Open-Source WFST tools for LVCSR Cascade Construction *

◎ Novak Josef, 峯松信明, 広瀬啓吉 (東大)

1 Introduction

This paper introduces an open source toolkit, Transducersaurus [1], leveraging OpenFst [2] which can be used to build WFST-based networks for LVCSR tasks. The toolkit provides a set of classes for generating each of the fundamental components of a typical WFST ASR cascade, including HMM, context-dependency, lexicon, language model and silence class models. The toolkit further implements a simple scripting language to facilitate cascade construction with various optimization methods and provides support for the T^3 and Juicer WFST decoders as well as a new WFST decoder currently under development at U. Tokyo, and both Sphinx and HTK format acoustic models. Results for a standard WSJ tasks are also provided. The results illustrate the flexibility of the toolkit as well as the tradeoffs inherent in various build algorithms.

2 WFST-based ASR Cascades

The Weighted Finite-State transducer (WFST) approach to Automatic Speech Recognition (ASR) has gained considerable popularity in recent years [3]. The WFST approach provides a flexible, unified mathematical framework for rendering each of the component knowledge sources typically utilized to build an ASR system. An integrated network is then produced by iteratively combining and optimizing the 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 [4]. Finally the H model performs a mapping from HMM distributions to triphone sequences is often described in the literature. The Transducersaurus toolkit provides individual programs suitable for constructing each of these component transducers, as well as a unified integration tool in the form of *transducersaurus.py*. For more details on the structure, algorithms and complexity associated with the component models please see [4].

2.1 Integration and Optimization

In order to utilize the component models in a WFST-based decoder it is necessary to first integrate them into a unified recognition network. This is typically achieved through iterative use of *composition*. The un-modified result of composition is, however not efficient, thus it is often necessary to optimize the network through the use of techniques such as *determinization*, *minimization*, and $weight-pushing$. A simple three-component integration and optimization algorithm then might take the form, $C \circ det(L \circ G)$ where the $\circ$ symbol refers to *composition*, and *det* refers to *determinization*. Much more complex construction algorithms are possible, however in general optimizations incur costs in terms of construction time, memory and storage requirements. It is important to strike a reasonable balance between these competing interests and the toolkit provides a means to achieve this with a minimum of overhead.

3 Cascade Integration with Transducersaurus

The toolkit provides an integration tool in the form of *transducersaurus.py*, which allows the user to provide a single cascade integration command following a simple DSL syntax, which will automate the build process. In past this required manual integration via OpenFst, or custom build programs, both of which are time consuming and error prone. An example of a complete build command,

```
./transducersaurus.py --tiedlist tiedlist
--hmmdefs hmmdefs --grammar my.lm
```

* Open-Source WFST tools for LVCSR Cascade Construction. By Josef Novak, Nobuaki Minematsu, and Keikichi Hirose, The University of Tokyo.

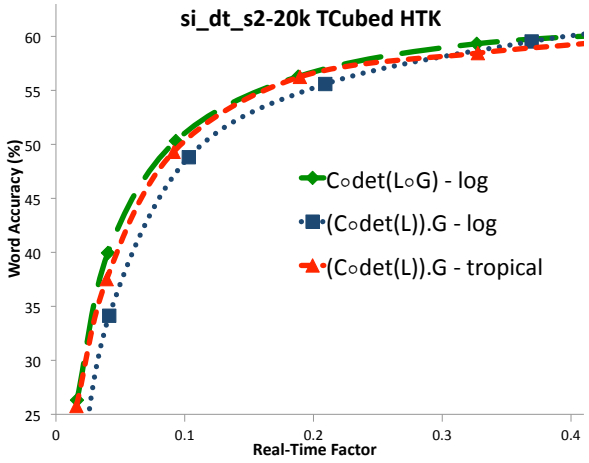

Fig. 1 SLA cascade build comparison for the si_dt_s2-20k task using the T^3 decoder and HTK AM.

```
--lexicon my.lex --amtype htk
--command "(C*det(L)).G"
```

The above command will construct an integrated cascade utilizing HTK acoustic models, and lookahead composition as described in [7].

4 Experiments

Experiments were conducted comparing static lookahead composition and standard composition. The recognition task, *si_dt_s2-20k*, focused on the WSJ1 Hub2 set comprising 207 sentences and utilizes the standard WSJ 20k non-verbalized closed bigram language model and corresponding vocabulary. Open source acoustic models from [5] were used, and parameters for the T^3 and Juicer decoders were specified as in [6]. Results for the experiment are depicted in Fig. 1.

The SLA cascade construction from [7] has advantages even in static mode. First, the composition operation is both faster and more efficient in terms of memory consumption. Second, 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 on-the-fly composition, the precomputed $(C \circ det(L))$ and G components may be used directly for an even greater savings.

5 Conclusions and Future work

The SLA experiments further confirm the superiority of the look-ahead composition algorithm, even where static cascades are concerned. In the above experiments 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 unweighted 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. These experiments show the basic flexibility of the toolkit, but for more in-depth discussion the reader is referred to [8].

In future we plan to further extend the toolkit to support a wider range of optimization algorithms, an d evaluate the SLA builds in an on-the-fly context with the decoder currently under development.

参考文献

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

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

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

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

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

[6] 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.

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

[8] Novak, J., Minematsu, M., Hirose, K., "Painless WFST cascade construction for LVCSR - Transducersaurus," InterSpeech 2011, accepted.