VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FINITE-STATE BASED RECOGNITION NETWORKS FOR FORWARD-BACKWARD SPEECH DECODING

ROZPOZNÁVACÍ SÍTĚ ZALOŽENÉ NA KONEČNÝCH STAVOVÝCH PŘEVODNÍCÍCH PRO DOPŘEDNÉ A ZPĚTNÉ DEKÓDOVÁNÍ V ROZPOZNÁVÁNÍ ŘEČI

DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE Dipl.-Ing. MIRKO HANNEMANN
AUTHOR

VEDOUCÍ PRÁCE Doc. Ing. LUKÁŠ BURGET, Ph.D.
SUPERVISOR

BRNO 2016

Abstract

Many tasks can be formulated in the mathematical framework of weighted finite state transducers (WFST). This is also the case for automatic speech recognition (ASR). Nowadays, ASR makes extensive use of composed probabilistic models – called decoding graphs or recognition networks. They are constructed from the individual components via WFST operations like composition. Each component is a probabilistic knowledge source that constrains the search for the best path through the composed graph – called decoding. The usage of a coherent framework guarantees, that the resulting automata will be optimal in a well-defined sense. WFSTs can be optimized with the help of determinization and minimization in a given semi-ring. The application of these algorithms results in the optimal structure for search and the optimal distribution of weights is achieved by applying a weight pushing algorithm. The goal of this thesis is to further develop the recipes and algorithms for the construction of optimal recognition networks. We introduce an alternative weight pushing algorithm, that is suitable for an important class of models – language model transducers, or more generally cyclic WFSTs and WFSTs with failure (back-off) transitions. We also present a recipe to construct recognition networks, which are suitable for decoding backwards in time, and which, at the same time, are guaranteed to give exactly the same probabilities as the forward recognition network. For that purpose, we develop an algorithm for exact reversal of back-off language models and their corresponding language model transducers. We apply these backward recognition networks in an optimization technique: In a static network decoder, we use it for a two-pass decoding setup (forward search and backward search). This approach is called tracked decoding and allows to incorporate the first pass decoding into the second pass decoding by tracking hypotheses from the first pass lattice. This technique results in significant speed-ups, since it allows to decode with a variable beam width, which is most of the time much smaller than the baseline beam. We also show that it is possible to apply the algorithms in a dynamic network decoder by using the incrementally refining recognition setup. This additionally leads to a partial parallelization of the decoding.

Keywords

Automatic speech recognition, LVCSR decoding, recognition networks, weighted finite state transducers, N-gram language models, weight pushing

Bibliographic citation

Mirko Hannemann: Finite-state based recognition networks for forward-backward speech decoding, PhD thesis, Brno, Brno University of Technology, Faculty of Information Technology, 2016

Abstrakt

Pomocí matematického formalismu váhovaných konečných stavových převodníků (weighted finite state transducers WFST) může být formulována řada úloh včetně automatického rozpoznávání řeči (automatic speech recognition ASR). Dnešní ASR systémy široce využívají složených pravděpodobnostních modelů nazývaných dekódovací grafy nebo rozpoznávací sítě. Ty jsou z jednotlivých komponent konstruovány pomocí WFST operací, např. kompozice. Každá komponenta je zde zdrojem znalostí a omezuje vyhledávání nejlepší cesty ve složeném grafu v operaci zvané dekódování. Využití koherentního teoretického rámce garantuje, že výsledná struktura bude optimální podle definovaného kritéria. WFST mohou být v rámci daného polookruhu (semi-ring) optimalizovány pomocí determinizace a minimalizace. Aplikací těchto algoritmů získáme optimální strukturu pro prohledávání, optimální distribuce vah je pak získána aplikací "weight pushing" algoritmu. Cílem této práce je zdokonalit postupy a algoritmy pro konstrukci optimálních rozpoznávacích sítí. Zavádíme alternativní weight pushing algoritmus, který je vhodný pro důležitou třídu modelů - převodníky jazykového modelu (language model transducers) a obecně pro všechny cyklické WFST a WFST se záložními (back-off) přechody. Představujeme také způsob konstrukce rozpoznávací sítě vhodné pro dekódování zpětně v čase, které prokazatelně produkuje ty samé pravděpodobnosti jako dopředná síť. K tomuto účelu jsme vyvinuli algoritmus pro exaktní reverzi back-off jazykových modelů a převodníků, které je reprezentují. Pomocí zpětných rozpoznávacích sítí optimalizujeme dekódování: ve statickém dekodéru je využíváme pro dvoustupňové dekódování (dopředné a zpětné vyhledávání). Tento přístup — "sledovací" dekódování (tracked decoding) — umožnuje zahrnout výsledky vyhledávání z prvního stupně do druhého stupně tak, že se sledují hypotézy obsažené v rozpoznávacím grafu (lattice) prvního stupně. Výsledkem je podstatné zrychlení dekódování, protože tato technika umožnuje prohledávat s variabilním prohledávacím paprskem (search beam) – ten je povětšinou mnohem užší než u základního přístupu. Ukazujeme rovněž, že uvedenou techniku je možné využít v dynamickém dekodéru tím, že postupně zjemňujeme rozpoznávání. To navíc vede i k částečné paralelizaci dekódování.

Klíčová slova

Automatické rozpoznávání řeči, dekodování řeči, rozpoznávací sítě, váhované konečné stavové automaty, jazykové modely

Bibliografická citace

Mirko Hannemann: Rozpoznávací sítě založené na konečných stavových převodnících pro dopředné a zpětné dekódování v rozpoznávání řeči, disertační práce, Brno, FIT VUT v Brně, 2016

Finite-state based recognition networks for forward-backward speech decoding

Declaration of Originality

I hereby declare that this thesis and the work reported herein was composed by and originated entirely from me. The work has been supervised by Doc. Ing. Lukáš Burget, Ph.D. and Doc. Dr. Ing. Jan Černocký. Information derived from the published and unpublished work of others has been acknowledged in the text and references are given in the list of sources. Some of the used recognition systems were set-up by the members of the BUT Speech@FIT research group or in cooperation with third parties (Microsoft Research, Kaldi team, Johns Hopkins University).

....
Mirko Hannemann
17.07.

Acknowledgements

I want to thank Lukáš Burget, who was my thesis supervisor, for his feedback and for the discussions, especially for his efforts to advance our general machine learning knowledge. I especially want to thank Daniel Povey, who was my first supervisor at Microsoft Research and who had the original idea of decoding forwards and backwards in time and to perform an iterative weight pushing. I also want to thank him and his family for their hospitality. Thanks go to Honza Černocký for reviewing this thesis and for never being tired in coming up with new projects and funding and to all colleagues from Speech@FIT for their help and good atmosphere at work. I also want to thank Jasha Droppo, who was my second supervisor at MSR, for the discussions on parallelizing the decoding algorithm and reversing the language model. I thank Geoffrey Zweig for hosting me at MSR and for useful discussions on decoding in general. Thanks go to Sanjeev Khudanpur for his input on the weight-pushing algorithm and on language modeling. I want to thank my wife Tereza for the patience she had with the inconveniences caused by the process of writing this thesis, and I want to thank God for providing me with the time and health necessary to achieve it. Some of the work described here was done when the author was summer intern at Microsoft Research, Redmond, WA. This work was partly supported by the Intelligence Advanced Research Projects Activity (IARPA) BABEL program, the European Union's Horizon 2020 project No. 645523 BISON and Czech Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project „IT4Innovations excellence in science – LQ1602".

Contents

Chapter 1

Introduction

The application I had in mind while writing this thesis is the decoding of the most probable sequence of words in large vocabulary automatic speech recognition (LVCSR). However, the approach presented here can also be used in other tasks, which can be formulated in the framework of weighted finite state acceptors (WFSA) or transducers (WFST), as e.g. finding the most probable sentence in statistical machine translation and finding the most probable pronunciation of a spelled word in grapheme-to-phoneme conversion.

When formulating automatic speech recognition (ASR) in the WFST framework [Allauzen et al.(2004)], [Mohri et al.(2008)], we use composed WFSTs, called decoding graphs or recognition networks. WFSTs are used to represent the language model (LM), the pronunciation lexicon and the Hidden Markov Models (HMM) in a unified framework. These components are integrated into a single WFST by the composition operation. Each component is a probabilistic knowledge source that constrains the search (called decoding) for the best path through the composed graph. The usage of a coherent framework guarantees, that the resulting automata will be optimal in a well-defined sense. WFST can be optimized by determinization and minimization in a given semi-ring. The application of these algorithms results in the optimal structure for search.

An optimized recognition network can contain up to millions of states, and the resulting search state space (trellis) is even several orders of magnitude larger. Given the complexity of the task, the search spaces cannot be explored exhaustively. It is necessary to use heuristic pruning techniques. In this case, we have to distinguish *search errors*, which are due to the incomplete exploration of the search space (e.g. through search beams and other pruning techniques), from *modeling errors*, which are due to insufficient (or bad) training data or due to inaccurate models (independence assumptions, choice of distribution, smoothing, ...). In general, the goal is to reduce the amount of search errors at given run-time requirements (decoding speed). This can be achieved by operations like weight pushing, which aim to distribute the weights along the path in a way that is optimal for pruned search.

The goal of this thesis is to further develop the recipes and algorithms for the construction of optimal recognition networks. We aim to find the optimal trade-off between improving search speed and reducing search errors. We introduce the idea of symmetrically decoding forwards and backwards in time. For some tasks, the pruned backward search can be more efficient than the forward search. Moreover, we show, that the search errors of forward and backward search are mutually independent. To concentrate on search errors rather than on modeling errors, we require both decoding passes to be symmetric – i.e. both models are equally powerful and are constructed to assign exactly the same probabilities to hypotheses. This guarantees that each difference in comparing the results of forward and backward decoding corresponds to a search error, which allows us to achieve significant speed-ups by decoding with a variable beam width.

1.1 Claims of the thesis

The main contributions of this thesis can be summarized in the following points:

- **Symmetric forward and backward decoding:** To speed-up the decoding, as opposed to multi-pass recognition techniques [Nguyen et al.(1993)], we use forward and backward recognition passes which are equally powerful. Equally powerful forward and backward decoding has been used before for the purpose of system combination [Li et al.(2009)] and confidence estimation [Jouvet and Fohr(2014)]. However, we require that the forward and backward recognition networks assign exactly the same probability scores, which allows us to detect search errors, to recombine partial paths and to incorporate the first pass into the second pass.

- **WFSTs resulting from back-off and interpolated language models:** We show, that the common practice to convert interpolated LMs into back-off LMs, when storing them in the ARPA file format, leads to problems in the construction of the recognition network in the log-probability semi-ring. We give details about the approximation and the correct handling of back-off arcs and explain “missing” N-grams.

- **Alternative weight pushing algorithm:** We give the theoretical justification and explain details of the alternative weight pushing algorithm, that is suitable for an important class of models – language model transducers, or more generally cyclic WFSTs and WFSTs with failure (back-off) transitions.

- **Construction of symmetric backward recognition networks:** We present a recipe to construct recognition networks, which are suitable for decoding backwards in time, fulfill the criteria of determinism and similar size, and at the same time, are guaranteed to give exactly the same probabilities as the forward recognition network.

- **Exact back-off language model reversal:** For the purpose of constructing backward recognition networks, we develop an algorithm for exact reversal of back-off language models and their corresponding language model transducers, which is valid for both types of approximations: using epsilon arcs or using failure arcs. We show the derivation of the formulas by a series of steps guaranteeing WFST equivalence, as well as the derivation from Bayes’ rule.

- **Tracked decoding and variable beam width:** We develop a two-pass decoding setup (forward search and backward search), that allows to incorporate the first pass decoding into the second pass decoding by tracking hypotheses from the first pass lattice. This technique allows to decode with a variable beam width, which is most of the time much smaller than the smallest single-pass beam and is only increased in areas, where forward and backward decoding disagree.

- **Speed-up and parallelization:** We have implemented the backward recognition networks for both static and dynamic network decoders and show experiments that demonstrate significant speed-ups in both cases. Applying the incrementally refining recognition setup of [Nolden et al.(2013)] additionally leads to a partial parallelization of the decoding.

1.2 Automatic speech recognition with weighted transducers

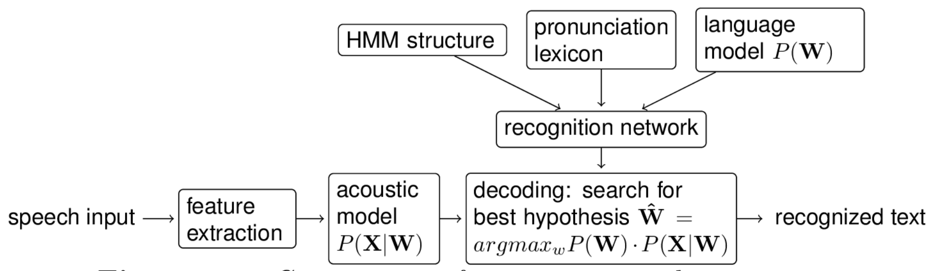
HMM structure
pronunciation lexicon
language model $P(\mathbf{W})$
recognition network
speech input
feature extraction
acoustic model $P(\mathbf{X}|\mathbf{W})$
decoding: search for best hypothesis $\hat{\mathbf{W}} = argmax_w P(\mathbf{W}) \cdot P(\mathbf{X}|\mathbf{W})$
recognized text

Figure 1.1: *Components of automatic speech recognition*

In figure 1.1, we summarize the basic structure of an automatic speech recognition (ASR) system. The task in automatic speech recognition (ASR) is to search for the word sequence $\hat{\mathbf{W}}$ with the maximum a-posteriori probability (MAP) given the acoustics, represented by a sequence of acoustic vectors $\mathbf{X} = x_1, \ldots, x_m$. The task is equivalent to:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} \frac{P(\mathbf{W})P(\mathbf{X}|\mathbf{W})}{P(\mathbf{X})} \approx \arg\max_{\mathbf{S}} \prod_{i=0}^{m-1} P(x_i|s_i)P(s_{i+1}|s_i). \qquad (1.1)$$

$P(\mathbf{X}|\mathbf{W})$ is called the acoustic model (computes the likelihood of the observation given the words) and $P(\mathbf{W})$ is the language model (LM) (the prior probability of the word sequence). Commonly, the acoustic model is a hidden Markov model (HMM). Each word is represented by a sequence of states, and the state transitions $P(s_{i+1}|s_i)$ model the temporal structure of speech, while the emission probabilities model the acoustic observations $P(x_i|s_i)$. Thus, in eq. 1.1 we approximate the search for the optimal word sequence by the search for the optimal state sequence $\mathbf{S}$. We search for the best path (state sequence) with the Viterbi algorithm [Viterbi(1967)] (trellis structure on left part of figure 1.2).

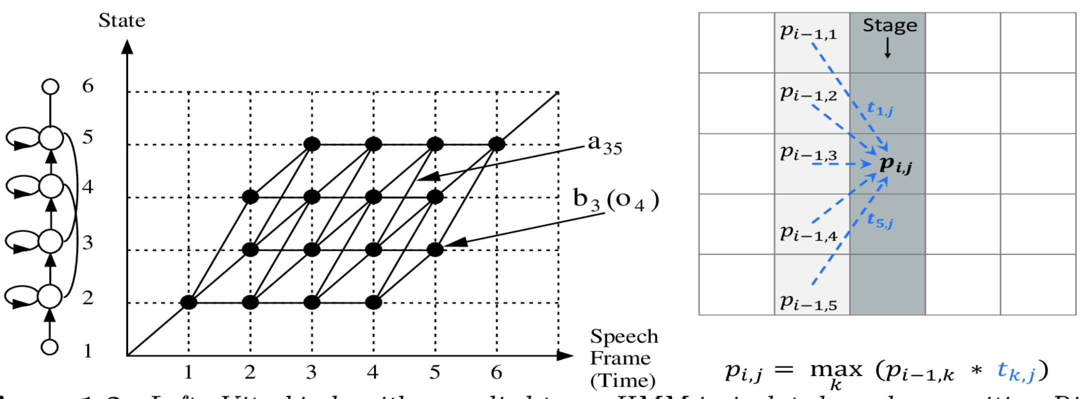

Figure 1.2: *Left: Viterbi algorithm applied to an HMM in isolated word recognition Right: Dependencies in time-synchronous Viterbi search: The global task of finding the best path is reduced to recursively solving the sub-problem of choosing the predecessor with the best partial path up to the current time step (dynamic programming [Bellman(1952)]). The score of a state depends only on all incoming arcs from the previous time step.*

ASR can be formulated in the framework of weighted finite state transducers (WFST) [Allauzen et al.(2004)], [Mohri et al.(2008)]. The HMM is represented as WFST (figure 1.3), called HMM structure transducer H or decoding graph.

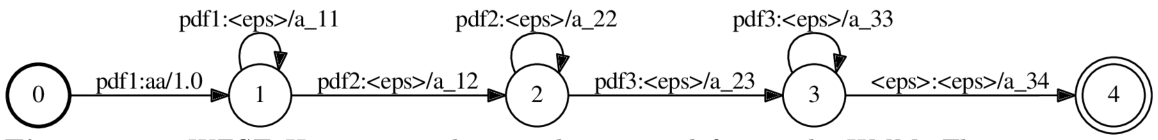

Figure 1.3: *WFST H corresponding to three-state left-to-right HMM. The arc notation is „input:output/weight", „< eps >" stands for ϵ (no symbol). We attach the emission probabilities to the incoming arcs of a state. Thus, the input labels correspond to identifiers of probability density functions (PDF-ids, often context-dependent HMM states). During decoding, PDF-ids are used to evaluate the corresponding HMM emission probabilities. The acoustic likelihood score is combined with the arc weight, corresponding to the HMM transition probability. The output label is the identity of the phoneme/word (aa). The final arc is non-emitting (<eps> input), it serves to interconnect (sub-word) HMMs.*

A recognition network (or decoding graph) (figure 1.1) is a composite HMM, connecting the individual phoneme HMMs (figure 1.3) according to the the pronunciation lexicon (mapping the words to phonemes) and the LM. The LM functions as a grammar, which constrains which words can follow each other. Figure 1.4 shows a simple recognition network for connected speech recognition with sub-word units.

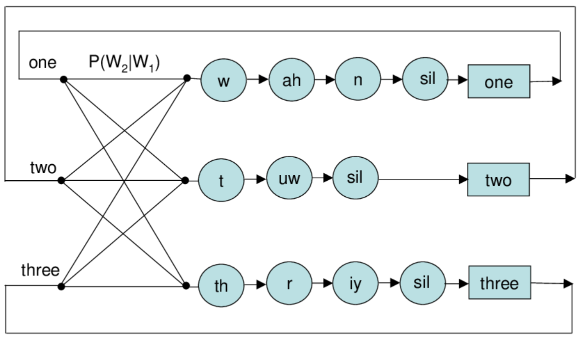

Figure 1.4: *Simple recognition network. Words are modeled by phonemes (sil: silence) and bi-gram probabilities are applied at word transitions. Each state (phoneme) is actually a three-state HMM.*

The standard recipe for the decoding graph construction is [Mohri et al.(2008)]:

$$HCLG = \min(\det(H \circ C \circ L \circ G)), \tag{1.2}$$

Here, H, C, L and G are the components, which are created separately and are integrated into a single WFST($HCLG$) with WFST composition (denoted as $\circ$). H, C, L and G represent the HMM structure, the phonetic context-dependency transducer, the lexicon transducer and the LM (grammar), respectively.

To decode an utterance, i.e. to find the most likely state sequence, we construct the search space (trellis) S [Povey et al.(2012)] by: $S \equiv U \circ HCLG$, where U is an acceptor (WFSA), whose arc weights (for each combination of (time, PDF-id)) correspond to the acoustic likelihoods. We search for the best path through S with the shortest path algorithm in the tropical (Viterbi) semi-ring. The best path is a linear WFST, whose output symbols represent the decoding result, i.e. the recognized sequence of words. The input symbols represent the sequence of PDF-ids for each time frame, from which we obtain the sequence of states. In practice, S is not searched exhaustively, but beam pruning is used. I.e. we are searching the best path through B, which contains a subset of the states and arcs of S, obtained by some heuristic pruning procedure.

Chapter 2

Forward and backward decoding

For tasks like LVCSR decoding, the search graph can contain up to millions of states. Thus, the search space cannot be explored exhaustively and it is necessary to use heuristic pruning techniques. The most widely used search technique in LVCSR is the Viterbi algorithm with beam search [Lowerre(1976)]. Beam search is a breadth-first style search, comparing partial paths of the same length (time-synchronously). At each time only those paths are kept and further expanded, whose path score is better than the current best score extended by a beam width. The beam width is a trade-off between speed and accuracy.

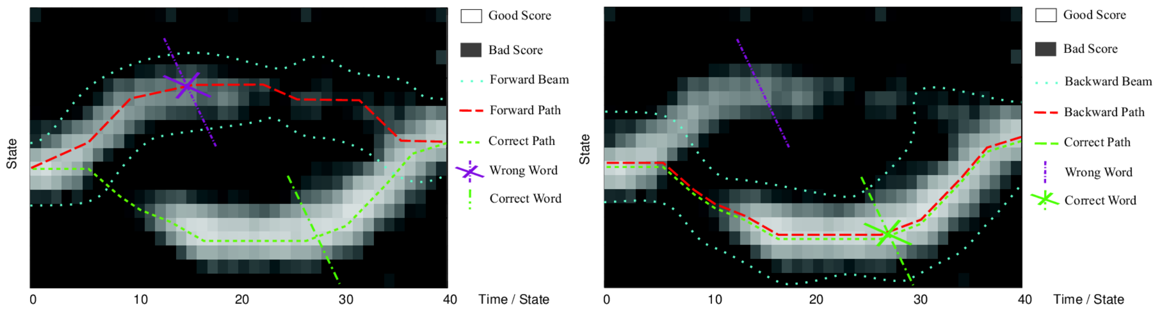

Figure 2.1: *Illustration of forward and backward search [Nolden et al.(2013)]. In the background, acoustic likelihoods for each state are shown as they evolve over time. Bright colors indicate higher probability. In the forward search (upper part), the low-score 'valley' around frame 7/8 causes the correct path (green) to fall out of the beam (dotted). The red path is chosen, but later (frames 20-30) it turns out to have poor scores. Even if it has better overall scores, the correct path can not be recovered, since it was already pruned. In the backward search (lower part), the situation is different - starting from the end, the lower path looks much more promising (frames 30-35) and the upper path falls out of the beam. The low likelihoods around frame 7/8 do not distract the recognizer this time, so the backward search does find the correct path. The illustrating explains, that to a certain extent, search errors of forward and backward search are independent. Of course, with a wide-enough beam, also the forward search would find the overall best path.*

In this thesis, we introduce the idea of symmetrically decoding forwards and backwards in time. For some tasks, the pruned backward search is more efficient than the forward search. [Tang and Cristo(2008)] showed, that for the recognition of street-city-state tuples (as used e.g. in the US), the error rate is lower when searching backwards in time. Figure 2.1 illustrates the potential of forward and backward search. A path that has low scores at the beginning is likely to be pruned by forward search, even if it has a high overall score towards the end. It has a chance not to be pruned by backwards search, because looking backwards this path has high scores at the beginning (which was the end in forward search). Forward search prunes based on the “history” and backward search prunes based on the “future”. Moreover, we showed in an experiment with the Kaldi recipe on the Wall Street Journal corpus (WSJ), that the search errors of forward and backward search are mutually independent.

We can detect and evaluate search errors by aligning the recognition outputs to a decoding with a very wide beam. We align the results of both forward and (reversed) backward decodings with the wide-beam-decoding. Table 2.1 shows an example of such an alignment. Table 2.2 confirms the intuition that forward and backward search errors are independent. With the help of the tracked forward-backward decoding proposed in this thesis, most of the search errors were eliminated.

```
f: BRIAN J. KILLING CHAIRMAN OF BELL - ATLANTA X.  INVESTMENT
     .    .    S       .     .   .   .    .     S      .
b: BRIAN J.  DAILY  CHAIRMAN OF BELL AND LAND  SIX INVESTMENT
     .    .    .       .     .   .   I    S     S      .
p: BRIAN J.  DAILY  CHAIRMAN OF BELL - ATLANTA ITS INVESTMENT
     .    .    .       .     .   .   .    .     .      .

w: BRIAN J.  DAILY  CHAIRMAN OF BELL - ATLANTA ITS INVESTMENT
r: BRIAN J.  KELLY  CHAIRMAN OF BELL - ATLANTIC'S  INVESTMENT
```

Table 2.1: *Analysis of search errors on the WSJ Nov’92 test set by aligning forward and backward search errors (with beam 11.0) against a decoding with a wide beam (29.0).*
Shown are the outputs of forward decoding (f), backwards decoding (b) and forward-backward ’ping-pong’ decoding (p), aligned to a decoding with very wide beam (w) and reference transcription (r). The search errors are indicated by ’I’ for insertion, ’S’ for substitution and ’-’ for deletion.

beam width	**forward errors**	**backward errors**	**co-occur**	**ping-pong**
11.0	144	230	32	**14**
13.0	84	108	14	**6**

Table 2.2: *Analysis of search errors on WSJ Nov’92 test set by aligning against a wide beam (29.0). The co-occurrence of an error (’co-occur’) means that for both, forward and backward pass, an error occurs at the same alignment position. This does not necessarily mean that both produced the same error. With two-pass ’pingpong’ decoding, all independent search errors were corrected (all those that are not co-occurring), and even a good portion of the co-occurring could be removed.*

Additionally to beam search, a strategy to deal with the complexity of the task is to use multiple decoding passes (e.g. [Murveit et al.(1993)]). Usually, inexpensive and approximate models are used in a first pass [Nguyen et al.(1993)] to generate an intermediate representation (e.g. lattices), which is then ‘re-scored’ using more complex models. In [Austin et al.(1991)], the idea of performing the second pass backwards in time was introduced. Since the forward scores are used as an estimator for the remaining part of the utterance, the second pass usually takes only a fraction of the time of the first pass, so that more complex algorithms or models can be used. A more recent re-discovery of the same idea is [Lee et al.(1998)] and [Lee and Kawahara(2009)], which use a word trellis as intermediate representation and stack decoding (A-star search) in the backward pass. Also [Cardinal et al.(2013)] use a uni-gram Viterbi backward pass, which is then used as a heuristic in A-star forward decoding with the full language model.

Opposed to these works, we focus on using forward and backward passes that are symmetric, i.e. using models that are equally powerful in both passes. The idea of symmetric passes was already used by [Li et al.(2009)] and [Abo-Gannemhy et al.(2010)] (see also [Tang and Cristo(2008)]). They combine the outputs of both passes based on LM scores or confidence measures (ROVER technique [Fiscus(1997)]). Also [Jouvet and Fohr(2013a)] and [Jouvet and Fohr(2013b)] use the framework of [Lee and Kawahara(2009)] to ROVER two symmetric passes, and they show that the combination of forward and backward passes is especially effective in improving the performance. On top of using equally powerful models, in this work, we require that the forward and backward recognition networks are constructed to assign exactly the same probabilities to hypotheses (paths, word sequences). The exact symmetry of both passes allows us to concentrate on search errors rather than on modeling errors. When comparing the recognition results of forward and backward decoding, each difference detects a search error.

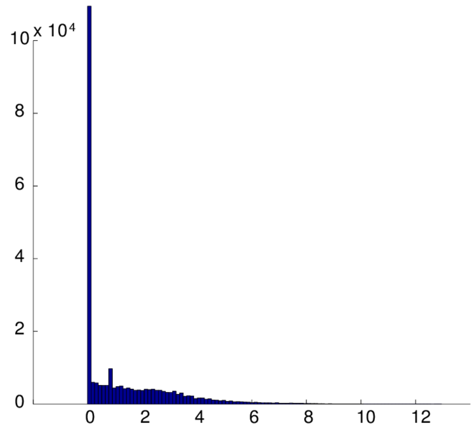

Figure 2.2: *Histogram of score differences: Shown are the scores of the current best partial path at each frame minus the partial score of the path that is going to be the final best path – not necessarily the correct one (decode beam 13.0, WSJ Nov'92 test set at WER 10.8%).*

In beam search, usually a constant beam width is applied to the whole test set. While analyzing the pruning behavior of the Kaldi decoder on the WSJ test set, we found, that for most of the time frames, a very narrow beam is sufficient to keep the final best path. Figure 2.2 analyzes the score differences at each frame, between the current best active token and the score of the token that will ultimately result in the best overall path[1]. Most of the time this difference is much smaller than the typical beam width (between 10 and 15). This suggests that it would be beneficial to to decode with a variable beam width. For that, we need to identify problematic areas (frames) that lead to search errors. We can achieve that by comparing the hypotheses from forward and backward search. We use a small baseline beam and only increase it in places, where the forward and backward searches disagree. Thus, the idea of our work [Hannemann et al.(2013)] is to speed up the decoding by using the (dis)agreement of the two symmetric decoding passes - decoding forwards and backwards in time. We replace a single pass with a wide beam with two passes with a small beam.

[1]To determine this, we run a full decoding, back-track the best path and compute its score at each frame.

2.1 Incremental forward and backward search

One possible realization of the variable beam width decoding is to run the forward and backward passes in parallel. Inspired by [Hannemann et al.(2013)], the authors of [Nolden et al.(2013)] implemented an incremental high-level decoding algorithm, which iteratively refines the decoding (by increasing the beam width) in places, where both passes disagree. As a consequence, the system uses a variable beam width and is dynamically focusing only on the parts that are difficult. The pruning beam is tuned for single words in an unsupervised way. As opposed to [Hannemann et al.(2013)], where the results of the first pass are integrated into the second decoding pass, both passes, forward and backward search, are run independently and symmetrically.

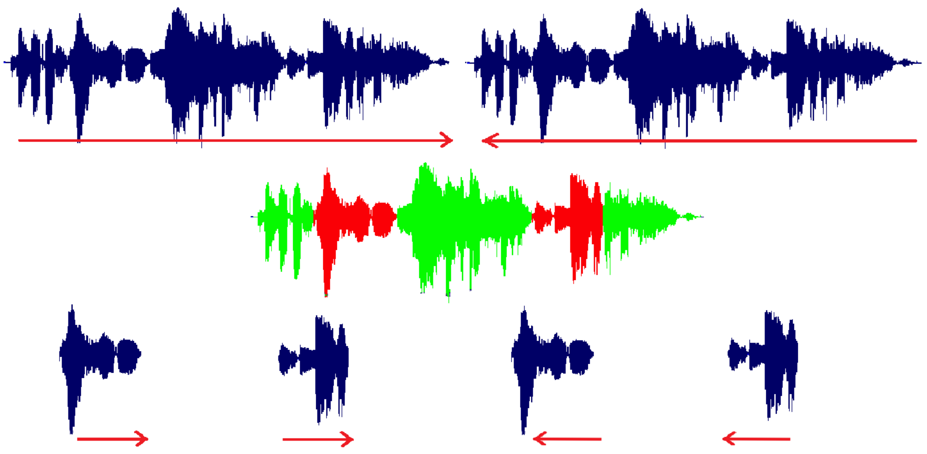

Figure 2.3: *Parallel implementation of incremental forward backward decoding [Nolden et al.(2013)]. First (upper part), two cores run a quick initial forward and backward decoding of the whole utterance with a narrow beam in parallel, then (center) the results are aligned and mis-matching regions ('islands') are identified (indicated in red). If there are no mis-matching regions, the decoding is done. Else (lower part), in a second pass, the identified mis-matching segments are decoded in parallel with a wider beam. In this example, there are two 'islands', both of them are decoded forwards and backwards, which means four cores can be used in parallel. The results of the decoded segments are integrated into the results of decoding the whole utterance, and this process is iterated until the results for the whole utterance match. This way, the beam for each word is tuned to the minimum necessary beam.*

Figure 2.3 explains the parallel implementation of the incremental decoding [Nolden et al.(2013)]. In the alignment of decoding results, words are considered matching, if they have the same word identity as well as a matching time boundary. All non-matching words are grouped into continuous segments which are extended by N-1 matching words (according to the LM) to the left and to the right. For the incremental decoding of partial utterances, the left and right LM contexts of the segment need to be correctly initialized in the decoding. We also need to remember the left and right acoustic cross-word contexts, which can be achieved by remembering the states of the recognition network at the segment boundaries in the first pass – these can then serve as initial and final states for the second pass decoding.

[Maleki et al.(2014)] presented a parallelization of the decoding of an utterance into chunks. They showed that it is possible to split an utterance at places, where the rank of all-pairs-shortest-path matrix will converge a to singular matrix. In other words, this happens at places, where just one token will survive. An open question is whether it is possible to automatically detect frames in advance, where this will happen, to find the optimal segmentation of a given utterance into chunks. At the points with low rank, i.e. with few remaining active states, a small beam should be sufficient to decode them. In other words, at those points, we would expect the decoding results of the forward and backward search to agree, even if both run with a small beam. Therefore, a good segmentation for the parallelization of the decoding is to split the utterance at points, where forward and backward search agree. Thus, the incremental decoding is very similar to chunk based decoding. Since it is a high-level technique, it can be applied on top of other coarse-grained and fine-grained parallelization techniques.

We implemented the incremental forward-backward decoding in the Microsoft Argon decoder (documented in [Agarwal et al.(2014)], Version 2016-02-17) and tested it on the Eval2000 database with a recognizer trained on the Switchboard database. We observed, that the overall speed-up of the technique is be determined by the setup of the first pass (forward and backward) decoding. We choose an operation point, that is several times faster than a well-tuned baseline (tuned for WER/RTF, around RTF 0.3-0.5), but still in the area where the results of forward and backward decoding are partially matching. Using such a setting, we observe that after the first parallel forward/backward pass, in average approximately 50% of the complete utterances agree and thus the decoding can be finished.

For the files that are partially mis-matching, we find in average around 1.5 mis-matching segments ('islands') per file. That means we can achieve a speed-up of 1.5 on these utterances, and only a part of the utterance actually needs to be decoded again. Therefore, the total amount of time spent in the second pass will be much smaller than in the first pass, even if it runs at a higher RTF (increased beam). Similarly, the amount of time spent in further iterations will quickly decrease.

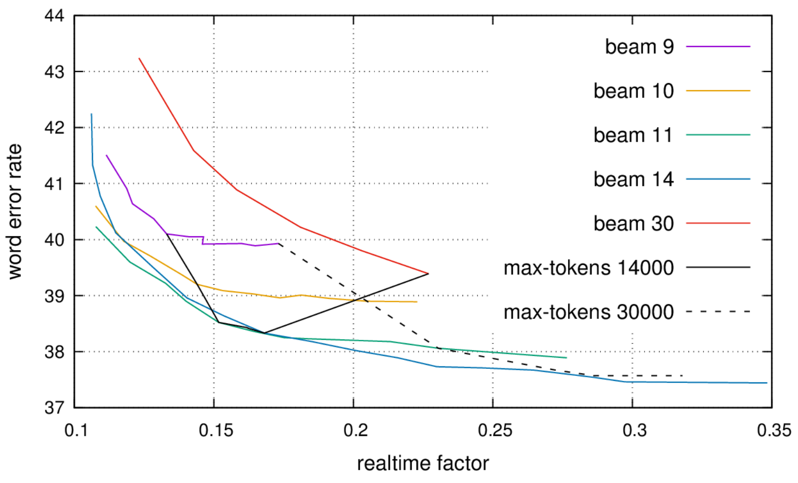

Figure 2.4: *Finding the optimal operating point on the real-time-factor and word error rate curve, while tuning the maximum number of active tokens (max-tokens) and beam width (beam). The settings of beam width and max-tokens are grouped by lines that leave one of the parameters fixed while varying the other. All curves 'beam' leave the beam width constant while running experiments with different values for max-tokens. The curves 'max-tokens' (black) measure different beam widths for a fixed number of max-tokens.*

Figure 2.4 explores the relation between WER and RTF on many different operating points, defined by a setting of the main tuning parameters, which are the log beam width and the maximum number of active tokens 'max-tokens' for the histogram pruning. We observe that both parameters depend on each other in a non-trivial way. Leaving either of them fixed while varying the other leads to sub-optimal solutions (max-tokens in black lines, beam 9-30 colored lines). Therefore, we have to test all possible combinations of parameters and then determine for each RTF the optimal WER and the corresponding tuning parameters. The resulting curve is sometimes called Pareto-optimal. What we observe is that along the Pareto-optimal curve, we have to proportionally increase both the beam width and the max-tokens. Perhaps surprising is the fact that if we over-shoot the beam (figure 2.4, red line, beam 30), we actually increase the WER significantly. We can explain this effect with the max-tokens beam resulting from histogram pruning, which turns out to be narrower for high beams. This is most probably a particularity of the Argon decoder, caused by the adaptive beam controlling [van Hamme and van Aelten(1996)].

2.2 Tracked decoding

Another realization of the variable beam width is the tracked decoding [Hannemann et al.(2013)] presented in this thesis. After using independent and parallel forward/backward decoding passes in the last section, in this section we run forward and backward decoding sequentially. Our approach towards decoding is to do a first pass (which happens to be a forward pass) with a narrow beam, and then to do a second pass in the opposite direction, also with a narrow beam, but using knowledge obtained during the first pass. In this approach, the beam width can be adjusted for every frame, so that a more careful search is only carried out in areas where the two passes disagree. The speed-up is achieved by using a narrow beam during the forward pass, and in the backward pass in places where no disagreement is detected.

The first pass outputs a lattice with state-level alignments [Povey et al.(2012)]. Note that this lattice does not contain all partial paths explored in the first pass, but only those word-sequences that are within a specified beam of the best word-sequence (lattice beam). We want to treat the paths in this lattice in a special way in the second decoding. That is,

1. We want to avoid pruning out paths that appeared in the first-pass lattice.
2. On frames where we would otherwise have pruned out those paths, we want to increase the pruning beam.

During decoding, we need to be able to identify which active tokens in our second-pass decoder correspond to paths in the first-pass lattice. These are called tracked tokens and we track tokens with what we call an *arc-lattice*. It is a special kind of lattice that allows us to identify arcs in $\mathrm{HCLG}_{\mathrm{2nd}}$ that were present in the first-pass lattice. This means there is a path in the lattice, that went through the corresponding state in $\mathrm{HCLG}_{\mathrm{1st}}$ at the given time. The arc-lattice is an *acceptor* FST, i.e. it has only one symbol on each arc. These symbols correspond to arcs in $\mathrm{HCLG}_{\mathrm{2nd}}$. Algorithm 1 summarizes the arc-lattice generation.

The second-pass decoder, which we will refer to as our *tracking decoder*, is a lattice-generating decoder that takes an extra input, namely the arc-lattices for each utterance. Let a *token* be a record of a particular state in HCLG that is active on a particular frame. Our tracking decoder gives tokens an extra, Boolean property that identifies whether they are *tracked* or not. A tracked token is one that corresponds to a state in the arc-lattice,

Algorithm 1 *Generation of arc-lattices (graph-state-lattices):*

```
1. Map $HCLG_{2nd}$ to PDF-to-Arc transducer $HCLG_{arc}$:
    (a) $HCLG_{2nd}$ : transduces PDF-ids into words
    (b) Encode $HCLG_{2nd}$ (node-id, arc-id) into output symbols.
    (c) Map input to be self-loop order independent.
2. Map first-pass lattice $LAT_{1st}$ to $LAT_{rev}$:
    (a) Map input (self-loops), project on input, remove weights.
    (b) Time reverse lattice and remove epsilons.
3. Compose: $LAT_{arc} = LAT_{rev} \circ HCLG_{arc}$:
    (a) Obtains sequences of $HCLG_{2nd}$ arcs for PDF sequence in lattice.
    (b) $det(LAT_{arc})$: Lattice-determinize (on PDF-ids) in special semi-ring
    $\rightarrow$ single $HCLG_{2nd}$ path left for each sequence of PDFs.
    (c) Project to $HCLG_{2nd}$ (node, arc) symbols, determinize again.
$\rightarrow$ The output is an acceptor lattice for $HCLG_{2nd}$ graph arcs.
```

i.e. it was reached by a sequence of $\mathrm{HCLG}_{\mathrm{2nd}}$-arcs in the arc-lattice that correspond to a path in the first pass lattice. Tracked tokens are never pruned, regardless of the beam width. If a token other than those that came through the tracked path, with better score, reaches the same state at the same time, the tokens recombine, i.e. it replaces the tracked token, but inherits the status of being tracked.

Tracked tokens are used to determine the variable pruning beam for each frame. In places where disagreement is detected, the beam is increased to include all of them. Otherwise in the second pass, the same narrow beam is used that was used in the first pass. The decoder has three configurable values that specify how it sets the frame-specific beam: the *beam*, the *max-beam* and the *extra-beam.* On a particular frame, let the score difference between the highest-score token and the lowest-score tracked token be D. Then the beam width on that frame is given by:

$$\max(\text{beam}, \min(\text{max-beam}, D + \text{extra-beam})).$$

In initial experiments, we found, that the technique is not sensitive to the setting of *extra-beam*, so we typically set it to zero. The *max-beam* is usually large (e.g. 40 for this task in Kaldi). We try various values of the *beam* for our experiments here.

Note that even if we keep the beam equal to the single pass *beam* during the tracked second pass, our method is doing more than simply choosing the best path from two (forward and backward) passes, because it is possible to "recombine" partial paths from the first-pass and second-pass search (effectively combining the forward and backward lattices). Some parts of the utterance might be advantageous for backwards decoding, other parts might have the opposite characteristic. If partial paths of tracked tokens and second-pass-only tokens meet in the same state, they can recombine and thus we would continue decoding the rest of the utterance with the maximum of the two partial scores (likelihoods). Therefore, the combined path can have a better score than either two single paths. On top of combining the lattices, the variable beam leads to the generation of extra tokens in areas where both passes disagree, which gives an additional speed-up.

2.2.1 Experimental results

We tested the forward-backward tracked decoding on the WSJ November'92 open-vocabulary test set (333 utterances) using a standard tri-phone HMM+GMM system (Kaldi recipe 'tri2a' [Povey et al.(2011)], trained on the 'si84' portion of WSJ). The experiments were conducted with the extended 146k vocabulary using the pruned tri-gram language model 'bd_tgpr' that was trained on all WSJ training texts. We measured the total elapsed time for the two-pass forward and backward (tracked) decoding and relate it to the word error rate (WER). The real-time factor was measured on a single core of an Intel(R) CPU i5-2500 (3.3GHz, 8GB RAM).

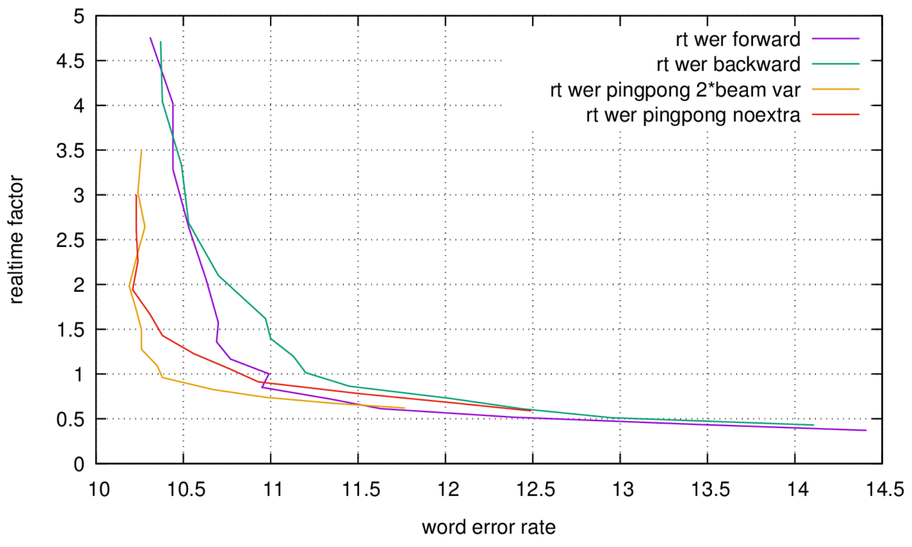

Figure 2.5: *Performance of tracked decoding: Shown are curves for word error rate vs. real-time factor on the WSJ Nov'92 test set. For single-pass decodings, the beam varies between 10-18, for the two-pass ('pingpong') decoding the beam varies between 7-13. We used extrabeam* $= 0$ *and found maxbeam* $= 2 \cdot$ *beam as a good compromise between speed and accuracy. The lattice-beam is* 4.0*, but for beam* < 10.0 *we decrease it step-wise to* 0.5*. We compare the variable-beam decoding ('2beam var', orange) to a decoding without generating extra tokens in the variable beam ('noextra', red) by setting maxbeam = beam, which shows the additional benefit of the variable beam over just combining lattices of forward and backward passes.*

The results in figure 2.5 show, that for the lowest word error rates (WER< 10.5), the two-pass tracked decoding runs about 2-3 times faster than the individual forward/backward passes at the same WER. This corresponds to the "more accurate" operating points of decoding where search errors are small. However, in this setup, the speed-ups are diminishing for operating points faster than ≈ 0.6 real-time using our method. The issue seems to be that if the beams are too narrow, the two decoding passes disagree substantially and too much effort is spent in decoding with a widened beam in areas that disagree. Also, [Nolden et al.(2013)] points out that a too narrow beam could lead to a degenerated search, where both passes produce the same errors (e.g. focussing on silence and noise models, which are symmetric). The WER curve in figure 2.5 is not always smooth, which points to the fact that fixing a search error does not necessarily mean fixing a word error.

To get an insight on the optimal size of the forward/backward beam, we profiled the tracked decoding in figure 2.6. We observe, that the time spent in the two individual

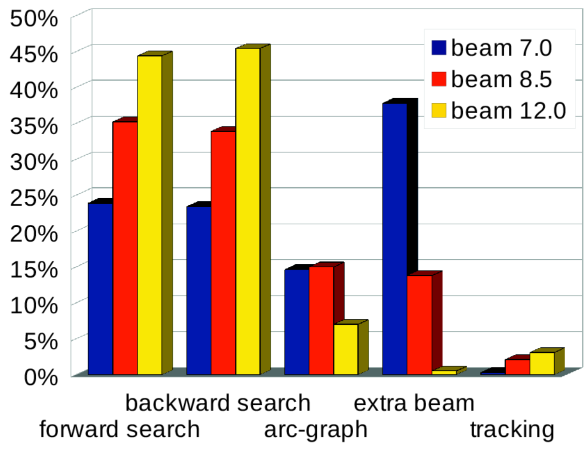

Figure 2.6: *Profiling the tracked two-pass decoding. Shown is the percentage of time spent in different parts of the algorithm at three operating points (beam 8.5 as optimal, others as not optimal). The first pass is the lattice-generating 'forward search' (which is also our single-pass baseline) and the second pass can be seen as consisting of a) normal backward decoding (column 'backward search'), b) generating the arc-lattice ('arc-graph'), c) additionally tracking tokens from the first pass ('tracking') and d) generating extra tokens within the increased variable beam ('extra beam'). The acoustic scores were not cached between the two passes. The contributions of 'arc-graph' and 'tracking' (together $< 20\%$) could be possibly optimized by a better implementation, but the two individual passes constitute a lower bound (around 70% of the time is spent there).*

decoding passes (without tracking / extra tokens) is the dominant factor. Thus, the baseline beam determines the possible speed-up. It should be small enough to decode at least two times faster than the original single pass, and it should be wide enough to allow for a reasonable comparison of the forward and backward search results, i.e. either of the two passes should obtain a solution, that is at least partly correct.

If we reduce the beam too much, we observe (figure 2.5 for error rates $> 11.5\%$) that the two-pass decoding is no longer better than the single-pass decoding. From figure 2.6 we see, that for low beam widths, most of the time is consumed in the generation of extra tokens, which effectively means decoding with a higher beam. Below a certain beam width (11% in figure 2.5) the error rates in the single passes grow rapidly with only little RTF to gain. This means that the divergence between the best paths from forward and backward decoding is too big, so that the algorithm has to increase the variable beam a lot to track the first pass tokens and an excessive amount of extra tokens are generated. With an optimal setting of the beam, we can reach a significant WER decrease by just generating a small amount of extra tokens in the variable beam ('extra beam' in figure 2.6, optimal around beam 8.5). This point corresponds to the turning point in figure 2.5 (around RTF 1.0) - it is the 'sweet spot'. Above that, though little time needs to be spent for tracking and for generating extra tokens, too much time is spent in the individual forward/backward decodings, and the overall RTF increases rapidly.

The proposed decoder has several parameters to tune: forward beam, backward beam, lattice-beam, extra-beam and max-beam. Since the WER-RTF curves for single-pass forward and backward decodings are similar, we typically set the forward beam and backward beam to the same value. Concerning the other parameters, there seem to be two strategies one could pursue: either track many tokens and try to combine good forward and backward paths, while limiting the generation of extra tokens, or just track few tokens and generate many extra tokens up to the variable beam difference. To analyze this, we can compare a decoding using the variable beam to a decoding without generating extra tokens. We can achieve this by limiting the beam to $maxbeam = beam$ and thus effectively disabling the variable beam. Since tokens 'tracked' by the first-pass lattice are kept anyway, this effectively corresponds to combining the lattices of the forward and backward pass. Figure 2.5 ('2beam' vs. 'noextra') shows that creating extra tokens within the variable beam gives a substantial improvement on top of that. Especially for the operating points with low WERs, the extra tokens are important.

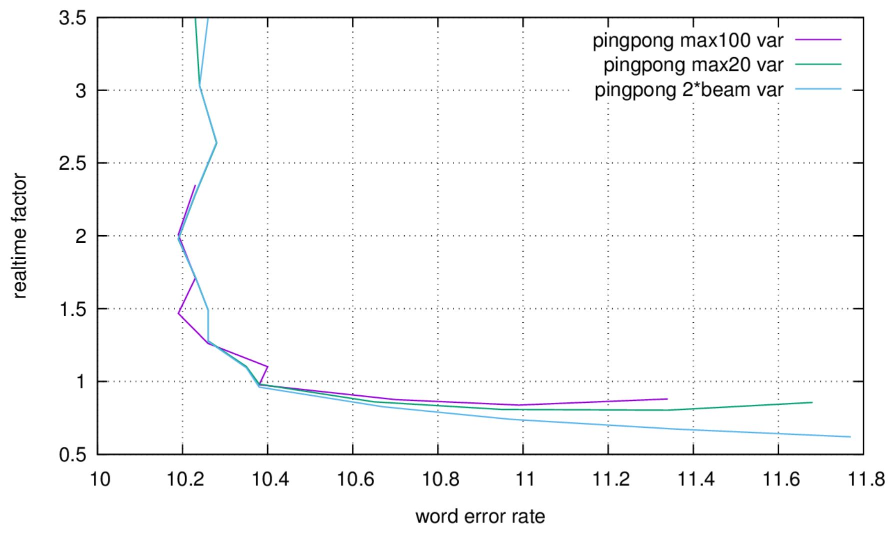

Figure 2.7: *Analyzing the effect of different max-beam settings on WSJ Nov'92 test set using the big bi-gram LM with 147k vocabulary. As already explored, we set the parameters to* $extrabeam = 0.0$ *and linearly increased the lattice-beam from 0.5 to a maximum of 4.0. Now, we compare three strategies of setting max-beam: a) using a fixed max-beam of 100.0 b) using a fixed max-beam of 20.0 c) changing the max-beam linearly with the beam:* $maxbeam = 2 \cdot beam$*. We also tried* $maxbeam = beam$*, which had slightly worse performance for* $WER > 11.0$.

In an experiment to investigate the effect of the lattice-beam, we observed, that the generation of the lattice has mainly the effect of adding an offset to the RTF/WER curve. Thus, we want to make it as small as possible. However, for the most accurate operating points with low WER, we want to have a wider lattice that contains the best path. It seems to be a good strategy to increase the lattice-beam linearly with the beam. We set an upper bound of 4.0, which is enough to get good results in re-scoring the lattice. Finally, after tuning all other parameters, we investigate different settings of the max-beam parameter. It is an upper limit to the variable beam, which becomes effective in the areas of higher WER. Figure 2.7 suggests, that the exact setting of the parameter max-beam doesn't influence the potential speed-up of the technique (the 'sweet-spot'), but mainly influences the shape of the curve from the 'sweet spot' towards the higher WER. Using no limit for the beam even for huge divergences between forward and backward pass seems wasteful. Therefore, it seems to be reasonable (figure 2.7) to increase the max-beam slowly with increasing beam. Once a reasonable beam has been reached, the divergence between forward and backward passes gets smaller, and the max-beam is no longer needed.

Chapter 3

Exact reversal of the recognition network

For the decoding techniques presented in the last chapter, i.e. to search in two symmetric forward and backward decoding passes, we need two corresponding decoding graphs - $HCLG_{fwd}$ and $HCLG_{bwd}$. Both models should be equally powerful, i.e. have roughly the same accuracy and run-time requirements, and have similar structure, size and level of determinism to have optimal pruning behavior. To compare the probabilities (or scores) of the outputs from the forward and backward passes (e.g. to estimate the optimal beam width). we need models $HCLG_{fwd}$ and $HCLG_{bwd}$, that ideally produce the same overall score for the same hypothesis in both the forward and the backward passes. Due to the pruned search, both passes can result in different search errors (due to the different branching forwards and backwards). However, both models should not make different modeling errors, hence they should assign the same scores to the same hypotheses. We also want to be able to compare the scores of partial results (paths). Therefore, also the model structure (the distribution of weights along paths) should be similar in the forward and backward passes. Given a forward graph $HCLG_{fwd}$, the task is to obtain a backward graph $HCLG_{bwd}$ that will assign exactly the same overall score to the same utterance and will fulfill all the above stated requirements. Because our method treats disagreement between the best paths found by the two passes as a search error, we want the backward decoding graph to be equivalent to the reverse of the forward one.

The trivial solution to apply WFST reversal to $HCLG_{fwd}$ is not sufficient, since the resulting graph would not have a similar level of determinism and distribution of weights as the forward graph, i.e. it would show sub-optimal behavior when used in a pruned search. To make the resulting WFST determinizable, we would have to introduce disambiguation symbols [Mohri et al.(2008)] at different places than in the forward graph. As we will see shortly, especially the (reversed) LM component would introduce a great degree of local ambiguity. Instead, the solution is to separately construct the time-reversed versions of H, C, L and G (introduced in chapter 1) and then to build a composed model $HCLG_{bwd}$ in an analogous way as the forward graph was constructed. Another point is that the stochasticity of outgoing arcs will not be satisfied when reversing the model, i.e. the optimal weight distribution for backward search is different from the one used in forward search. Therefore, we have to apply weight pushing to the reversed components.

Our approach to the construction of backward recognition networks is not limited to static network decoders. Since all components are reversed individually, no change is nec-

```
A         ax #1
ABERDEEN  n iy d er b ae
ABOARD    dd r ao b ax
ADD       dd ae #1
```

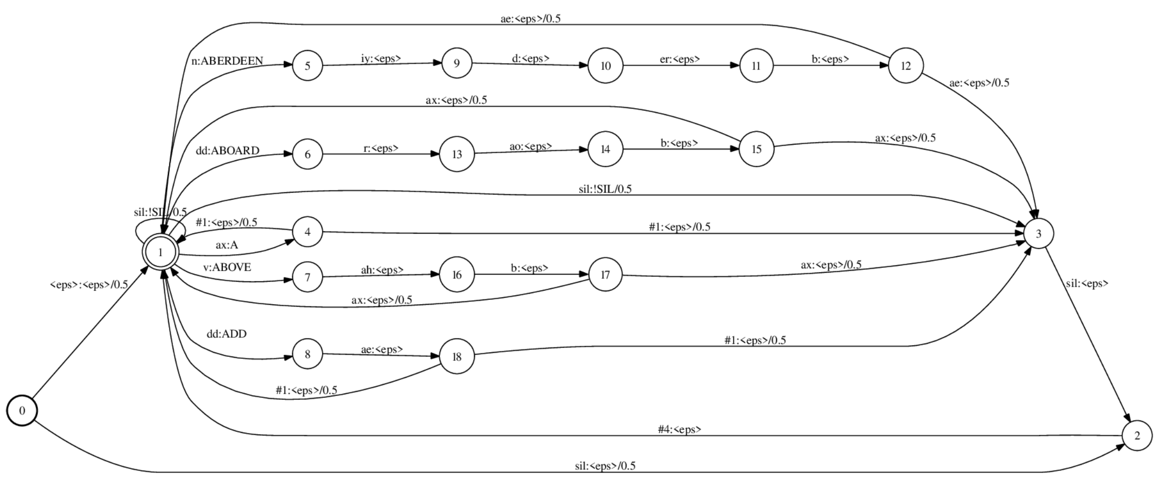

Figure 3.1: *Reversing lexicon transducer L. The phone sequences are reversed (upper part), and new disambiguation symbols (#1) are inserted after that. Then, the lexicon transducer is built in the same way as in the forward network (lower part).*

essary when dynamically composing the components in a dynamic network decoder. The time-reversed versions of H, C, L and G are again not simply the WFST reverses of the forward ones, but must be separately constructed. Depending on the task, the reversal of each component is of different complexity.

The construction of the reversed pronunciation lexicon transducer L_{bwd} (phones to words) is simple: the individual phone sequences (pronunciations) are reversed, and the disambiguation symbols are introduced after that. The disambiguation symbols now distinguish similar word endings, while in the forward case they distinguish word beginnings. Figure 3.1 shows a reversed toy lexicon and the resulting transducer.

The context-dependency transducer C_{bwd} is constructed in the usual way, and looks identical to C_{fwd}. After the composition of $L_{\text{fwd}} \circ G_{\text{bwd}}$, the phonetic context window (which are the input symbols for C) is reversed in time (a-b-c to c-b-a). Therefore, to look-up the corresponding models (PDFs) in the phonetic decision tree, we have to reverse the phonetic context. Then, we look-up using the phoneme context window and the HMM state. The HMM structure transducer H_{bwd}, is constructed in the same way as H_{fwd}, except for the reversed phonetic context. The individual (three-state) HMMs for each phone are constructed separately and the relevant PDFs are looked-up from the decision tree. Then, the phone HMMs must be reversed and "weight-pushed" in the log-semi-ring (including epsilon removal) to make the time-reversed probabilities sum to one. After that, the reversed phone HMMs are composed to the H_a transducer. Specific to the Kaldi toolkit is, that the H transducer is first created without self-loops (H_a), which are then added at a later stage. Due to the reversal of individual HMMs, the ordering of the self-loops and forward transitions changes, which doesn't matter for decoding, needs to be considered when mapping resulting alignments at transition level.

3.1 N-gram back-off LM and weight pushing

N-gram LMs approximate the conditional probability of a word given its history $P(w_i|\ldots)$, by reducing the history to the previous $N-1$ words:

$$P(w_1 \ldots w_m) \approx \prod_{i=1}^{m} P(w_i|w_{i-N+1} \ldots w_{i-1}) \tag{3.1}$$

N-gram LMs can be expressed as weighted finite state acceptors (WFSA) - each LM history corresponds to one state of the automaton ($h_i = w_{i-N+1} \ldots w_{i-1}$). However, the number of possible states of a model of order N with a vocabulary size V is V^{N-1} and the number of possible arcs (and N-grams) is V^N, which becomes clearly intractable for higher orders of N (typical vocabulary sizes go into the hundreds of thousands). As a consequence, LMs only store the probabilities of those N-grams that occur sufficiently often in the training texts. Statistical smoothing techniques are applied to the distribution of counts. On top of that, typically models of different N-gram order are combined. Either different orders of history are interpolated, or the higher order model performs backing-off for unseen N-grams by leaving out the first word in the history and looking-up the shorter history in the lower order model ($N-1$). This process is repeated recursively until the word in context is found. Back-off LMs were introduced by S. Katz [Katz(1987)]:

$$P_{Katz}(w_i|w_{i-N+1} \ldots w_{i-1}) =$$
$$= \begin{cases} P'(w_i|h_i) = d_{w_{i-N+1} \ldots w_i} \cdot \dfrac{C(w_{i-N+1} \ldots w_{i-1}w_i)}{C(w_{i-N+1} \ldots w_{i-1})} & \text{if } C(w_{i-N+1} \ldots w_i) > k \\ \alpha_{w_{i-N+1} \ldots w_{i-1}} \cdot P_{backoff}(w_i|w_{i-N+2} \ldots w_{i-1}) & \text{otherwise} \end{cases} \tag{3.2}$$

Here, d is the amount of discounting applied, C is the occurrence count of the given N-gram in the training corpus and k is the minimum number of occurrences. $\alpha_{w_{i-n+1} \ldots w_{i-1}}$ is the so called back-off weight, which is dependent on the current history. It usually corresponds to the sum of probability mass that was discounted from all N-grams sharing the same history and is now available to be re-distributed by the lower order distribution $P_{backoff}$, that can be recursively defined in exactly the same way as P_{Katz}.

As seen in Figure 3.2, back-off LMs can be represented as WFSA, but usually an approximate structure with back-off arcs is used [Allauzen et al.(2003)], [Mohri et al.(2008)]. Back-off arcs with the symbol ϵ introduce non-determinism. When the weights are taken as probabilities from a back-off model, this non-determinism can cause problems. In this case, the resulting WFSA is no longer stochastic – i.e. the probabilities of outgoing arcs do not sum to one. The weight of some cycles is greater than one and results in an infinite total weight. An exact and deterministic implementation of back-off LMs with WFSA would require a different type of arc. The so called *failure arcs* were introduced for efficient string matching [Aho and Corasick(1975)]. Usually, in the literature (e.g. [Allauzen et al.(2003)]), a special arc label φ (or ϕ) is used to mark failure arcs. A failure-arc doesn't consume any symbol and it has the semantic interpretation, that a failure-arc can only be taken if no other symbol on any of the other out-going arcs of out of the same state can be accepted[1].

[1]Failure arcs have the peculiarity that the decision which arc to take is made based on the symbol, but the symbol is consumed later in the next non-failure arc. Usually, there are even several failure-arcs in a row, all testing the same input symbol, but against different outgoing arcs.

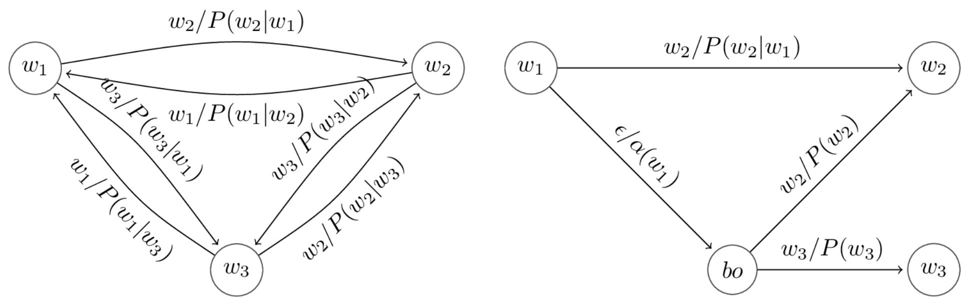

Figure 3.2: *Weighted finite state acceptor (WFSA) implementation of a bi-gram LM. Left: fully connected model ($V \times V$ arcs) Right: WFSA approximation of a bi-gram back-off model [Mohri et al.(2008)], just showing the representation of transitions leaving state w_1. The bi-gram $w_1 w_2$ was seen sufficiently often during training and is thus represented by a direct link between the history states w_1 and w_2. The bi-gram $w_1 w_3$ was not seen sufficiently often, thus the model backs-off to the history-less state "bo" with the cost of the back-off weight $\alpha(w_1)$. No symbol is consumed in this transition - indicated by the ϵ-symbol. Leaving the back-off state, the lower order (uni-gram) probabilities are applied ($P(w_3)$). The approximation with the back-off state can greatly reduce the number of arcs, but it also introduces non-determinism. If ϵ would be a regular label, the WFSA would be deterministic (only a single outgoing arc per label in each state). However, since ϵ doesn't consume a symbol, the bi-gram $w_1 w_2$ can be either formed by taking the arc $w_1 \rightarrow w_2$ or by going over the back-off arc: $w_1 \rightarrow bo \rightarrow w_2$. Usually, $P(w_2|w_1) \gg \alpha(w_1)P(w_2)$ (modeling error is small).*

This works similar to the 'default' case in a C-language 'switch' statement. These failure-arc-type WFSA accept sequences of words with exactly the same probabilities as when implemented as a back-off LM in any of the LM tool-kits. However, we actually violate the assumptions of the WFST algorithms, when using failure arcs, the semi-ring concept is changed and a new class of algorithms is needed.

3.1.1 Weight pushing

Two WFSAs are equal, if they accept the same set of input label sequences with the same path weights. In other words, two equivalent WFSAs (or WFSTs) may differ by the way the weights (and output labels) are distributed along the path. The distribution of weights along the path plays a crucial role in pruned search [Mohri and Riley(2001)]. According to [Mohri and Riley(2001)], the optimal distribution of weights for pruned search should be such, that the weights (coming from different knowledge sources such as acoustic and language model) are locally synchronized for the sequential decisions which state to take next. Another common wisdom is, that the knowledge should be applied as early as possible in search - to be able to rule out unlikely paths as early as possible. This corresponds to "pushing" the weights as much as possible towards the initial state, which is equivalent to making the WFSA stochastic [Mohri et al.(2008)] (the outgoing weights of a state sum to one ($\bar{1}$) in the given semi-ring). It is only possible to make the WFSA stochastic if the total weight of the entire WFSA is $\bar{1}$. The total weight of an WFSA is the sum of all successful paths (from the initial state to all of the final states).

Re-weighting [Mohri et al.(2008)] is an operation that alters the weights $w[t_i]$ of individual transitions and the final-probabilities $\rho(n[t_n])$, while leaving unaffected the weights $w[\pi]$ of successful paths. The possible ways to change the transition weights of a WFSA can be expressed with the help of a *potential function* $V : Q \rightarrow \mathbb{K} - \bar{0}$, which can be an arbitrary function on states Q, assigning a value of $\mathbb{K}$ (except $\bar{0}$) to every state q. Given such a function, we can update the initial weight λ, the transition weights $w[e], e \in E$ and the final weights $\rho(f)$ according to the following [Mohri and Riley(2001)]:

$$\begin{aligned} \lambda &\leftarrow \lambda \otimes V(i) && (3.3)\\ \forall e \in E, w[e] &\leftarrow [V(p[e])]^{-1} \otimes (w[e] \otimes V(n[e])) && (3.4)\\ \forall f \in F, \rho(f) &\leftarrow [V(f)]^{-1} \otimes \rho[f] && (3.5) \end{aligned}$$

If the re-weighting is carried out this way, the overall weight of a successful path is not changed, since the potentials along any successful path cancel each other. Thus, the resulting WFSA is equivalent to the original one. *Weight pushing* is a special case of re-weighting, that aims to make the WFSA stochastic (push the weights towards the initial state). This is achieved [Mohri and Riley(2001)] by setting the potential function $V(q)$ to the *shortest distance* $d[q]$ from q to any of the final states F:

$$\forall q \in Q,\, V(q) = d[q] = \bigoplus_{\pi \in P(q)} w[\pi] \tag{3.6}$$

Here, $P(q)$ is the set of all paths from q to any of the final states F. Thus, the crucial step in weight pushing is to run the shortest path algorithm, whose complexity depends on the given semi-ring and the type of WFSA that is dealt with.

For the tropical semi-ring, a Viterbi algorithm can be used. If the log-probability or probability semi-ring is used, however, all possible paths towards a state need to be summed, which is especially difficult, if the WFSA has cycles. A cycle can be followed an infinite amount of times, generating an infinite number of paths that need to be summed. So we need to guarantee that the weight of any cycle is $w(\pi) < \bar{1}$. In other words, we need to be able to compute the closure $\bigoplus_{i=1}^{\infty} w^i$, otherwise the cycle would result in an infinite total weight. If a semi-ring fulfills this condition for $\forall w \in \mathbb{K}$, it is called *closed semi-ring* [Mohri(2002)]. If the structure of the WFSA is simple, i.e. the cycles are not nested and can be easily identified, the closure operation could be directly applied. For WFSAs resulting from LMs, this is not true, since the cycles are nested in a complex way.

A generic shortest distance algorithm is presented by [Mohri(2002)] (page 9), which is iterative and operates by locally forwarding weight mass through the WFSA according to a queue policy. A state is put to the update queue, if its weight has changed since the last visit – called $r$elaxation condition. The algorithm assumes, that an update is not necessary, if the weight of a loop has already been multiplied k times (k-closed). For the (log-) probability semi-ring, there is no $k < \infty$, for which this would hold, thus the algorithm cannot be used. Closed semi-rings are covered by the generic Floyd-Warshall and Gauss-Jordan algorithms [Lehmann(1977)]. These algorithms compute the all-pair shortest distance with a time complexity of $O(n^3)$ (n proportional to the number of states) As soon as the WFSA has thousands of states, this is clearly not feasible.

The original relaxation condition for the generic algorithm [Mohri(2002)] is given by:

$$d[n[e]] \neq d[n[e]] \oplus (r' \otimes w[e]) \tag{3.7}$$

where $d[n[e]]$ is the shortest distance of state $n[e]$ and $(r' \otimes w[e])$ is the weight to be added in the update. To handle also semi-rings that are not k-closed, [Mohri(2002)] replaces the relaxation condition by an approximate test with a metric Δ:

$$\Delta(d[n[e]], d[n[e]] \oplus (r' \otimes w[e])) \geq \delta \tag{3.8}$$

where $\delta \geq \bar{0}$ is a positive number used for approximation. Due to limited machine precision, there is actually always some δ for which this condition will not be met. Thus, with a k-closed semi-ring, a cycle will not be followed more than k times, and in our case the algorithm stops updating as soon as:

$$l : \Delta\left(\bigoplus_{i=0}^{l} w[\pi]^i, \bigoplus_{i=0}^{l-1} w[\pi]^i\right) = w[\pi]^l \leq \delta \tag{3.9}$$

where $w[\pi]$ is the weight of the cycle. If l is large ($w[\pi] \to \bar{1}$ or $\delta \to \bar{0}$) the algorithm will iterate for a long time until it converges. For $w[\pi] \geq \bar{1}$, the algorithm fails to converge at all. In this case, the total weight of the WFSA becomes infinity.

As explained in figure 3.2, for WFSA constructed from back-off LMs, the ϵ-style back-off arcs lead to duplicate paths. In case the weights were taken from a back-off LM estimated for failure arcs, but the back-off arcs are represented with ϵ, the outgoing arcs will sum to a slightly higher value than one. When occurring in a loop, the condition $w[\pi] < \bar{1}$ does no longer hold. That means, the generic weight pushing algorithm [Mohri(2002)] as implemented in OpenFST will fail to converge, because the total weight of the entire WFSA will not be finite[2].

3.1.2 Alternative weight pushing algorithm

We present an alternative weight pushing algorithm, which will always converge. Using the transition matrix $\mathbf{P}_{ij}$, we represent the WFSA in the probability semi-ring, where p_{ij} is the sum of all transition weights between state i and state j. The transition matrix is usually sparse (contains $\bar{0}$ for all non-existing transitions). Our solution is based on the theory of non-negative matrices and ergodic Markov chains. The fundamental limit theorem for regular chains states, that for any initial probability vector $\boldsymbol{\lambda}$, the process approaches the fixed row vector $\mathbf{w}$ (stationary distribution) for $n \to \infty$:

$$\lim_{n\to\infty} \boldsymbol{\lambda}\,\mathbf{P}^n = \boldsymbol{\lambda}\,\mathbf{W} = \mathbf{w}. \tag{3.10}$$

This suggests an iterative algorithm to find the stationary distribution, which is similar to the power method for finding the dominant eigenvector $\mathbf{w}$ of the matrix $\mathbf{P}$, by starting from a random or uniform positive vector $\mathbf{v}$ and iterating by letting $\mathbf{v} \leftarrow \mathbf{P}\,\mathbf{v}$ each time.

If the WFSA is not normalized, the generalization is given by the Perron theorem, which [Berman and Shaked-Monderer(2012)] states, that for every non-negative primitive (i.e. regular) matrix $\mathbf{P}$, the maximum eigenvalue $\rho(\mathbf{P})$ (also called spectral radius) is positive, simple (algebraic multiplicity one), singular (only one eigenvalue of this modulus) and has a positive eigenvector (called left and right Perron vector, whose normalized entries sum to one). For $n \to \infty$, the matrix converges:

$$\lim_{n\to\infty}\left(\frac{\mathbf{P}}{\rho(\mathbf{P})}\right)^n = \mathbf{L},\ \mathbf{L} = \mathbf{x}^T\,\mathbf{y},\ \mathbf{x}\,\mathbf{y}^T = 1, \tag{3.11}$$

where $\mathbf{x}$ and $\mathbf{y}$ are positive right and left eigenvectors: $\mathbf{P}\,\mathbf{x}^T = \rho(\mathbf{P})\,\mathbf{x}^T,\ \mathbf{x} > 0,\ \mathbf{y}\,\mathbf{P} = \rho(\mathbf{P})\,\mathbf{y},\ \mathbf{y} > 0$. From this, it follows that for any initial probability vector $\boldsymbol{\lambda}$, if we multiply from the right, the ratio of state weights in $\boldsymbol{\lambda}$ will converge to a vector proportional to the right eigenvector $\mathbf{x}^T$:

[2]If the weights are correctly estimated as interpolated LM, the total weight is one, but the weight in cycles can still be very close to one, so that the generic algorithm is inefficient (equation 3.9).

$$\lim_{n\to\infty} \left(\frac{\mathbf{P}}{\rho(\mathbf{P})}\right)^n \boldsymbol{\lambda}^T = \mathbf{L}\,\boldsymbol{\lambda}^T = \mathbf{x}^T\,(\mathbf{y}\,\boldsymbol{\lambda}^T) = d\,\mathbf{x}^T. \tag{3.12}$$

Since we do not know the normalizing spectral radius $\rho(\mathbf{P})$ in advance, we re-normalize the resulting vector $\mathbf{v}$ at each step. Every normalization results in the same eigenvector, so we choose unit length.If we iterate $\mathbf{v} \leftarrow \mathbf{P}\,\mathbf{v}$ (equation 3.12), it results in the dominant *right* eigenvector of $\mathbf{P}$, which, in the probability semi-ring, is the minimum distance towards the final states (or the super-final state).

The Perron theorem is only true for regular chains, but we can make every trim WFSA ergodic[3] by connecting the final states $f \in F$ to the initial state I. Now, every state can be reached from any other state, by going over any of the final states. That means we modify one column in the transition matrix: if j is the initial state, then p_{ij} is set to the final-probability $\rho[i]$ of state i.

As a second step, we need to guarantee, that the resulting ergodic WFSA is also regular[4]. We can make the WFSA regular by interpolating $\mathbf{P}$ with the identity matrix: If $\mathbf{P}$ is the transition matrix of an ergodic Markov chain, then we can obtain the transition matrix of a regular chain by:

$$\mathbf{P}' = k\,\mathbf{I} + (1-k)\,\mathbf{P}\,,\; 0 < k < 1,\; k \in \mathbb{R} \tag{3.13}$$

Since the ergodicity guarantees, that every state can be reached, interpolating with the identity matrix $\mathbf{I}$ guarantees, that the diagonal elements $p'_{ii} > 0$ are positive, which means, that it is possible to take self-loops to stay in every state. Thus, after n steps, when all states of the ergodic chain have been reached, $\mathbf{P'}^n$ will have all elements positive. $\mathbf{P}'$ and $\mathbf{P}$ have the same eigenvectors[5].

Alternatively, we can modify the iteration to $\mathbf{v} \leftarrow \mathbf{P}\,\mathbf{v} + k\,\mathbf{v}$. The parameter k is set to a small value (0.1) to not slow down the convergence too much. This algorithm is very efficient in practice, it generally converges within several tens of iterations.

At the end, we have a vector $\mathbf{v}$ with $v_I = 1$, and a scalar $c > 0$, such that

$$c\,\mathbf{v} = \mathbf{P}\,\mathbf{v}. \tag{3.14}$$

The vector $\mathbf{v}$ contains the distribution of average state occupancies and is used as the potential function $V(q) : Q \to \mathbb{K} - \bar{0}$ for the re-weighting operation (equation 3.4). This means we compute a modified transition matrix $\mathbf{P}^*$, by letting

$$p^*_{ij} = p_{ij}\,v_i/v_j, \tag{3.15}$$

and transforming the final probabilities by $\rho^*_i = \rho_i\,v_I/v_i$, where v_I is the potential of the initial state. Using the re-weighting with the potential function V guarantees, that the resulting WFSA is equivalent to the original one. If we use the *right* Perron eigenvector as potential function in equation 3.4, it results in pushing the weights towards the initial state, i.e. in making the WFSA output stochastic. Either all outgoing arcs sum to one, if the total weight is one, or more generally to the same quantity for all states.

We show this by writing one element of equation 3.14 as

$$c\,v_i = \sum_j p_{ij}\,v_j, \tag{3.16}$$

[3]A Markov chain is ergodic, if it's possible to go from any state to any state (not necessarily one move).

[4]That is, we don't want the matrix to have several multiple eigenvalues with the same magnitude but different complex phases, as is the case, e.g. for linear WFSAs.

[5]If $k \ll 1$ or $k' \to 1$, then also the eigenvalue will be very similar.

by dividing by v_i, it easily follows that $c = \sum_j p^*_{ij}$. This means each row of the modified matrix $\mathbf{P}^*$ sums to c (the eigenvalue modulus of the Perron vector). In the classical weight pushing algorithm [Mohri and Riley(2001)], we assume a stochastic WFSA, so that after weight pushing, all outgoing transitions of a state "sum to" $\bar{1}$ in the given semi-ring. Our solution is to use a modified pushing operation, which results in a WFSA, for which the transitions out of all states (and the final probability ρ) "sum to" the same quantity c:

$$\forall q \in Q, \left(\bigoplus_{e \in E[q]} w[e] \right) \oplus \rho[q] = c. \tag{3.17}$$

This means that the left-over weight, which is usually added to the initial or final states and which can cause the standard algorithm to fail, is now uniformly "smeared" all over the WFSA. Our algorithm is in practice an order of magnitude faster than the more generic conventional weight pushing algorithm.

3.2 Exact back-off LM reversal

For the construction of the backward recognition network, the most difficult component to reverse is the WFSA resulting from the back-off LM. In section 3.1, we explained how N-gram back-off LMs are represented as WFSA. We require that the reverse LM assigns exactly the same probabilities as the forward LM. To guarantee an optimal search, the backward WFST should also be deterministic, stochastic and of minimal size. Thus, simple WFST reversal is not sufficient. We derive the construction of the backward LM satisfying these requirements, which is valid when using failure arcs and also when using epsilon arcs to represent the back-off structure as WFSA.

The WFSA corresponding to the forward LM accepts a sequence of words and accumulates the weights along the path - see figure 3.3. If the probability semi-ring is used, the path weight is the product of the individual probabilities. If logarithmic probabilities are used, the path weight is the sum of the individual scores. Two WFSA are equal, if they accept the same set of sequences with the same path weights. Thus, it is possible to distribute the weights differently along the path, as long as the total product (or sum for logarithmic weights) stays the same for all paths. When we directly apply WFSA reversal, which basically corresponds to swapping the source and destination states of the arcs, the resulting structure would be highly non-deterministic. In the example, starting backwards from the final state, all incoming arcs (only one example is shown) have the label $</s>$. Thus, we have to apply $P(</s>|c,d)$ after only having seen only one symbol of the tri-gram $(</s>)$. However, only after two more symbols d, c have been seen, the destination state can be determined unambiguously. For that reason, it would be logical to delay the application of the weight (probability), until a sufficient number of symbols (two for tri-grams) have been consumed to unambiguously determine the destination state. Figure 3.4 shows the corresponding path in the backward LM.

When certain N-grams do not have sufficient coverage in the training corpus and are approximated by backing-off to lower order N-grams (see figure 3.5), the sequence of the weights in the backward LM is again exactly reversed as in the forward LM, and the same delay of the weights is applied to make the model deterministic.However, the sequence of the labels for back-off arcs is changed - back-off weights and lower-order N-grams change their role. The reason for that is we always have to back-off to a common history before consuming the next label - so the failure-arc (symbolized by φ) in the backward model takes

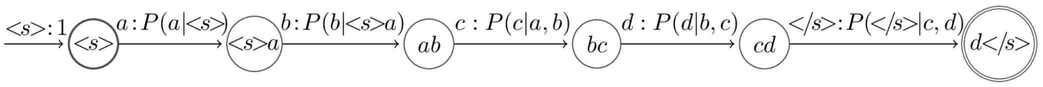

Figure 3.3: *Example of a forward path through a tri-gram language model - every state corresponds to a history of the two last symbols consumed. The model accepts the sequence a, b, c, d (input symbols) and the path weight is the product of the individual probabilities. For simplicity, sentence-start and sentence-end are treated here as ordinary symbols. Only one path is shown, but of course there are multiply arcs entering the final state, all with the same label.*

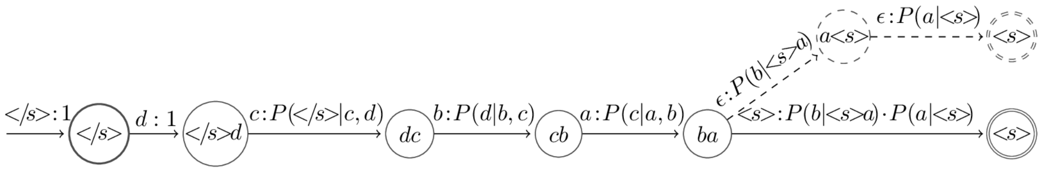

Figure 3.4: *The backward path corresponding to the path in figure 3.3. Additionally to reversing the path, the weights/probabilities have been delayed by two steps. Therefore, two arcs with probability one have been inserted at the beginning. To compensate, we could add two ϵ-arcs at the end (as depicted in dashed), where the last arc corresponds to backing-off to a history-less state at the sentence beginning (now end). Instead of introducing back-off arcs at the end of the sentence, we can collapse the probabilities of all the lower-order back-offs onto one arc, i.e. we use $P(b|{<}s{>}a) \cdot P(a|{<}s{>}) \cdot P({<}s{>})$. Also the WFST LM implementation assumes, that a state reached by an N-gram containing the sentence-end symbol is a final state.*

the lower-order N-gram probability (from the forward model) and the label-arc takes the former back-off weight. Figure 3.6 shows this in the construction of a backward back-off LM from figure 3.5. We see, that it is possible to construct a backward LM, that has the same size and structure as the forward LM and is deterministic. From the construction, we observe, that a forward LM can be transformed into a backward LM by a series of relatively simple steps: Since the sequence of labels is processed in reversed order, the names of all states and N-grams are reversed (*abc* becomes *cba*). The N-grams of the highest order do not have back-off weights, and thus they stay unchanged (arcs appear similar in the forward and backward models). However, for all lower-order N-grams, the role of the back-off weight and the N-gram probability change.

When represented in the ARPA format in figure 3.7, the transformation becomes even simpler: For all lower-order N-grams, the whole line is reversed, and for the highest-order N-gram, only the N-gram is reversed. E.g. for a tri-gram LM, a bi-gram entry $P(b|a)\ a\,b\ \alpha(a,b)$ becomes $\alpha(a,b)\ b\,a\ P(b|a)$ and a tri-gram entry $P(c|a,b)\ a\,b\,c$ becomes $P(c|a,b)\ c\,b\,a$. The symbols for sentence begin and sentence end have to be exchanged, and special care has to be taken for N-grams starting and ending a sentence. For all N-grams ending a sentence, we multiply all lower-order probabilities (e.g. for N-gram *cba* we use $P(c|a,b) \cdot P(b|a) \cdot P(a)$).

With the help of a series of weight pushing operations and representation changes of the probabilities, where each step guarantees WFSA equivalence, the LM reversal algorithm can also be derived step by step. By applying the constraint that the joint word probabilities should be the same for the forward and backward LM for all N-gram orders, we are also able to derive our algorithm from Bayes' rule. The application of weight pushing to the resulting backward LM is crucial for optimal performance.

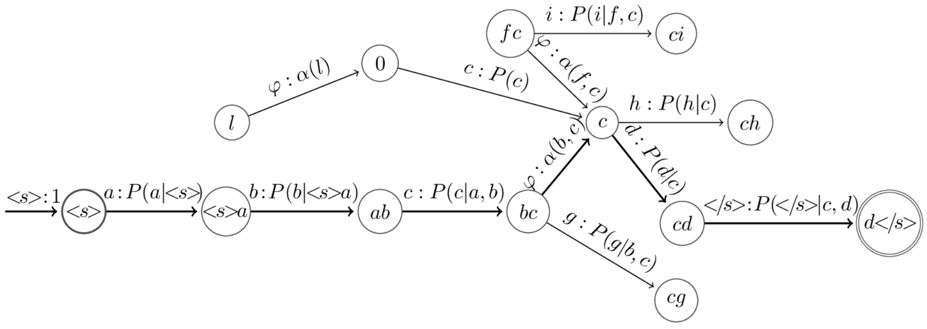

Figure 3.5: *The same example of a forward path as in figure 3.3, but with backing-off. The thick arcs correspond to the path in fig. 3.3 and further arcs have been added to illustrate the structure of the back-off LM. Since the N-gram bcd was not seen sufficiently often, it is approximated by backing-off to state c with back-off weight $\alpha(b, c)$ and then using the bi-gram cd with probability $P(d|c)$. The failure-arc (symbolized by φ) doesn't consume any symbol, but this arc is chosen for all symbols, that have no outgoing arc out of the same state ('default' clause). For the non-deterministic WFSA approximation, we would use the symbol ϵ instead and not consume a symbol either. The state 0 corresponds to the history-less back-off state when backing-off to uni-grams.*

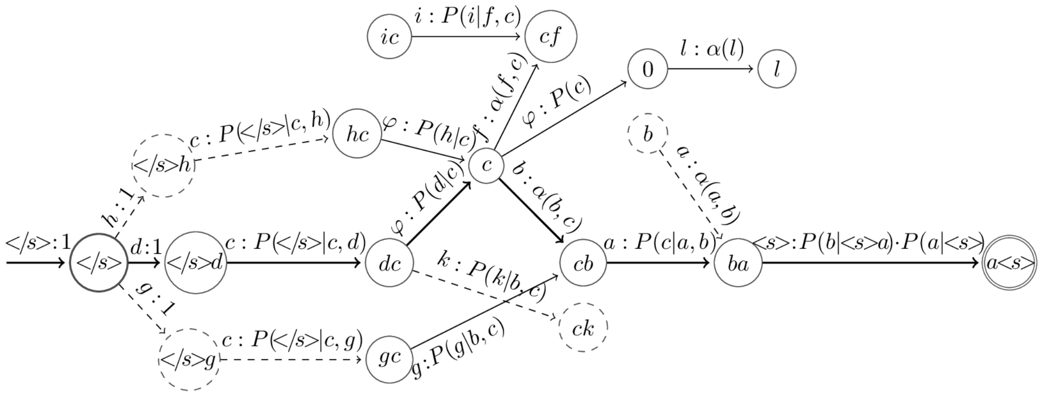

Figure 3.6: *The backward structure corresponding to figure 3.5. The thick arcs correspond to the path in fig. 3.4, and all solid arcs are reversed arcs from fig. 3.5. Dashed arcs have been added to illustrate further structure of the backward model. Similar as in figure 3.4, the weights have been delayed by two steps. Compared to the forward structure in figure 3.5, the sequence of weights is exactly reversed. The probability on an arc between two particular states is the same in the forward and backward model. I.e. compare the forward arcs $bc - c - cd$ in fig. 3.5 to the backwards arcs $dc - c - cb$ in this figure (cb corresponds to bc). However, since all labels are off by two states in the backward model, the back-off probability $\alpha(b, c)$ is now actually applied on a bi-gram arc with a label (b) and the bi-gram probability $P(d|c)$ is applied on a back-off arc with φ. Since all backward-tri-grams ending in cb (like dcb, hcb) share b as last label, it is logical to first back-off from the history (dc, hc) to the common history c and then apply the common label b. Since the reverse order of the weights has been preserved, the bi-gram probabilities serve now as back-off weights, and the former back-off weights serve as bi-gram probabilities. The same holds for the history-less state 0 - the uni-gram back-off weight $\alpha(l)$ and the uni-gram probability $P(c)$ have switched their role.*

```
\data\
ngram 1=4
ngram 2=2
ngram 3=2

\1-grams:
-5.234679 a -3.3
-3.456783 b
0.0000000 <s> -2.5
-4.333333 </s>

\2-grams:
-1.45678 a b -3.23
-1.30490 <s> a -4.2

\3-grams:
-0.34958 <s> a b
-0.23940 a b </s>
\end\
```

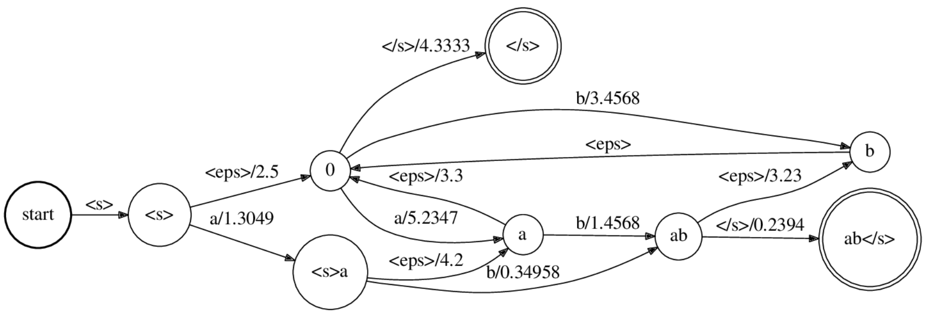

Figure 3.7: Left: *Definition of a tri-gram ARPA back-off language model. For each N-gram 'abc', there is an entry in the form '$P^*(c|a,b)$ abc $\alpha(a,b,c)$', where $P^*(c|a,b)$ is the discounted probability $P(c|a,b)$, and $\alpha(a,b,c)$ is the back-off weight of backing-off from a higher order N-gram to the shortened history abc. The probabilities are by convention given as logarithms to the basis of two.* **Right:** *The WFSA resulting from the tri-gram back-off ARPA LM defined on the left. The highest-order N-grams (tri-grams) behave slightly differently than lower-order N-grams: The transition for tri-gram <s>ab is going from state <s>a to state ab, which is equivalent to going to an imaginary state <s>ab and immediately backing-off to state ab. If for some reason the bi-gram ab would be missing in the ARPA file (removed line '$-1.45678\,a\,b\ -3.23$'), the state ab would be created as target state for the tri-gram abc, however, the arc from a to ab would not exist, and the back-off arc from ab to b would be with zero cost.*

3.2.1 The treatment of missing N-grams

While the rules are rather simple, an additional complexity arises, when representing back-off N-gram LMs as WFSAs. If there is an N-gram entry for *abcd* in the ARPA, the resulting WFSA needs the back-off states *bcd*, *cd* and *d*. Due to e.g. LM pruning, for some of the N-grams *abcd* defined in the ARPA file, there is no corresponding tri-gram entry *bcd* or bi-gram entry *cd*, i.e. we are not given the probality $\alpha(bcd)$ of backing-off $abcd \rightarrow bcd$, neither $P(d|b,c)$. N-grams that are needed for the construction of the WFSA, but not defined in the ARPA, we call missing N-grams.

During the construction of the recognition graph from the forward LM, missing back-off states are usually added automatically. For example, in the tri-gram LM of figure 3.7, the tri-gram *<s>ab* leads into the state *ab*. Let's imagine the corresponding back-off bi-gram *ab* is not present in the LM: In this case, during the construction of the recognition graph, the state *ab* needs to be automatically created, as it is the target of the tri-gram. Since there is no bi-gram probability for *ab* ($P(b|a) = 0.0$), we should immediately back-off to state *b*. Thus, the bi-gram *ab* is added with back-off weight $\alpha(a,b) = 1.0$ (zero in log-domain). However, it should not be possible to reach the newly created state *ab* from *a*, since the N-gram *ab* is missing ($P(b|a) = 0.0$ or minus infinity in log-domain).

In terms of the WFSA representation of the LM (right part of figure 3.7), this would mean, that there would be no link between *a* and *ab*, and the link between *ab* and *b* would be added with zero cost. In the reverse LM, where forward probability and back-off weight change their role, this does lead to the situation, that we are able to reach *ba* from *b* with $\alpha(a,b) = 1.0$, but we are not able to back-off from *ba* to *a*, since this corresponds to a path that was not present in the forward model ($P(b|a) = 0.0$). To summarize, to make missing N-grams explicit in the forward ARPA file, it results in an entry '$-$`inf` $a\,b$ 0.0', and in the backward ARPA, it results in an entry '0.0 $b\,a\ -$`inf`'.

Missing N-grams result from a complex interplay of the type of back-off distribution, cut-off frequencies and LM pruning. The first type of N-grams that are missing in the ARPA file are back-off N-grams that end a sentence. By convention, the state reached by an N-gram containing the sentence-end symbol is a final state. For example, for the tri-gram *cd</s>* in figure 3.5 (final arc), there is no lower-order N-gram *d</s>*, since after observing the sentence-end symbol, no other N-gram can follow. According to the rule, we create '$-$`inf` *d </s>* 0.0', which results in adding '0.0 *</s> d* $-$ `inf`' in the backward LM. This exactly corresponds to the state *</s>d* in figure 3.6, which can be reached with probability one, but there can be no back-off arc leaving this state.

Otherwise, in an un-pruned N-gram LM, usually the presence of a higher-order N-gram implies the presence of the lower-order N-gram (e.g. with a shortened history), since the observation count of the lower-order N-gram should be equal or higher than the count of the higher-order N-gram. We encounter missing N-grams, when we use different cut-off frequencies (parameter k in equation 3.2) for different N-gram orders, and also, if we use lower-order distributions, which are not based on counts. The distribution for the highest-order N-grams $P'(w_i|h_i)$ (equation 3.2) is usually based on the counts $C(h_i, w_i)$. [Kneser and Ney(1995)] showed, that when backing-off, one should make use of the fact that this particular word is unseen in the given context. In other words, we should use a different type of distribution for $P_{lower}(w_i|\bar{h_i})$ than for $P'(w_i|h_i)$. [Kneser and Ney(1995)] use a back-off distribution, where the probability of a word, unseen in a certain context, is proportional to the number of possible predecessor words types that can occur before that context:

$$P_{backoff}(w_i|w_{i-n+1}\dots w_{i-1}) = \frac{|w_{i-n} : C(w_{i-n}\dots w_i) > k|}{\sum_{w_i} |w_{i-n} : C(w_{i-n}\dots w_i) > k|}. \tag{3.18}$$

As a consequence, we can expect words or phrases, that appear frequently, but only in very few different contexts, to have a low probability in the back-off model. For example, we would not expect the word „Francisco“ to appear in many other contexts than together with „San Francisco“, despite the fact that it is a frequent word. For that reason, we often find higher-order N-grams in the LM, such as „San Francisco area“, for which the back-off N-gram „Francisco area“ is missing. When constructing the backward LM, we will add the missing N-gram „area Francisco“ with probability one and infinite back-off weight. This means, that after observing „area Francisco“, we are only able to continue with „San“ and there can be no back-off to „Francisco“, which would allow to continue with another word.

In fact, when experimenting with LMs trained on sentences from the Wall Street Journal corpus [Paul and Baker(1992)], we observed that any common multi-word phrase can result in missing lower-order N-grams. An N-gram starting within a multi-word phrase has very few different left contexts, which causes it to have low back-off probability. If the right context of that N-gram is either almost completely undetermined or completely determined (e.g. sentence end), all N-grams that would continue the phrase fall below the cut-off frequency and are thus not present in the LM. Typically, a multi-word phrase like „on behalf of“ or „New York City“ is followed by a word that introduces lot's of ambiguity - e.g. „on behalf of **the**“. If no N-gram „behalf of the X“ is above the cut-off frequency, then also the back-off N-gram „behalf of the“ is missing in the LM, since the probability of seeing it in a new context other than „on“ is extremely low. As already mentioned, also for all N-grams ending a sentence, there is no succeeding N-gram, which is a similar situation. It is quite obvious, that LM pruning (e.g. based on entropy [Stolcke(1998)]) will increase the number of missing N-grams. According to the same principle, N-grams with a low probability in the back-off distribution, and no successor N-grams (due to pruning) are missing as well.

Chapter 4

Conclusions

In this thesis, we introduced the idea of symmetrically decoding forwards and backwards in time. For some tasks, the pruned backward search can be more efficient than the forward search. Moreover, we showed, that the search errors of forward and backward search are mutually independent. To concentrate on search errors rather than on modeling errors, we require both decoding passes to be symmetric – i.e. both models are equally powerful and are constructed to assign exactly the same probabilities to hypotheses. This guarantees that each difference in comparing the results of forward and backward decoding corresponds to a search error. For most of the time frames in beam search decoding, a very narrow beam is sufficient. Therefore, we decode with a variable beam width – using a small baseline beam and only increasing it in places, where the forward and backward searches disagree.

One possible realization of the variable beam width decoding is to run both passes in parallel, iteratively refining the decoding (by increasing the beam width) in places, where both disagree. For about 50% of the utterances, the results already match after the first iteration. For the rest, the stretches of mis-matching words can be decoded in parallel.

Another realization of the variable beam width is the tracked decoding presented in this thesis, which runs forward and backward decoding sequentially. During the second pass (tracked decoding), we identify active tokens corresponding to paths that were present in the first-pass lattice. These tracked tokens are never pruned, regardless of the beam width and are used to determine the variable pruning beam for each frame. In places where disagreement is detected, the beam is increased to include all of them. Otherwise, the same narrow beam is used as in the first pass. Our method is doing more than simply choosing the best path from two passes, because it is possible to "recombine" partial paths from the first-pass and second-pass search. On top of that, the variable beam leads to the generation of extra tokens in areas where both passes disagree, which gives an additional speed-up.

Tracked decoding leads to a 2-3 times speed-up compared to a single pass forward decoding. Since most of the time is spent in the forward and backward decoding with the narrow beam, the baseline beam determines the possible speed-up. When we decrease the beam below a critical threshold the speed-up vanishes, since an excessive amount of extra tokens is generated. At least one of the two passes should obtain a (partly) correct solution.

To construct the backward recognition network, it is not sufficient to apply WFST reversal to the forward network. To guarantee an optimal search (determinism, stochasticity, minimal size), it is necessary to construct reverse models for each component separately and to compose the components in the same way as in the forward network. It turned out that the transducers for HMM structure, context-dependency and pronunciation lexicon are rather easy to reverse, however, the reversal of the LM transducer is difficult. Weight pushing has to be applied to the reversed components, to obtain the stochasticity of outgoing arcs. Since all components are reversed individually, our approach to the construction of backward recognition networks can be applied in static and dynamic network decoders.

To represent N-gram LMs as WFSTs, an approximate structure is necessary. When representing back-off arcs as epsilon arcs, non-determinism is introduced, resulting in an infinite total weight. The convergence of the iterative weight pushing algorithm depends on the weight in a loop, which must be smaller than one. Therefore, the algorithm will not converge for WFSA resulting from back-off LMs. We presented an alternative weight pushing algorithm, which will always converge. We use the Perron theorem to obtain the dominant right eigenvector of the transition matrix of an ergodic WFST. This vector represents the minimum distance to the final state, which we use as the potential function in re-weighting. This results in pushing the weights towards the initial state and making the WFSA output stochastic. The total weight, causing the standard algorithm to fail, is now uniformly "smeared" all over the WFSA. Our algorithm is in practice an order of magnitude faster than the more generic conventional algorithm.

The most difficult component to reverse is the WFST resulting from the back-off LM. We require that it assigns exactly the same probabilities as the forward LM. We derive the construction of the backward LM satisfying these requirements, which is valid when using exact back-off models using failure arcs, and also when approximating them with epsilon arcs. We especially concentrate on the correct handling of back-off arcs and missing N-grams. Our 'exact' LM reversal gives slightly better performance than a backward LM resulting from training on the reversed training texts.

4.1 Future work

The proposed method could be applied in the fast generation of lattices for audio indexing and the tracked decoding could be used to generate lattices that contain desired paths, such as the forced-alignment reference for the discriminative training of acoustic models. Additionally to decoding forwards and backwards in time, depending on the task, there might be other ways of decoding, which could result in independent search errors.

The alternative weight pushing algorithm was derived under the assumption, that all arcs in the WFST are of the same type. However, there are "emitting" arcs with a word label, and "non-emitting" arcs representing the back-off arcs. An open problem is to derive a weight pushing algorithm respecting the special semantics of back-off arcs as failure arcs. Under this correct interpretation, the total weight of the transducer will be one, and we avoid the negative log-probabilities resulting from pushing weights greater than one. The original Kaldi recipe for the construction of recognition networks [Povey et al.(2011)] used the assumption, that all components are stochastic, which eliminates the necessity for weight pushing. We want to derive a properly normalized stochastic backward LM WFST.

There is inconsistency between the algorithms for decoding graph construction (assuming the log-semi-ring), and for decoding (tropical semi-ring). In addition to different interpretations of the failure/epsilon arcs, this opens several design choices, which should be systematically explored to find a consistent framework for decoding graph construction that results in an optimal decoding. The non-determinism introduced by using epsilon arcs for back-offs results in multiple evaluations of the same models during decoding. After the composition with the lexicon transducer, it should be possible to apply another determinization step, which not only removes the non-determinism, but also leads to a graph, which is consistent with the log-semi-ring.

Bibliography

[Abo-Gannemhy et al.(2010)] W. Abo-Gannemhy, I. Lapidot, and H. Guterman, "Speech recognition using combined forward and backward Viterbi search." in *IEEE Convention of the Electrical and Electronic Engineers in Israel*, 2010.

[Agarwal et al.(2014)] A. Agarwal, E. Akchurin, C. Basoglu, G. Chen, S. Cyphers, J. Droppo, A. Eversole, B. Guenter, M. Hillebrand, R. Hoens, X. Huang, Z. Huang, V. Ivanov, A. Kamenev, P. Kranen, O. Kuchaiev, W. Manousek, A. May, B. Mitra, O. Nano, G. Navarro, A. Orlov, M. Padmilac, H. Parthasarathi, B. Peng, A. Reznichenko, F. Seide, M. L. Seltzer, M. Slaney, A. Stolcke, Y. Wang, H. Wang, K. Yao, D. Yu, Y. Zhang, and G. Zweig, "An introduction to computational networks and the computational network toolkit." Tech. Rep. MSR-TR-2014-112, August 2014. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=226641

[Aho and Corasick(1975)] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search." *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[Allauzen et al.(2003)] C. Allauzen, M. Mohri, and B. Roark, "Generalized algorithms for constructing statistical language models." in *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics – Volume 1*, ser. ACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 40–47.

[Allauzen et al.(2004)] C. Allauzen, M. Mohri, M. Riley, and B. Roark, "A generalized construction of integrated speech recognition transducers." in *Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004. (ICASSP '04)*, vol. 1, May 2004, pp. I–761–4 vol.1.

[Austin et al.(1991)] S. Austin, R. Schwartz, and P. Placeway, "The forward-backward search algorithm." in *Proc. ICASSP*, 1991, pp. 697–700.

[Bellman(1952)] R. Bellman, "On the theory of dynamic programming." *Proceedings of the National Academy of Sciences*, vol. 38, no. 8, pp. 716–719, 1952.

[Berman and Shaked-Monderer(2012)] A. Berman and N. Shaked-Monderer, "Non-negative matrices and digraphs." in *Computational Complexity*, R. A. Meyers, Ed. Springer New York, 2012, pp. 2082–2095.

[Cardinal et al.(2013)] P. Cardinal, P. Dumouchel, and G. Boulianne, "Large vocabulary speech recognition on parallel architectures." *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 11, pp. 2290–2300, Nov 2013.

[Fiscus(1997)] J. G. Fiscus, "A post-processing system to yield reduced word error rates: Recognizer Output Voting Error Reduction (ROVER)." in *Proceedings 1997 IEEE Workshop on Automatic Speech Recognition and Understanding*, Dec 1997, pp. 347–354.

[Hannemann et al.(2013)] M. Hannemann, D. Povey, and G. Zweig, "Combining Forward and Backward Search in Decoding." in *Proc. ICASSP 2013*, 2013, pp. 6739–6743. [Online]. Available: http://www.fit.vutbr.cz/research/view˙pub.php.en?id=10324

[Jouvet and Fohr(2013b)] D. Jouvet and D. Fohr, “Combining Forward-based and Backward-based Decoders for Improved Speech Recognition Performance.” in *Proc. Interspeech 2013 - 14th Annual Conference of the International Speech Communication Association*, 2013.

[Jouvet and Fohr(2014)] ——, “About Combining Forward and Backward-Based Decoders for Selecting Data for Unsupervised Training of Acoustic Models.” in *Proc. Interspeech 2014*, 2014, pp. 815–819.

[Jouvet and Fohr(2013a)] ——, “Analysis and Combination of Forward and Backward Based Decoders for Improved Speech Transcription.” in *Text, Speech, and Dialogue*, ser. Lecture Notes in Computer Science, I. Habernal and V. Matoušek, Eds. Springer Berlin Heidelberg, 2013, vol. 8082, pp. 84–91. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40585-3˙12

[Katz(1987)] S. M. Katz, “Estimation of probabilities from sparse data for the language model component of a speech recognizer.” in *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35(3), 1987, pp. 400–401.

[Kneser and Ney(1995)] R. Kneser and H. Ney, “Improved backing-off for M-gram language modeling.” in *Proc. ICASSP-95, International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, May 1995, pp. 181–184.

[Lee and Kawahara(2009)] A. Lee and T. Kawahara, “Recent development of open-source speech recognition engine Julius.” in *Proc. APSIPA Annual Summit and Conference*, 2009.

[Lee et al.(1998)] A. Lee, T. Kawahara, and S. Doshita, “An efficient two-pass search algorithm using word trellis index.” in *Proc. ICSLP*, 1998.

[Lehmann(1977)] D. J. Lehmann, “Algebraic structures for transitive closure.” *Theoretical Computer Science*, vol. 4, pp. 59–76, 1977.

[Li et al.(2009)] T. Li, W. Xu, J. Pan, and Y. Yan, “Improving automatic speech recognizer of voice search using system combination.” in *Sixth International Conference on Fuzzy Systems and Knowledge Discovery, 2009. FSKD ’09*, vol. 4, Aug 2009, pp. 477–480.

[Lowerre(1976)] B. Lowerre, “The Harpy Speech Recognition System.” Ph.D. dissertation, Carnegie Mellon University, 1976.

[Maleki et al.(2014)] S. Maleki, M. Musuvathi, and T. Mytkowicz, “Parallelizing Dynamic Programming Through Rank Convergence.” in *Proc. ACM PPoPP’14*. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), February 2014. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=208241

[Mohri(2002)] M. Mohri, “Semiring frameworks and algorithms for shortest-distance problems.” *Journal of Automata, Languages and Combinatorics*, vol. 7, pp. 321–350, March 2002.

[Mohri and Riley(2001)] M. Mohri and M. Riley, “A Weight Pushing Algorithm for Large Vocabulary Speech Recognition.” in *Proc. Eurospeech 2001, 7th European Conference on Speech Communication and Technology*, 2001.

[Mohri et al.(2008)] M. Mohri, F. C. N. Pereira, and M. Riley, “Speech recognition with weighted finite-state transducers.” in *Handbook on Speech Processing and Speech Communication, Part E: Speech recognition*, L. Rabiner and F. Juang, Eds. Heidelberg, Germany: Springer-Verlag, 2008, p. 31.

[Murveit et al.(1993)] H. Murveit, J. W. Butzberger, V. V. Digalakis, and M. Weintraub, “Large-vocabulary dictation using SRI’s decipher speech recognition system: Progressive search techniques.” in *Proc. ICASSP Vol. 2*, 1993, pp. 319–322.

[Nguyen et al.(1993)] L. Nguyen, R. Schwartz, F. Kubala, and P. Placeway, “Search algorithms for software-only real-time recognition with very large vocabularies.” in *Proceedings of the Workshop on Human Language Technology*, 1993, pp. 91–95.

[Nolden et al.(2013)] D. Nolden, R. Schlüter, and H. Ney, “Efficient nearly error-less LVCSR decoding based on incremental forward and backward passes.” in *Proceedings ASRU 2013, IEEE Workshop on Automatic Speech Recognition and Understanding*, 2013, pp. 1–6.

[Paul and Baker(1992)] D. B. Paul and J. M. Baker, “The Design for the Wall Street Journal-based CSR Corpus.” in *DARPA Speech and Language Workshop.* Morgan Kaufmann Publishers, 1992.

[Povey et al.(2011)] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, “The Kaldi speech recognition toolkit.” in *Proc. ASRU.* IEEE, 2011.

[Povey et al.(2012)] D. Povey, M. Hannemann, G. Boulianne, L. Burget, A. Ghoshal, M. Janda, M. Karafiat, S. Kombrink, P. Motlicek, Y. Quian, N. Thang Vu, K. Riedhammer, and K. Vesely, “Generating exact lattices in the WFST framework.” in *Proc. ICASSP.* IEEE, 2012, pp. 4213–4216.

[Stolcke(1998)] A. Stolcke, “Entropy-based pruning of backoff language models.” in *Proceedings DARPA Broadcast News Transcription and Understanding Workshop.* Morgan Kaufmann, February 1998, pp. 270–274.

[Tang and Cristo(2008)] M. Tang and P. D. Cristo, “Backward viterbi beam search for utilizing dynamic task complexity information.” in *Proc. Interspeech*, 2008, pp. 2090–2093.

[van Hamme and van Aelten(1996)] H. van Hamme and F. van Aelten, “An adaptive-beam pruning technique for continuous speech recognition.” in *Proc. ICSLP 96, Fourth International Conference on Spoken Language Processing*, vol. 4, Oct 1996, pp. 2083–2086.

[Viterbi(1967)] A. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.” *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, April 1967.