D)$ 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 derived 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 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, $L_m$. 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 $L_m \circ G_m$. Thus the same tools used to create the base $L$ and $G$ can be reused to generate sub-grammar components $L_m \circ G_m$ 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 $<\text{name}>$ 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 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 $C \circ \text{det}(L)$ 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 $C.L$ 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 “$\epsilon:<\text{name}>$” 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 $C \circ \text{det}(L)$ component, the top-level grammar, $G$ and any sub-grammars are prepared and compiled it is sufficient to construct a cascaded, delayed transducer,

$$(C \circ \text{det}(L)).\text{arcs}(\text{repl}(G,g_1,\ldots,g_n))$$

where $\text{arcs}$ refers to input arc sorting - a requirement for lookahead composition, and $\text{repl}$ 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] s$\text{t}_t$-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...
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.

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