OPENDCD: OPEN SOURCE WFST DECODING TOOLKIT

Paul R. Dixon *Josef R. Novak*

Yandex Zurich
Switzerland

ABSTRACT

In this paper we introduce *OpenDcd* a lightweight and portable Weighted Finite State Transducer based speech decoding toolkit written in C++. OpenDcd is a collection or tools for speech recognition decoding, cascade construction and related conversion and results manipulation. The toolkit is faster and requires substantially less memory than other open source alternatives. We hope this toolkit rescues from inefficient full $HCLG$ compilations.

Through sophisticated use of templates OpenDcd can be built against Kaldi and re-use the frontend and acoustic scoring. This Allows OpenDcd to be used easily as a complete speech recognition system.

***Index Terms*—** WFST, Decoding, Speech recognition, Open source

1. INTRODUCTION

Open source and reproducible results are speech recognition are an essential part of modern science. Unfortunately, speech recognition seems to lag other disclipines for open source toolkits and open access journals.

Recently, speech recognition is catching up. There are now many excellent toolkit for manipulating automata [1], building language models [**?**], all the way up constructing entire sophisticated speech recognitions systems [2] and other components in the speech recognition system [3].

However, the state of modern decoding libraries is still lagging. In this paper we describe the OpenDcd release a set of tools under active development for constructing and decoding finite state speech recognition cascades. The toolkit makes use of OpenFst for the underlying finite state representations and operations, and the OpenGrm [4] for language model conversion.

We follow the Google OpenFst style guide for coding.

One extremely important feature of the toolkit is it can be built as The toolkit can be built as standalone decoder with no dependencies, or as addon with Kaldi. For the latter there is full access to all of the IO mechanism and acoustic models available in Kaldi. In later sections we discuss the lattice generation mechanisms that make it possible to use the OpenDcd in many of Kaldi's training and rescoring pipelines.

This means if you have a set of converted Kaldi models no additional software needs to be installed. OpenDcd without Kaldi can be deployed to any platform and the recognition system run.

Furthermore the modular nature of OpenDcd's design allow for new types of acoustic model for example LSTM or RNN to plugged into the decoder.

The library started off as an internal project for research use we are releasing the project under an open source licence. One design goal was to allow for customization via `C++` templates.

The focus of the toolkit is provide a fast easy-to-use WFST decoder, with minimal dependencies that can be used as a tool research or as part of a larger software system.

OpenDcd is by far the best open source decoder avaliable anyway. It is a state-of-the-art and highly extensiable toolkit for speech recognition. The highly modular and pluggable architecture is designed to allow for researcher to integrate new models and algorithms.

Goals of the OpenDcd toolkit: **Simplicity**

- Concise code - only 5000 lines
- It *just works* simple to build with make
- No build dependencies
- One decoding command
- Streamlined

2. OVERVIEW

There are several components in the system. The cascade tools, the decoder core and utility tools. In addition we provide a set of helpful scripts and Kaldi style example that show how to integrate the decoder into a standard Kaldi reciee.

This paper presents an overview of this new toolkit.

The main intended use of OpenDcd is:

- Provide a research base for decoding algorithms and comparison.
- The provide a drop in replacement to the Kaldi decoder
- To provide a standalone decoder core that can be used the foundations of large speech recognition system.

2.1. Weighted Finite State Transducers

3. DECODER CORE

3.1. Extensability

The decoder in heavily influenced by OpenFst and makes heavy use of templates for extensability. In additon python bindings are provided that allow Python language to extend the decoder and access the internals. The former is more difficult to implement and requires a good C++ understanding. This is gives best performance. The later is usually quicker to implement but comes with certain limitations and does not have full access in some cases.

3.2. Transition Model

The search component assumes that the transition model[1] will consume a single frame of speech per time step. The transition module (nearly) full encapsulates the acoustic model. The decoupling allows for specialized token expansion algorithm to handle any topolgy of HMM or even allow for totally arbitrary non-finite state models.

The recognition output is written as either an OpenFst FAR file or a Kaldi table. Each of which can stored in the tropical semiring or a Kaldi lattice weight. In addition tools are provided to convert between FAR and table formats. To batch manipulate the results we provided a reimplementation of the AT&T `farfilter` command. `farprintnbeststrings` adds AT&T functionality missing from OpenFst that is the ability to print an nbest list from lattices contained within a FAR file, we make use of two different algoritms.

3.3. Customization

The core of the decoder is the `ArcDecoder` class that is parametrized on a transition model, Fst and lattice.

```
template<class T, class F, class L> class ArcDecoder;
```

For example the following code in Kaldi mode will create a decoder that uses cascade in the tropical semiring, has HMM transition model with GMM state output distributions and generates lattice.

```
typedef HMMTransitionModel<DecodableAmNnet>
ObservationModel; ArcDecoder<StdFst, ObservationModel, StdLattice>
Decoder;
```

[1] Our model and definition of TransitionModel are different to that used in Kaldi.

3.4. Lattice Generation

The decoder supports lattice generation based on the *phone-pair* approximation and a *full* lattice generation strategy. In addition there a two approaches for generating full state level lattices. The first is compose a HMM level.

The decoder can be simply configured to generate lattice any of the below formats Tropical, Log, Lattice and CompactLattice. Not there are certain restrictions. In addtion the toolkit provides the code necesssary to build dynamic shared objects that allow the OpenFst tools to work with Kaldi's custom weight semirings.

3.5. Parallelization

The recent work on acoustic lookahead and parallelization only focus on dynamic decoders or WFST decoder that use lookahead composition. We introduce a different parallelization scheme that uses a worker thread to pre-emp upcoming states and precompute the composition state. In practise we find that this heuristic can give an approximately 20% speed-up.

3.6. Factorization

The de-coupled transition model implemenation not allows for non HMM based models to be easily integrated into the decoder but allows for arbitrary topologies that may be the result of more complex models or more complex factorizations of the search space.

4. INSTRUMENTATION, DEBUGGING AND LOGGING FEATURES

The decoder has several built analysis mechanism, these are the call analyzer, memory analyzer and the search space analyzer. For fined grained analysis we make use of the ShinyProfiler which generate very detailed call graphs.

5. CASCADE TOOLS

The toolkit includes a very fast build procedure for constructing a recognition *cascade* from a language, lexicon and context-dependency models. In machine translation experimental pipelines have become more popular [**?**]. However, most of these are very similar to well known Unix make tool. We make use of `make` as our basic tool to track the components. If a compoenent in the cascade is modified make we only rebuild.

The FST framework provides many operations for combining and manipulating automata. To construct a speech recognition cascade and recognize speech typically requires three core operations: Composition, Determinization and Shortestpath. The later is handled be the decoder command and by careful component construction. We

The ngram model uses the direct minimalistic construction proposed by Caseiro [5] and implemented by OpenGrm. In practise we find that the minimized LM improves the performance of the lookahead composition. In addtion we find that pushing the labels are forward as possible can also help the efficiency substantially.

6. UTILITY TOOLS

In addition to main decoding command, the OpenDcd distribution contains many useful tools. These include the following:

7. KALDI INTEROP

With recent importance of Kaldi as the foundation of most speech recognition research. OpenDcd provides two levels of compatability. When the decoder is built for standalone operation we support native reading of Kaldi features and writing of the table format.

8. EVALUATIONS

In this section we demonstrate the performance of the decoder on several tasks.

8.1. Wall Street Journal

9. SUMMARY

In this paper we have described OpenDcd a toolkit. In other papers we intend to describe the algoritms and cusomization in more depth. In future release we plan to release one-pass neural network re-scoring.

We hope readers to will download OpenDcd from `http://www.opendcd.org` and try the easy to use scripts and command line tools with their existing Kaldi setups. We hope that this library will become the default library and toolkit for many years to come.

10. REFERENCES

[1] C. Allauzen, M Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A general and efficient weighted finite-state transducer library," in *Proc. of CIAA 2007*, 2007, pp. 11–23.

[2] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlıcek, Yanmin Qian, Petr Schwarz, et al., "The kaldi speech recognition toolkit," in *Proc. ASRU*, 2011.

[3] Josef R. Novak, Nobuaki Minematsu, and Keikichi Hirose, "WFST-based grapheme-to-phoneme conversion: Open source tools for alignment, model-building and decoding," in *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, July 2012, pp. 45–49.

[4] Brian Roark, Richard Sproat, Cyril Allauzen, Michael Riley, Jeffrey Sorensen, and Terry Tai, "The opengrm open-source finite-state grammar software libraries.," in *ACL (System Demonstrations)*, 2012, pp. 61–66.

[5] Diamantino Caseiro and Isabel Trancoso, "Transducer composition for on-the-fly lexicon and language model integration," in *Proc. ASRU*, 2001, pp. 393–396.