Towards the Optimal Minimization of a Pronunciation Dictionary Model

Simon Dobrišek[1], Janez Žibert[2], and France Mihelič[1]

[1] University of Ljubljana, Faculty of Electrical Engineering,
Tržaška 25, SI-1000 Ljubljana, Slovenia
{simon.dobrisek,france.mihelic}@fe.uni-lj.si
[2] University of Primorska, Primorska Institute of Natural Sciences and Technology,
Muzejski trg 2, SI-6000 Koper, Slovenia
janez.zibert@upr.si

Abstract. This paper presents the results of our efforts to obtain the minimum possible finite-state representation of a pronunciation dictionary. Finite-state transducers are widely used to encode word pronunciations and our experiments revealed that the conventional redundancy-reduction algorithms developed within this framework yield suboptimal solutions. We found that the incremental construction and redundancy reduction of acyclic finite-state transducers creates considerably smaller models (up to 60%) than the conventional, non-incremental (batch) algorithms implemented in the OpenFST toolkit.

PREPRINT
For personal use only

1 Introduction

A spoken-language model is commonly designed as a cascade of deterministic, non-deterministic and probabilistic finite-state transducers (FSTs). Such a composition can be seen as a unified, multi-level, probabilistic finite-state network, where different levels correspond to the various spoken-language constraints, such as grammar, lexicon, pronunciation rules, acoustic-phonetic models, etc [1]. This approach reduces the problem of automatic speech recognition (ASR) to the problem of searching for the most probable path through a finite-state network, given a sequence of acoustic speech observations [2].

Large-vocabulary (LV) ASR systems require very large finite-state models that may be composed of tens of millions of states and transitions. Any implementation of a LV-ASR system requires an optimization of the model in terms of its size and algorithmic complexity. The concept of weighted FSTs and the toolkits developed by Mohri et al. [3] provide a general representation and an algorithmic framework for such an optimization. The framework is implemented in the toolkit called OpenFST [4]. This toolkit provides efficient implementations of the conventional algorithms for constructing, combining, optimizing, and searching the generalized weighted FSTs.

We explored the concepts and algorithms provided by the OpenFST library, where we focused on the algorithms for optimizing FSTs in terms of size. We have limited our research to the construction of a pronunciation dictionary component of the whole finite-state model [5]. We experimented with unweighted acyclic FSTs to model large pronunciation dictionaries with hundreds of thousands of word forms. To our

P. Sojka et al. (Eds.): TSD 2010, LNAI 6231, pp. 267–274, 2010.
© Springer-Verlag Berlin Heidelberg 2010

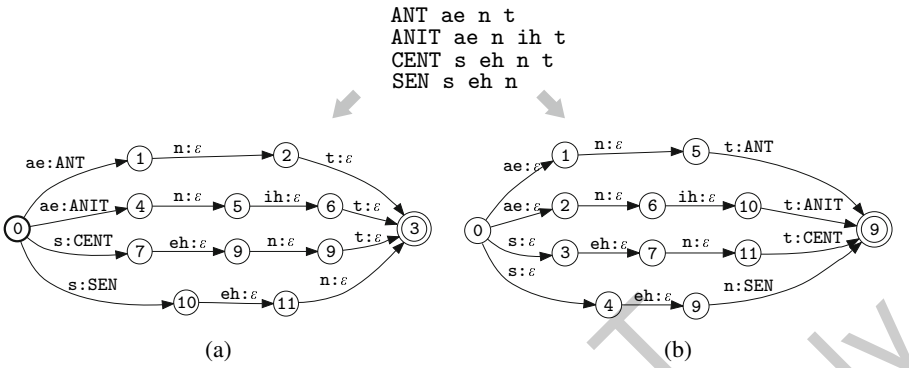

Fig. 1. The two possible initial unweighted acyclic FST that were constructed from the given list of words with their pronunciation. The states are represented by circles and marked with their unique number. The initial state is represented by a bold circle and the final state by a double circle. The input label `i` and the output label `o` of a transition are marked on the corresponding directed arc by `i:o`.

surprise we found that the incremental construction and optimization of such FSTs yield considerably better results than the conventional non-incremental approach. We developed an alternative algorithm for the incremental construction and redundancy reduction of unweighted acyclic FSTs. The algorithm proved to be faster and creates considerably smaller FSTs than the original OpenFST algorithms.

2 Pronunciation Dictionary Model

Word pronunciations are assumed to be finite and consequently they can be encoded using acyclic FSTs. Without any loss of generality we restricted our experiments to the unweighted FSTs as the extension to the weighted FSTs seems to be straightforward.

Let us assume that the initial FST that represents the pronunciation dictionary is constructed from a list of words with their pronunciations, as depicted in Fig. 1. The symbol ε denotes the empty string that is defined as the identity element with the string concatenation operation. Accordingly, the output label that represents a word can be placed on any of the directed arcs between the initial and final state, and consequently, different FSTs can be constructed from the same word list. However, such different FSTs are considered to be equivalent since they encode the same word pronunciations.

Fig. 1 shows the two possible FSTs that are considered to be equivalent. We could say that the output labels of FST(a) are pushed towards the initial state, and the output labels of FST(b) are pushed towards the final state. Since we are restricted to the unweighted FSTs, there is no need to use different final states for each of the word pronunciations.

3 Redundancy Reduction

In a deterministic FST, no two transitions from the same state share the same input label, which is not the case with the two shown initial FSTs. A deterministic FST has

PREPRINT
For personal use only

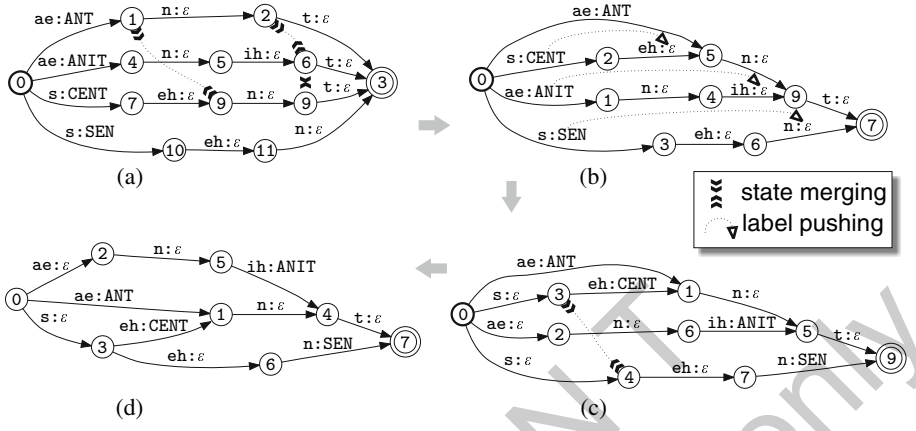

Fig. 2. The three steps of the FST redundancy reduction via the initial FSA minimisation, the intermediate output label pushing towards the final state, and the final FSA determinisation

an advantage over the equivalent nondeterministic one in terms of lower redundancy. As shown by Mohri et al. [1] the nondeterminism of a FST can be eliminated, or at least reduced, by first encoding it as a finite-state acceptor (FSA), i.e., each pair of input and output labels is treated as one label, and then using the classical FSA determinisation algorithm [6] the obtained nondeterministic FSA is transformed into an equivalent deterministic FSA.

The redundancy of the obtained deterministic FSA can be reduced even further by using the classical minimization algorithm that merges its equivalent states [7]. The two states of a deterministic FSA are said to be equivalent if exactly the same sequence of symbols labels the paths from these states to the final state. The final minimal FSA can then be decoded back as a FST.

However, this redundancy-reduction procedure via the FSA determinisation and minimisation does not yield the minimum possible FST being equivalent to the initial FST. The procedure has to be combined with the output label pushing that preserves the equivalency of the transformed FST and enables a further redundancy reduction. The combined procedure is illustrated in Fig. 2.

Fig. 2 shows the initial FST(a) constructed from the word pronunciation list given in Fig.1. The initial FST is encoded as a FSA, which is then determinised and minimised as described above. The determinisation step is actually not required since, due to the left-most position of the output labels in the initial FST(a), the initial FSA is already deterministic. Fig. 2 shows the FST(b) that resulted from the FSA minimisation. As can be seen from the figure, further redundancy reduction is possible if the output labels are pushed towards the final states as far as the equivalency of the transformed FST is still preserved. FST(c) is equivalent to FST(b), and since it is nondeterministic its redundancy can be further reduced. The reduction is performed by, again, first encoding the FST as a FSA, performing its determinisation and then obtaining the final FST(d) that is decoded back from the determinised FSA. The described procedure is

PREPRINT
For personal use only

conventional and can be performed using the algorithms that are implemented in the OpenFST toolkit.

An important observation that can be drawn from the analysis of the described procedure is that the final FST is not necessarily the minimum possible FST that is equivalent to the initial FST. The procedure consists of three steps that are optimized locally; however, they are mutually dependent. For instance, the suboptimal redundancy-reduction at the first minimisation step may lead to a better total redundancy reduction obtained at the third determinisation step. One could also start the procedure with the initial FST(b) shown in Fig. 1 and perform the FSA determinisation at the first step, the output label pushing towards the initial state at the second step, and the FSA minimisation at the last step. Our experiments revealed that the two possible resulting FSTs can have different sizes, even though they represent the same pronunciation dictionary.

4 Incremental FST Construction and Redundancy Reduction

The FST redundancy-reduction procedure discussed in the previous section is based on the size optimisation of the whole initial FSTs. An alternative approach would be to construct a FST and reduce its redundancy incrementally, word by word. In this procedure, a left-to-right single-path FST is created for each given word pronunciation. A union between this FST and the incrementally growing FST, which is being constructed to represent the complete pronunciation dictionary, is then created. After each such union, the described redundancy reduction can be performed as usual. This procedure is illustrated in Fig. 3.

The incremental procedure can be performed by combining the conventional algorithms implemented in the OpenFST toolkit; however, such a procedure has a very

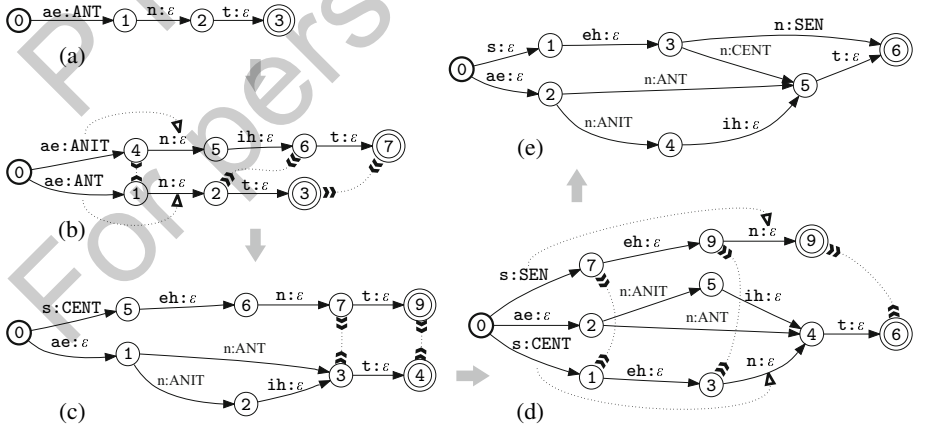

Fig. 3. Incremental construction of a FST from the list of word pronunciations. The usual redundancy reduction is performed after each union between the newly-added FST, representing the newly added word pronunciation, and the existing incrementally-growing FST being constructed to represent the complete pronunciation dictionary.

PREPRINT
For personal use only

high (exponential-time) complexity. Our experiments showed that it can take several hours to create a FST from a 50k-word pronunciation dictionary. But to our surprise, the incremental procedure yields considerably smaller FSTs. This can be seen from the comparison of the given figures. The final FST(e) shown in Fig. 3 has one state and one transition less than the final FST(d) shown in Fig. 2; however, both FSTs represent the same pronunciation dictionary.

We developed an alternative algorithm for the incremental construction and redundancy reduction of acyclic FSTs that has a much lower (linear-time) complexity. The algorithm was actually developed independently from the algorithms that are provided in the OpenFST toolkit. For historical reasons (the existing computer code, etc) the first version of the proposed algorithm was developed for the Moore type of FSTs [5]. The algorithm is now improved to support the Mealy type of FSTs that are also supported by default by the OpenFST toolkit.

The algorithm incrementally constructs a FST by merging the newly added single-path FST, which represents the newly added word pronunciation, with the incrementally growing FST being constructed. The algorithm is composed of two parts that we call the incremental tail-merge minimisation and the incremental head-merge determinisation. The two algorithms are performed for each of the added word pronunciations.

First, a single-path FST that represents the newly added word pronunciation is constructed as usual. The output (word) label is put on the initial state transition. A union between the newly added FST and the incrementally growing FST is then performed, where the final states are merged into a single final state. The following two algorithms are then performed to reduce the FST redundancy.

The tail-merge minimisation: The algorithm iterates backwards from the final state to the newly added states and merges them with the existing states, if certain conditions are satisfied. The algorithm involves the following steps:

Step 1: The current state becomes the final state of the unified FST.

Step 2: For each of the current state's predecessors the following conditions are verified:

- The given predecessor state is not the initial state and also not the newly added predecessor state;
- The input label of the transition from the given predecessor state to the current state is the same as the input label of the transition from the newly added predecessor state to the current state;
- The given predecessor state has only one successor (the current state) and the transition to the current state has the ε-output label.

If all the above conditions are satisfied for the given predecessor state then:

- The given predecessor state and the newly added predecessor state are merged into one state, and all the transitions to and from the merged state are rearranged, as necessary;
- The current state becomes the merged state.
- Recurse back to **Step 2**.

Step 3: Quit and return the current state.

PREPRINT
For personal use only

The head-merge determinisation: The algorithm is performed after the tail-merge minimisation. This algorithm performs state merging from the opposite side of the unified FST. It also performs output-label pushing from transitions to transitions, if necessary. The algorithm is defined as follows:

Step 1: The current state becomes the initial state of the unified FST.
Step 2: For each of the current state successors the following conditions are verified:
- The given successor state is not the last state that was merged by the tail-merge algorithm and also not the newly added successor state;
- The input label of the transitions to the given successor state is the same as the input label of the transitions to the newly added successor state;
- The transition to the given successor state has the ε-output label, or if not so, the given successor state has only one predecessor and also only one successor with the ε-output label transition.

If all the above conditions are satisfied for the given successor state then:
- If the transition to the given successor state or to the newly added successor state has a word-output label, then this output label is pushed forward to the successor's transition to its single successor;
- The given successor state and the newly added successor state are merged into one state, and all the transitions to and from the merged state are rearranged, as necessary;
- The current state becomes the merged state;
- Recurse back to **Step 2**.

Step 3: Quit and return the current state.

The presented algorithm constructs a FST in the same way as depicted in Fig. 3. It actually creates the same final FSTs as created by the generalized incremental procedure described at the beginning of this section.

In general, the presented incremental procedure also does not necessarily create the minimum possible FST that represents the given pronunciation dictionary. It is obvious that the final result depends on how the word sequence that is processed by the algorithms is ordered. We found experimentally that better results can be achieved if a given word pronunciation dictionary is not lexically ordered. Our experiments actually showed that the size of the obtained FSTs can be up to 10% smaller if the input word list is randomly shuffled.

5 Experimental Results

All the discussed algorithms were evaluated by performing experiments with several large pronunciation dictionaries. The OpenFST toolkit algorithms were used to perform the conventional non-incremental FST creation procedures and the obtained FST sizes were compared to the sizes obtained by the proposed alternative incremental algorithm.

In this paper, we report the results of experiments with dictionaries from three different language groups, namely, US English, Slovenian, and Italian. The first one is the well-known CMU Pronouncing Dictionary for North American English (CMU-US). The version we used contains over 133k phonetically transcribed words. The

PREPRINT
For personal use only

Table 1. The sizes (number of states/number of transitions) of the FSTs obtained from the three pronunciation dictionaries

Optimisation	CMU-US (133k)	MTE-SL (201k)	FST-IT (401k)
None	717,512/850,868	1,688,273/1,890,225	3,903,866/4,314,710
OFST-SFI	80,648/214,004	106,947/308,899	352,739/763,489
OFST-SFF	64,002/197,358	115,141/317,093	237,253/647,779
ITHM-ORD	33,473/166,829	34,974/236,926	99,801/510,645
ITHM-SHF	33,206/166,562	19,689/221,641	38,511/449,355

second one is the Multext-East Slovene Dictionary (MTE-SL). The version we used contains over 201k words transcribed using our TTS grapheme-to-phoneme converter. The Italian pronunciation dictionary (FST-IT) was taken from the Festival TTS toolkit and it contains over 410k transcribed words.

Let OFST-SFI and OFST-SFF denote the two versions of the conventional non-incremental FST creation and the redundancy-reduction procedures that were performed using the OpenFST algorithms, as described in Sec. 3. The first version starts with the initial FST having output labels pushed towards the initial state and the second one starts with the initial FST having output labels pushed towards the final state.

Then let ITHM-ORD and ITHM-SHF denote the proposed incremental head- and tail-merging algorithms, where ITHM-ORD denotes the lexically ordered input pronunciation dictionary, and ITHM-SHF denotes the the input pronunciation dictionary with the randomly shuffled word list. Table 1 shows the sizes of the obtained FSTs that represent one of the three pronunciation dictionaries. As can be seen from the results, the proposed incremental algorithms create considerably smaller FSTs, especially for the more inflected languages, like Slovenian and Italian. This is to be expected since both languages have abundant inflections and many word forms with the same suffixes and/or prefixes.

The proposed incremental algorithm is also considerably faster than the conventional non-incremental procedure. For instance, our Java implementation [8] of the proposed algorithm generated a FST from the 201k-word example in less than 6 seconds on a common PC (the OpenFST non-incremental procedure implemented in C++ required almost 25 seconds). The correctness of the implementation was systematically verified by the algorithm that generates all the possible input/output sequences from the given FST and these sequences were compared to the original pronunciation dictionaries.

6 Conclusions

The presented incremental FST construction and redundancy-reduction procedure does not create the minimum possible FST for the given pronunciation dictionaries, however, the global minimum cannot be far from the presented results since the number of FST transitions is close to the number of words in the input dictionaries (see Table 1).

PREPRINT
For personal use only

References

1. Mohri, M., Pereira, F., Riley, M.: Weighted Finite-state Transducers in Speech Recognition. Computer Speech and Language 16, 69–88 (2002)
2. Jelinek, F.: Statistical Methods for Speech Recognition. The MIT Press, Cambridge (1998)
3. Mohri, M., Pereira, F.C.N., Riley, M.: Speech Recognition with Weighted Finite-State Transducers. In: Rabiner, L., Juang, F. (eds.) Handbook on Speech Processing and Speech Communication, Part E: Speech recognition. Springer, Heidelberg (2008)
4. Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., Mohri, M.: OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In: Holub, J., Žďárek, J. (eds.) CIAA 2007. LNCS, vol. 4783, pp. 11–23. Springer, Heidelberg (2007)
5. Dobrišek, S., Vesnicer, B., Mihelič, F.: A Sequential Minimization Algorithm for Finite-State Pronounciation Lexicon Models. In: Proceedings of Interspeech 2009, International Speech Communication Association, Brighton, UK, pp. 720–723 (2009)
6. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers, Principles, Techniques and Tools. Addison Wesley, Reading (1986)
7. Revuz, D.: Minimisation of Acyclic Deterministic Automata in Linear Time. Theoretical Computer Science 92, 181–189 (1992)
8. Spider – The Demonstration Implementation of the Algorithm for the Incremental Construction and Redundancy Reduction of Unweighted Acyclic FSTs, `http://luks.fe.uni-lj.si/spider`
9. Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., Woodland, P.: The HTK Book (for HTK Version 3.4). Cambridge University Engineering Department, Cambridge (2006)

PREPRINT
For personal use only