INTERSPEECH 2009 BRIGHTON

A Sequential Minimization Algorithm for Finite-State Pronunciation Lexicon Models

Simon Dobrišek, Boštjan Vesnicer, France Mihelič

Faculty of Electrical Engineering, Ljubljana University, Ljubljana, Slovenia
simon.dobrisek@fe.uni-lj.si

10.21437/Interspeech.2009-246

Abstract

The paper first presents a large-vocabulary automatic speech-recognition system that is being developed for the Slovenian language. The concept of a single-pass token-passing algorithm for the fast speech decoding that can be used with the designed multi-level system structure is discussed. From the algorithmic point of view, the main component of the system is a finite-state pronunciation lexicon model. This component has crucial impact on the overall performance of the system and we developed a sequential minimization algorithm that very efficiently reduces the size and algorithmic complexity of the lexicon model. Our finite-state lexicon model is represented as a state-emitting finite-state transducer. The presented experiments show that the sequential minimization algorithm easily outperforms (up to 60%) the conventional algorithms that were developed for the static global optimization of the transition-emitting finite-state transducers. These algorithms are delivered as part of the AT&T FSM library™ and the OpenFST library.
Index Terms: finite-state lexicons, finite-state networks, finite-state transducers, sequential minimization

1. Introduction

Common assumption about the human-speech communication process is that it involves several stochastic finite-state mappings between symbol sequences. Accordingly, a composition of deterministic, non-deterministic and probabilistic finite-state transducers (FSTs) is commonly used to model this process. Such a composition can be seen as a unified multi-level probabilistic finite-state network (FSN), where different levels correspond to the various spoken-language constraints, such as grammar, lexicon, pronunciation rules, context-dependent phonetic models, etc. This approach reduces the problem of automatic speech recognition (ASR) to the problem of searching for the most probable path through such a network that could produce a given sequence of acoustic speech observations [2].

FSTs are finite-state machines (FSMs) with an output, and two concepts of such machines exist, which are known as Mealy and Moore machines. In a Moore machine the output is produced by its states, while in a Mealy machine the output is produced by its transitions.

Hidden Markov models (HMMs) that are commonly used for context-dependent acoustic-phonetic models are often represented as state-emitting probabilistic Moore machines. On the other hand, the higher components of the whole finite-state speech model (grammar, pronunciation lexicon etc) are often represented as transition-emitting weighted Mealy machines. This is the case with the mentioned AT&T FSM library™ [3] and the OpenFST library [1]. This is normally not a problem, as it can be easily shown that for each Mealy machine there is an equivalent Moore machine, and vice versa. However, the differences between the two concepts should not be ignored when one develops algorithms for model composition, determinisation, minimization, etc.

The performance of large-vocabulary (LV) ASR systems, which are based on the above modelling concept, critically depends on the size and algorithmic complexity of the whole finite-state speech model. Due to the limited computational resources of current computer systems, any implementation of a large-vocabulary ASR system requires a certain optimization of the speech model. 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. This concept and the two mentioned toolkits are widely accepted and used by the speech-technology community.

One of our main research goals is the development of our own state-of-the art LV-ASR system. The main motivation for this is our research commitment to developing such a system for our mother tongue, the Slovenian language. Our language is richly inflected with a grammar that is similar to other Slavic languages. However, it has certain distinctive characteristics, like a dual grammatical number, two accentual norms, and abundant inflections. Although Slovene is basically a subject-verb-object language, the word order is very flexible, and is often adjusted for emphasis or stylistic reasons. These characteristics provide a good reason, as to why we should not strictly follow the concepts that are used for the less inflected languages with a less flexible word order, like English or German. We also have an ambition to develop a single-pass LV-ASR system that requires certain adaptations of the system structure.

2. A multi-level LV-ASR system

Let us first present the structure of the LV-ASR system that we are developing. Later, we focus of the finite-state lexicon model that has a crucial impact on the overall performance of the presented system.

In order to provide more flexibility, we divided our LV-ASR system structure into the three major levels that can be developed and optimized separately. The three levels are defined as follows:

- The grammar level, with context-free and/or stochastic grammar models;
- The lexicon level, with the finite-state pronunciation lexicon models;
- The phonetic level, with a set of context-dependent phonetic models.

From the algorithmic point of view the main level is the lexicon level. In our system, pronunciation lexicons are strictly

Copyright © 2009 ISCA

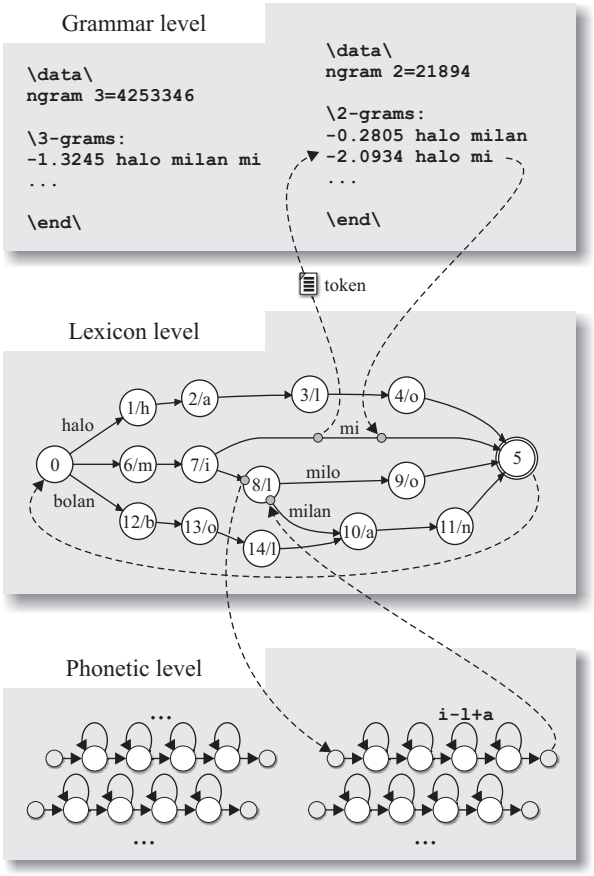


Figure 1: The three-level LV-ASR system structure and a symbolic illustration of a multi-level, token-passing, best-path search.

modelled as Moore FSTs. The reason for this is that we decided to follow a more conventional representation of HMMs as probabilistic Moore transducers (the output is considered to be produced by the HMM states and not by transitions).

The other two levels are not necessarily modelled as finite-state machines at all. We believe that this multi-level system structure enables more flexible research on new acoustic and grammar models that might be required due to the mentioned distinct characteristics of our language.

The system structure is illustrated in Fig 1. The best-path search algorithm is based on the conventional token-passing paradigm [4]. Each token has its own cost value and the costs increase while the tokens pass through the finite-state pronunciation lexicon network. Cost values would normally be assigned using some pattern-matching similarity or probability measures. The token-passing mechanism is triggered by the incoming sequence of speech-observation vectors. LV-ASR systems that are based on this paradigm require a pruning mechanism that simultaneously eliminates the less prospective tokens that are passing the network.

2.1. A token-passing algorithm

The multi-level structure of the system requires a slightly more complex token-passing algorithm than was originally proposed by Young et al [4]. For instance, when tokens pass into the acoustic or into the grammar level, they are tagged with the additional back-reference information that enables their proper return to the lexicon level. This mechanism is illustrated in Fig. 1. When a token passes an allophone state in the lexicon FSN, it enters the phonetic model in the acoustic level that is selected according to the token's phonetic context. All the tokens are tagged with their phonetic contexts, i.e., with the information about which state in the lexicon FSN they have already passed. When the token exits the phonetic model, it is returned back to proper position in the lexicon FSN using the back reference that was added to the token.

Tokens are also tagged with their word contexts. This information is used when tokens pass into the grammar level. This transition is triggered by their pass through the non-empty (word) transition in the lexicon FSN. The main task of the grammar level is to assign (as fast as possible) the cost of passing the particular token through its current word transition in the lexicon FSN, given its word context. The grammar level can thus be developed and optimized separately from the lexicon level. The same holds for the acoustic level the task of which is to assign (also as fast as possible) the costs of passing the particular token through the phonetic model considering the given speech-observation vectors.

3. Finite-state pronunciation lexicon model optimization

From the previous section it is clear how important the optimization of the finite-state pronunciation lexicon (FSPL) model is to the overall performance of the developing LV-ASR system. For historical reasons (the existing computer code, etc) and our ambition to develop a licence-free LV-ASR system, we decided to develop and implement our own optimisation algorithm for the FSPL. During the development, we did not refer to or use any of the two mentioned FST toolkits. We just tried to combine the conventional head and tail sharing strategies for building lexicon trees into a unified optimisation strategy.

We succeeded in developing an ad-hoc sequential optimization algorithm that enables the rapid sequential (word-by-word) building and minimisation of the FSPLs that are modelled as Moore FSTs. As Moore FSTs can easily be converted into Mealy ones and vice verse, we compared the obtained FSPLs to the ones that were obtained using the OpenFST toolkit. To our surprise our FSPLs were up to 60% smaller in size than the ones obtained with this toolkit. Before investigating the differences between the two optimisation strategies, let us first present our algorithm.

3.1. A sequential optimization algorithm

The proposed algorithm is based on the concept of embedding/integrating new words (word-by-word) into the existing FSPL using head- and tail-merge strategies. First, a left-to-right single-path FST with non-emitting initial and final states is created for a given word and its pronunciation. This FST contains one emitting state for each of the allophones of the word's pronunciation. The very first transition from the initial state is annotated as (input) word transition and all the others as non-word null transitions. A union between this FST and the existing FSPL is then created by merging both initial and final states of the two FSTs.

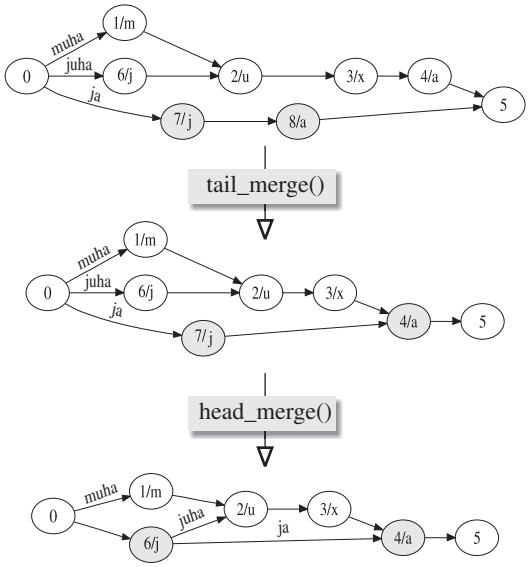


Figure 2: An illustration of the sequential minimization of the finite-state pronunciation lexicon model.

3.1.1. The tail-merge algorithm

The tail-merge 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 recursive tail-merge algorithm is defined as follows:

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

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

- The predecessor state is not the initial state of the FST and also not the newly added predecessor state;
- The predecessor state emits the same symbol as the newly added predecessor state;
- The predecessor state has only one transition from it (to the current state) and this transition is not a word transition.

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

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

Step 3 Quit and return the current state.

3.1.2. The head-merge algorithm

The head-merge algorithm is performed after the tail-merge one. This algorithm performs state merging from the opposite side of the unified FST. It also performs word-tag pushing from state transition to transition, if necessary. The recursive head-merge 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 successor state is not the last state that was merged by the tail-merge algorithm and also not the newly added successors state;
- The successor state emits the same symbol as the newly added successor state;
- The transition from the current state to the successor state is not a word transition, or if so, the successor state has only one predecessor state and also only one non-word transition to its single successor state.

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

- If the transition from the current state to the successor state or from the current state to the newly added successor state has a word tag, then this tag is pushed forward to the successor's transition to its single successor;
- The successor state and the newly added successor state are merged into one state;
- All the transitions to and from the merged state are rearranged, as necessary;
- The current state becomes the merged state;
- The algorithm recurses back to **Step 2**.

Step 3 Quit and return the current state.

Figure 2 illustrates the sequential minimization of a small example FSPL using the two presented algorithms. From this figure it can be seen how the word "*ja*" with the pronunciation [j a] is optimally embedded into the existing FSPL.

In general, the presented sequential minimization does not necessary minimize the FSPLs to their minimum possible sizes. It is obvious that the final result depends on how the word sequence that is processed by the algorithm is ordered. We found experimentally that it is better that the word sequence is not lexically ordered. Our experiments actually showed that the size of the FSPLs can be up to 20% smaller if the word sequences are randomly shuffled before using them for the construction of the FSPLs (for instance, using the Linux command *shuf*).

4. Experimental Results

In this paper, we report only the results of experiments with the presented sequential minimization algorithm. The obtained results were compared to the results obtained with the OpenFST toolkit. This toolkit is a successor of the commercial AT&T FSM library. For the comparison, the final state-emitting FSPLs were always converted into transition-emitting FSTs that were compatible with the OpenFST toolkit.

For the experiments we used two relatively large pronunciation dictionaries. The first one is the CMU Pronouncing Dictionary for North American English. The version we used contains 133,720 words and their transcriptions. The second one is the Multext-East Slovene Dictionary. The version we used contains 201,954 words that were transcribed using our TTS grapheme-to-phoneme converter.

For the OpenFST optimization tools, non-optimal FSTs (with word transitions from the initial state) were initially created directly from the word pronunciation lists. We experimented with both optimization strategies that are supported by the OpenFST toolkit. Better results were obtained with the strategy where the FSTs are temporarily encoded as finite-state acceptors (FSAs) and minimized as such. In this case, several optimization steps need to be performed. First, the FST is determinized and minimized as a FSA and then the labels are pushed to the final state. The FST is then again determinized and minimized as a FSA, and finally the labels are pushed to the initial state.

Optimization	States	Transitions
None	1,688,273	1,890,225
OpenFST	115,141	317,093
Sequential (ordered)	48,638	250,590
Sequential (shuffled)	29,063	231,015

Table 1: The sizes of the FSTs achieved from the Multext-East Slovene Lexicon that contains 201,954 transcribed words.

We soon discovered that for the same pronunciation lexicon our algorithm produces FSTs that have considerably fewer states and transitions than is the case with the OpenFST algorithms. The results are given in Table 2 and Table 1. An interesting observation that can be made from the two tables is that our algorithm constructed an FST with fewer states for the Slovenian dictionary than is the case with the English dictionary, even though the total number of Slovenian words is considerably higher than the total number of English words. The reason for this might be that the Slovenian language is much more inflected than English and that the Slovenian words were automatically transcribed using our TTS grapheme-to-phoneme converter.

Optimization	States	Transitions
None	720,095	853,813
OpenFST	63,793	197,511
Sequential (ordered)	38,096	171,814
Sequential (shuffled)	34,723	168,441

Table 2: The sizes of the FSTs achieved from the CMU Pronouncing Dictionary for North American English that contains 133720 transcribed words.

The presented algorithm was implemented in C++ using STL and also in Java. To our surprise the Java implementation is considerably faster than the C++ one. The Java implementation (`http://luks.fe.uni-lj.si/~simond/java/spider.zip`) generated a sequentially minimized FST from the 200k word example in less than 10 seconds on a common PC. 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 then compared to the original pronunciation lexicons.

We carefully studied the differences between our optimization algorithm and the OpenFST one and we believe that the main reasons for the achieved better optimization results lies in the sequential nature of the presented approach. A kind of proof for this claim is the fact that sequential optimization can also be simulated using the original OpenFST algorithms and that in this case the final results are also considerably better than the ones achieved by the global static optimization. The sequential optimization can be achieved by repeating a sequence of several OpenFST operations. First, we have to perform a union of the left-to-right FST, created from the newly added word, with the existing FST, and then perform the usual global FST optimization. This sequence of operations is then repeated for each of the newly added words. The main disadvantage of this approach is that it is very inefficient and time consuming.

Optimization	States	Transitions	Time
None	237,041	272,454	1s
OpenFST	22,655	58,055	7s
Sequential	12,520	47,927	9s
OpenFST (sequential)	11,441	46,846	397m50s

Table 3: The sizes of the FSTs and the computational times for their construction achieved from the 35k word lexicon.

We implemented such a simulated sequential optimisation algorithm in C++ using the OpenFST library version 1.1. As expected, the implementation was computationally very slow. For the comparison with other algorithms we were forced to use a smaller pronunciation lexicon with only 35k words. The obtained results are given in Table 3. From the above results it is clear that the sequential OpenFST minimization obtained slightly better results (up to 4%) than our algorithm; however, the price is a very long computation time (almost 7 hours). We attribute this smaller size to the fact that Meally FSTs can always have fewer or equal number of states than the equivalent Moore FSTs.

5. Conclusions

The LV-ASR system for the Slovenian langauge is still under development[1] and this paper briefly presents the multi-level system structure that our system is based on. The finite-state pronunciation-lexicon model is the crucial part of this system and the presented sequential minimization algorithm provides a simple and very efficient tool for its optimization. We believe that the presented sequential minimization algorithm for the state-emitting FSTs can also be rewritten to be used with transition-emitting FSTs that are supported by the OpenFST library.

6. References

[1] Allauzen, C., Riley, M., Schalkwyk, J., Skut, W. and Mohri M., "OpenFst: A General and Efficient Weighted Finite-State Transducer Library", in Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007), Lecture Notes in Computer Science, 4783:11–23, Springer, 2007.

[1] Dobrišek, S., Vesnicer, B., Žganec, G. J. and Mihelič, F., "Adaptation of Acoustic Feature Space Using Canonical Acoustic Model" In: Proceedings of the 9th International Multiconference Information Society, Ljubljana, Slovenia, 89–92, 2006.

[2] Jelinek, F., "Statistical Methods for Speech Recognition", The MIT Press, 1998.

[3] Mohri, M., Pereira, F. and Riley, M., "Weighted finite-state transducers in speech recognition", Computer Speech and Language, 16:69–88, 2002.

[4] Young, S. J., Russel, N. H. and Thornton, J. H. S., "Token passing: A simple conceptual model for connected speech recognition systems", Cambridge University Engineering Department Technical Report, F/INFENG/TR.38, 1989.