Painless WFST cascade construction for LVCSR - Transducersaurus

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

[1]Graduate School of Information Science and Technology, The University of Tokyo, Japan

{novakj,mine,hirose}@gavo.t.u-tokyo.ac.jp

Abstract

This paper introduces the Transducersaurus toolkit which provides a set of classes for generating each of the fundamental components of a typical WFST ASR cascade, including a Context-dependency transducer, a Lexicon, a stochastic language model and an optional silence class model. The toolkit further implements a simple scripting language in order to facilitate the construction of cascades with a variety of popular combination and optimization methods and provides integrated support for the T^3 and Juicer WFST decoders, and both Sphinx and HTK format acoustic models. New results for two standard WSJ tasks are also provided, comparing a variety of cascade construction and optimization algorithms. These results illustrate the flexibility of the toolkit as well as the tradeoffs inherent in various build algorithms.

Index Terms: Speech Recognition, WFST, LVCSR

1. Introduction

In recent years the Weighted Finite-State Transducer (WFST) paradigm [1] has gained considerable popularity as a platform for Automatic Speech Recognition (ASR). The WFST approach provides an elegant, unified mathematical framework that can be utilized to train, generate, combine and optimize the many heterogenous knowledge sources that typically make up a modern Large Vocabulary Continuous Speech Recognition (LVCSR) system. This has lead to the development of several excellent general software libraries devoted to the construction and manipulation of WFSTs including the popular open source OpenFst [2] toolkit. Much research has also been conducted on the theoretical construction, integration and optimization of WFST models for ASR [3]. Nevertheless to our knowledge at present there is no open source toolkit devoted to the construction of ASR-specific WFST models.

This lack of available tools represents a serious obstacle to the wider dissemination of WFST-based methods. The current work introduces the Transducersaurus WFST toolkit [4], which aims to provide a unified, flexible and transparent approach to the construction of integrated WFST-based ASR cascades, while incorporating recent research results on this important topic. It includes a set of classes for constructing component models as well as a simple Domain Specific Language (DSL) suitable for specifying cascade integration and optimization commands. It provides integrated support for HTK and Sphinx acoustic models and cascade construction support for both the T^3 [5] and Juicer [6] WFST decoders. Where in past complicated development was required, with this toolkit input knowledge sources and a single command are sufficient to build a high-performance system.In addition to introducing the toolkit, this work contributes new experimental results for two LVCSR tasks from the Wall Street Journal [7] (WSJ) corpus, and provides discussion of alternative cascade build chains.

The remainder of the paper is structured as follows. Section 2 describes the main component models of a typical WFST-based ASR cascade. Section 3 describes the cascade integration tool and its capabilities. Section 4 describes new experimental results that explore the flexibility of the Transducersaurus toolkit. Section 5 provides additional analysis and explores the practical implications of various construction techniques. Finally, Section 6 concludes the paper.

2. Cascade components

A typical WFST-based ASR cascade constitutes the integrated product of multiple component models. The most straightforward and perhaps common construction involves three component models. The Grammar - G, typically takes the form of a statistical language model. The Lexicon - L, encodes a pronunciation dictionary which maps monophone sequences to words. The Context-dependency transducer - C, maps triphone sequences to monophones, thus encoding contextual information for longer sequences of monophones or words. Each of these core component transducers is supported by the Transducersaurus toolkit. The toolkit also includes an optional class suitable for constructing a silence class model - T as described in [3]. In addition to the C, L, G, and T models, an H model which performs a mapping from HMM distributions to triphone sequences is often described in the literature. Support for the H model has been omitted at this point, but may be included in future releases. The remainder of this section briefly describes each of these components.

2.1. Grammar

The grammar, G typically represents a statistical language model and encodes it as a Weighted Finite-State Acceptor (WFSA). The literature, particuarly [9] discusses several different approaches to the construction or transformation of a standard N-gram model to WFSA format. The simplest approach utilizes standard ϵ-transitions to represent back-off arcs and a history-less back-off state; a small example is depicted in Figure 1. This approach has the minor drawback that in some cases back-off arcs are less costly than normal N-gram arcs, resulting in sub-optimal path choices. This can be avoided by utilizing failure or ϕ-transitions for the back-off arcs, which explicitly encode the idea that a back-off arc should only be utilized where a normal N-gram alternative does not exist. Yet another alternative involves generating additional back-off states, mutating the original input model so as to eliminate competition between normal N-grams and back-off arcs. The Transducersaurus toolkit currently implements the standard ϵ-transition back-off strategy by default. This choice reflects the relative simplicity of the algorithm, conciseness of the result and the fairly limited impact that this choice has on ASR accuracy. Support for alternative constructions is under way.

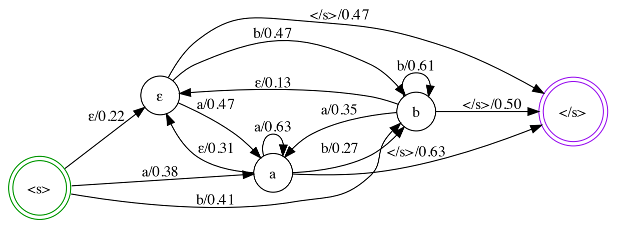

Figure 1: Detail of a bi-gram model for a simple two word LM.

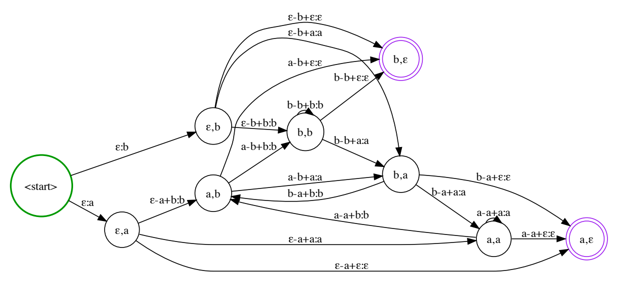

Figure 2: Detail of a 2-phone Context-dependency WFST C. Explicit auxiliary self-loops have been omitted.

2.2. Lexicon

The lexicon transducer, L provides a mapping from monophone sequences to words. The construction of the L transducer is straightforward however, as described in [3] there is one caveat. The introduction of homophones, entries which share the same pronunciation but differ in terms of their surface form or orthography, will have the undesirable effect of making the lexicon or resulting cascade non-determinizable. This can be avoided by augmenting the phoneme set with auxiliary symbols that are mapped down through the cascade. The lexicon class handles this automatically.

2.3. Context-dependency

The Context-dependency transducer, C provides a mapping from triphone sequences to monophone sequences. The construction algorithm is essentially a simple thrice nested for-loop which constructs all possible logical triphone sequences from a list of input monophones. Again [3] introduces two fundamentally different approaches to constructing the C transducer. The first results in a non-deterministic, non-delayed C, while the second results in a deterministic but delayed C. The toolkit focuses on the deterministic construction as this is preferable for the construction and optimization of integrated ASR cascades. Figure 2 shows a detail of a simple 2-phone context-dependency transducer using the deterministic construction algorithm. The project also supports the inclusion of auxiliary symbols in the form of simple self-loops, in order to handle the homophony issue described above.

HTK and Sphinx format acoustic models differ in terms of formatting, and Sphinx models default to the use of positional triphones. These differences necessitate slightly different approaches to the construction strategies for the C transducer. Thus the project includes separate classes for constructing the HTK versus Sphinx based C models, which provide integrated support for the HTK hmmdefs and tiedlist files and the Sphinx mdef files respectively.

2.4. Silence class

The silence class transducer, T transforms a grammar by grafting optional silence transitions onto existing word sequences. Other approaches to silence modeling for WFSTs include augmenting the lexicon with silence tokens, or utilizing a force-aligned language model with integrated silence or short pause information. These alternatives are implicitly supported by the defaut L and G models. estimated from several hundred hours of force-aligned data from the Fisher corpus [8].

3. Cascade integration

Before the component WFST models can be used for recognition inside of a WFST decoder, it is first necessary to integrate them into some form of cascade. The Transducersaurus tookit provides such an integration tool in the form of transducersaurus.py, which calls the model construction classes described in Section 2, and performs integration, optimization and conversion of the ASR cascade automatically. The tool utilizes the OpenFst toolkit for all compilation, composition and optimization procedures. The tool provides a wide range of features which allows the user to specify information such as the desired semiring [11], the set and order of combination and optimization operations, the acoustic model type and the preferred decoder just to name a few. There are many different approaches to combining and optimizing WFST cascades, as can be appreciated from the literature on the subject, and the diversity of the construction procedures described in the previous sections. The approaches described in the literature typically advocate,

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

where π refers to auxiliary symbol removal, min refers to weighted minimization, det refers to weighted determinization, and $\circ$ refers to standard composition. This may be further augmented by weight-pushing. Alternatively, the simpler and less memory intensive,

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

has also been shown [10] to produce good results. Furthermore the log semiring is typically recommended as it helps preserve any stochastic properties of the input. Nevertheless, it can be time consuming to produce alternative build constructions, yet the process of experimentation and exploration is essential to developing a strong understanding regarding the effects of these procedures on Word Accuracy (WACC) and Real-Time Factor (RTF). Thus the flagship contribution of this toolkit is a simple WFST-oriented DSL which aims to streamline the specification of build algorithms.

3.1. Cascade construction DSL

The DSL supported by the build tool allows the user to specify a build chain using a subset of the standard FST-based combination and optimization algorithms, as well as shorthand for the component models described earlier. The user specifies a simple chain for example,

```
--command "min(det(C*det(L*(G*T))))",
--command "(C*det(L)).(G*T)"
```

and the build tool will automatically tokenize and parse the command into the appropriate series of OpenFst commands, generating intermediate results as necessary along the way. At present the DSL is quite limited, but supports the min, det, $\circ$ (specified "$*$" on the command line) and "." operations as well as the construction of the C, L, G, and T component transducers. The "." operation refers to static Look-Ahead (SLA) composition, which was released in a recent version of OpenFst, and which implements the Look-Ahead composition algorithm proposed in [12]. Auxiliary symbol replacement is handled automatically in a manner dependent on the set of build commands issued by the user.

The advantage of the DSL approach is that it permits very simple specification of the build chain, which in turn encourages experimentation and hopefully learning, and lends itself easily to further extension through the future addition of other standard operations. Thus the user only needs to prepare the component knowledge sources, and specify a build algorithm. For example the command,

```
./transducersaurus.py --tiedlist tiedlist
--hmmdefs hmmdefs --grammar my.lm
--lexicon my.lex --amtype htk
--command "(C*det(L)).(G*T)" --convert tj
```

would automatically construct a cascade utilizing a silence class model, SLA composition and an HTK acoustic model and output cascades suitable for use in both Juicer and T^3.

4. Experiments

In order to showcase the versatility of the proposed toolkit, we conducted a variety of experiments using standard WSJ test and training sets, and a wide selection of different build chains. Below we report several results for both the Juicer and T^3 WFST decoders and HTK and Sphinx based acoustic models.

4.1. Experimental setup

All experiments for this work were performed on an 8 core Intel Xeon based machine running at 3GHz with a 6MB cache and 64GBs of main system memory and using the RHEL operating system. As with our previous results from [10], the experiments covered two popular tasks from the WSJ corpus. The first task, *nov92-5k*, focuses on the November 1992 ARPA WSJ test set which comprises 330 sentences, and was evaluated using the WSJ 5k non-verbalized vocabulary and the standard WSJ 5k closed bigram language model. The second task, *si_dt_s2-20k*, focuses on a subset of the WSJ1 Hub2 test set which comprises 207 sentences. The *si_dt_s2-20k* task, which is somewhat more difficult, was evaluated with the standard WSJ 20k non-verbalized closed bigram language model and corresponding vocabulary. In order to help ensure the repeatability of our experiments, open source AMs described in [13] were used throughout, and auxiliary parameter values for the T^3 and Juicer decoders were specified as in [10]. Unless otherwise specified the log semiring was used for all constructions.

4.2. Basic build chains

The first set of experiments looked at the impact of different optimization chains on the small bigram nov95-5k task. Three different cascade generation commands were fielded to the build tool, $C \circ det(L \circ (G \circ T))$, $det(C \circ det(L \circ (G \circ T)))$, and $min(det(C \circ det(L \circ (G \circ T))))$ to construct integrated cascades for the *nov92-5k* task. The RTF vs. WACC results of the experiments using HTK models and the T^3 WFST decoder are shown in Figure 3.

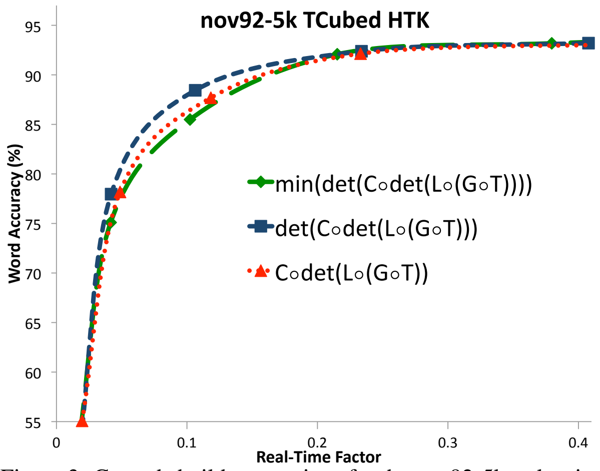

Figure 3: Cascade build comparison for the nov92-5k task using the T^3 decoder, HTK AM and various optimization techniques.

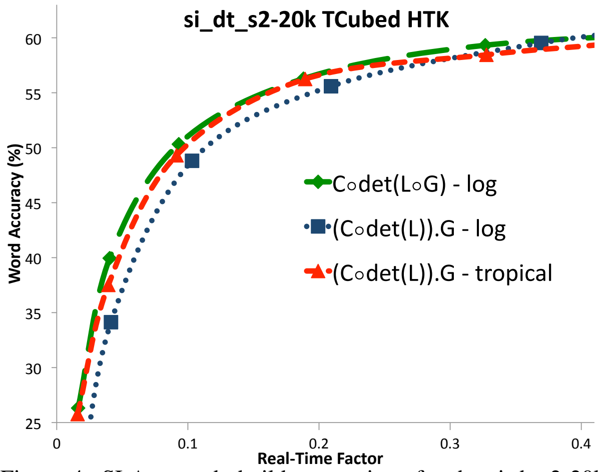

Figure 4: SLA cascade build comparison for the si_dt_s2-20k task using the T^3 decoder and HTK AM.

4.3. Standard composition versus SLA composition

The build tool allows the user to select either standard composition or SLA composition for each of the composition operations by simply specifying the "$*$" or "." operator respectively. As such this made it simple to evaluate the effectiveness of the SLA approach versus the standard composition approach. RTF vs. WACC results for these approaches for the *si_dt_s2-20k* task are depicted in Figure 4. The SLA cascade construction from [12] purports two major advantages over the standard approach, even in the case where composition is not performed during decoding. The first is that the composition operation is both faster and more efficient in terms of memory consumption. The second is that, in the case where the full cascade is not optimized, omitting the $det(L \circ G)$ operation, affords a substantial reduction in maximum memory requirements. Furthermore, if the decoder supports fast On-the-Fly composition, the precomputed $(C \circ det(L))$ and $(G \circ T)$ components may be used directly.

4.4. T^3, Juicer, HTK and Sphinx

A major feature of the toolkit is the ability to automatically generate cascades for HTK or Sphinx models and for either the T^3 or Juicer WFST decoders. In order to illustrate this added flexibility and further show the relatively equal performance of

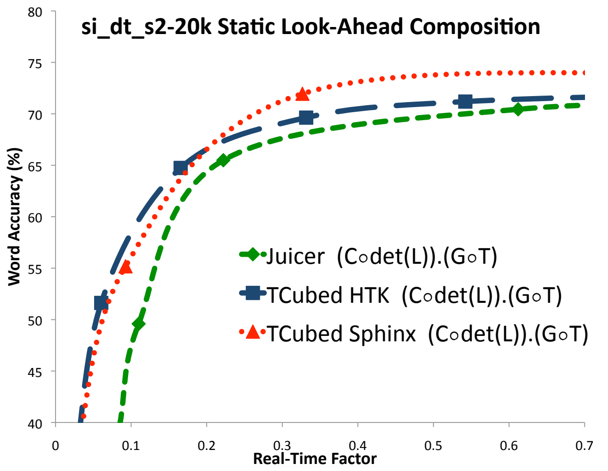

Figure 5: Cascade build comparison for the si_dt_s2-20k task using the T^3 and Juicer decoders and both HTK and Sphinx AMs using SLA composition.

these combinations an additional cross-comparison experiment was performed utilizing the *si_dt_s2-20k* task. The results of this experiment are displayed in Figure 5.

5. Discussion

Overall the results serve to confirm the integrity of the toolkit, backup past experience and replicate results from recent work. The results described in 4.2 show little tangible difference among the different optimization procedures. This was also reflected in the size of the cascades, which were nearly identical in all three cases. This is most likely a consequence of the small size and simplicity of the nov92-5k training data, however we plan to explore this further in future. Larger and more complex inputs tend to benefit more from final optimization procedures. Nevertheless the most substantial relative improvements result from the initial determinization procedure.

The SLA experiments from 4.3 further confirm the superiority of the look-ahead composition algorithm, even where static cascades are concerned. The composition algorithm itself is an improvement over standard composition, but the most substantial gains result from the ability to avoid the otherwise costly $det(L \circ G)$ operation. In the above experiments as well as additional results for larger cascades which are not reported in this paper, the SLA approach resulted in an average memory savings of roughly 50%, and an average overall time savings of nearly 80%. The SLA build also benefited from use of the tropical semiring. This is due to the fact that determinization of the un-weighted (and thus non-stochastic) L transducer in the log semiring produces an undesirable re-weighting which negatively affects the final cascade. This issue can be avoided either by performing the entire build in the tropical semiring or performing just the $det(L)$ operation in the tropical semiring.

The cross-comparison results described in 4.4 serve to replicate our results from [10], albeit using the SLA build. The small performance variation among the AM types and T^3 versus Juicer again suggest that there is not much technical motivation to overtly favor any particular combination. Rather the availability of resources and existing expertise should guide development choices.

Finally, although T^3 supports GPU-based computation of acoustic likelihood scores, we have omitted these results here because the GPU acceleration, combined with the more accurate logsum operation tends to overshadow most cascade optimization effects.

6. Conclusion and Future Work

In this work we introduced Transducersaurus, a new open source software toolkit for building and manipulating WFST-based ASR cascades which provides integrated support for the T^3 and Juicer WFST decoders and both HTK and Sphinx acoustic models. We have shown the effectiveness of the toolkit for a variety of standard tasks, and furthermore provided a detailed explanation of the SLA build process and its merits and caveats. The ASR application development process is often iterative, and these results suggest that by utilizing a simplified build chain and the SLA composition approach overall efficiency can be greatly improved.

In future we plan to further expand the range of available operations, expand the current DSL build syntax, provide integrated OOV support and introduce parallel support for the AT&T fsmtools. Experimental results for a wider variety of languages and model inputs are also underway. We hope that this toolkit will facilitate learning as well as more efficient work in this area, and promote further discussion.

7. Acknowledgements

Thanks to Paul Dixon for many fruitful discussions, and specifically the suggestion to reevaluate the semiring choice for the SLA build. Thanks to Nikolay Shmyrev as well for providing helpful early feedback on the toolkit.

8. References

[1] Mohri, M., "Finite-state transducers in language and speech processing," Computational Linguistics, Vol. 23, 1997.

[2] Allauzen, C., Riley, "OpenFst: A General and Efficient Weighted Finite-State Transducer Library," tutorial, SLT 2010.

[3] Allauzen, C., Mohri, M., Riley, M., Roark, B., "A Generalized Construction of Integrated Speech Recognition Transducers," in Proc. ICASSP, pp. 761-764, 2004.

[4] Novak, J., `code.google.com/p/transducersaurus/`

[5] Dixon, P., Caseiro, D., Oonishi, T., Furui, S., "The Titech Large Vocabulary WFST Speech Recognition System," in Proc. ASRU, pp. 1301-1304, 2007.

[6] Moore, D., Dines, J., Magimai Doss, M., Vepa, O., Cheng, O., Hain, T., "Juicer: A Weighted Finite State Transducer Speech Decoder," in Proc. Interspeech, pp. 241-244, 2005.

[7] Paul, D., B., Baker, J., M., "The Design for the Wall Street Journal-based CSR Corpus," in Proc. ICSLP 92, pp. 357-362, 1992.

[8] Cieri, C., Miller, D., Walker, K., "The Fisher Corpus: a Resource for the Next Generations of Speech-to-Text," in Proc. LREC, pp. 69-71, 2004.

[9] Allauzen, C., Mohri, M., Roark, B., "Generalized Algorithms for Constructing Language Models," in Proc. ACL, pp.40-47, 2003.

[10] Novak, J., Dixon, P., Furui, S., "An Empirical Comparison of the T^3, Juicer, HDecode and Sphinx3 Decoders," in Proc. InterSpeech 2010, pp. 1890-1893, 2010.

[11] Mohri, M., "Semiring Frameworks and Algorithms for Shortest Distance Problems," J. Automata, Languages, and Combinatorics, vol. 7, no. 3, pp. 321-350, 2002.

[12] Allauzen, C., Riley, M., Schalkwyk, J., "A Generalized Composition Algorithm for Weighted Finite-State Transducers," InterSpeech 2009, pp. 1203-1206, 2009.

[13] Vertanen, K., "Baseline WSJ Acoustic Models for HTK and Sphinx: Training Recipes and Recognition Experiments," Cavendish Laboratory, University of Cambridge, 2006.