# On Integrating the Lexicon with the Language Model

*Diamantino Caseiro, Isabel Trancoso*

$L^2F$ Spoken Language Systems Lab.
INESC-ID/IST
Rua Alves Redol 9, 1000-029 Lisbon, Portugal
dcaseiro@l2f.inesc.pt Isabel.Trancoso@inesc.pt

## Abstract

The goal of this work was to develop an algorithm for the integration of the lexicon with the language model which would be computationally efficient in terms of memory requirements, even in the case of large trigram models.

Two specialized versions of the algorithm for transducer composition were implemented. The first one is basically a composition algorithm that uses the precomputed set of the output labels that can be reached from a particular epsilon edge of the lexicon; the second includes an ”on the fly” implementation of the pushing of weights and output labels. Very significant memory savings were obtained with the proposed algorithms compared with the general determinization algorithm for weighted transducers.

## 1. Introduction

The explicit integration of the lexicon with the language model generates very large unmanageable graphs, and has been regarded as applicable only to small bigram language models. Some approaches have been proposed to reduce the size of that graph by taking advantage of the sparse nature of the knowledge sources used in large vocabulary continuous speech recognition [1][2].

Most approaches were developed in an had hoc fashion for the data structures used. In [2] a very general and well founded technique was proposed that allows the use of general n-gram models. The main problem with that approach is the large memory requirements for the transducer determinization algorithm that is used to reduce the search space. This problem limits the use of this technique to smaller language models than otherwise would be possible. We propose an algorithm with very small memory requirements to approximate that technique.

Our algorithm is based on the observation that the traditional algorithm for transducer composition [3], when used to combine the lexicon with the language model, will essentially replicate the structure of the lexicon locally on the composition. In some cases, it is possible to directly build a sequential composition.[1] And, when the lexicon is sequential, the composition will also be mostly deterministic.

However the transducer composition algorithm cannot be applied blindly to this task due to the large number of output epsilons in the lexicon. The problem is that, until finding a non-epsilon output, the algorithm will happily follow all epsilons generating lots of dead-ends.

Not only is space and time required to generate those dead-ends, but they also have to be removed by an additional pass over the resulting graph.

The generation of dead-ends is avoided in the AT&T approach by the use of a linear lexicon in the composition where the first edge of every pronunciation outputs the word.

We add some lookahead information to the lexicon to avoid the generation of dead-end paths during the composition. This information is associated with epsilon outputting edges and takes the form of the set of non-epsilon output labels reachable from the edge.

Our composition algorithm is modified to take into account those sets when following an epsilon output edge of the lexicon, so that the edge is only considered if it reaches a compatible output.

An important factor that we took in account when designing the algorithm was to design it in such a way as to allow “on-the-fly” generation of the composition transducer.

The algorithm itself is formally described in section 2; the following section presents an ”on the fly” implementation of the pushing of weights and output labels; section 4 discusses some implementation issues; section 5 describes our experimental results; and the final section summarizes the main conclusions.

## 2. Algorithm for efficient integration of the lexicon and the language model

### 2.1. Notation

In this paper we represent a weighted transducer as a tuple $(Q, i, F, \Sigma, \Delta, E, R)$ where:

- $Q$ is the set of states,
- $i \in Q$ is the initial state,
- $F \subset Q$ is the set of final states,
- $\Sigma$ is the set of input labels,
- $\Delta$ is the set of output labels,
- $E$ is the finite-set of edges $Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times Q \times R$, each edge is characterized by an initial state, an input label, an output label, a destination state and a weight.
- $R$ is the weight semiring $(W, \oplus, \otimes, \bar{0}, \bar{1})$ that defines the operations we can do with the weights. [2]

We will use superscripts to distinguish the symbols that belong respectively to the lexicon $l$, the language model (or grammar) $g$ or to their composition $lg$.

[1] A sequential transducer is a transducer that is deterministic on the input side and has no epsilons (skips) on the input.

[2] In speech recognition, in particular, it is common the use of the semiring of probabilities $([0,1], \times, +, 0, 1)$, and of the semiring of (minus) $log$ likelihoods $(\Re^+ \cup \{\infty\}, +, min, \infty, 0)$.

```
input: $q = (q^l, q^g)$ {$q$ is a state of $lg$}
$A \leftarrow \{\}$
for all $a^l = (q^l, l^l, o^l, d^l, w^l)$ in the lexicon do
  $s^l \leftarrow destSet(a^l)$
  if $(s^l = \{\})$ then
    $A \leftarrow A \overset{\oplus}{\cup} \{(q, l^l, \epsilon, (d^l, q^g), w^l)\}$
  else
    for all $a^g = (q^g, l^g, o^g, d^g, w^g)$ in the grammar do
      if $(l^g = \epsilon)$ then
        if $(q^l = i^l)$ then
          $A \leftarrow A \overset{\oplus}{\cup} \{(q, \epsilon, o^g, (q^l, d^g), w^g)\}$
        end if
      else if $(l^g \in s^l)$ then
        if $(o^l = \epsilon)$ then
          $A \leftarrow A \overset{\oplus}{\cup} \{(q, l^l, o^g, (d^l, q^g), w^l)\}$
        else
          $A \leftarrow A \overset{\oplus}{\cup} \{(q, l^l, o^g, (d^l, d^g), w^l \otimes w^g)\}$
        end if
      end if
    end for
  end if
end for
return $A$
```

Figure 1: Algorithm for determining the set of edges $A$ leaving a state $q = (q^l, q^g)$ in the composition transducer.

### 2.2. The composition algorithm

The composition transducer $lg = l^* \circ g$ is defined as:

- $Q^{lg} = Q^l \times Q^g$,
- $\forall_{(f^l, f^g) \in Q^{lg}}, (f^l, f^g) \in F^{lg}$ iif $f^l \in F^l \wedge f^g \in F^g$,
- $\Sigma^{lg} = \Sigma^l$
- $\Delta^{lg} = \Delta^g$
- $E^{lg}$ is generated by the algorithm in figure 4 that takes as input a state $q = (q^l, q^g)$ and returns the set $A$ of edges leaving that state.

As we can see, the algorithm is just a composition algorithm that uses a function $destSet$ to determine the output labels that will be reached from a particular epsilon edge of the lexicon. The $\overset{\oplus}{\cup}$ operator used in the algorithm is just a union operation that, when merging multiple edges that differ only on the weight, keeps a single edge with the "sum" ($\oplus$) of the weights. In figure 4 we illustrate the result of applying the algorithm to the sample lexicon and language models shown in figures 2 and 3.

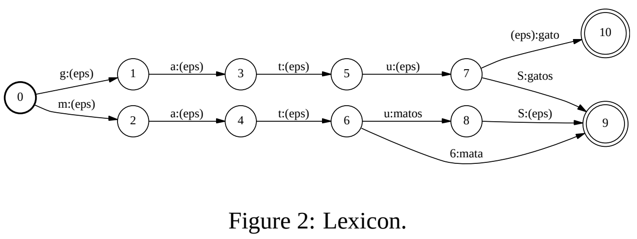

Figure 2: Lexicon.

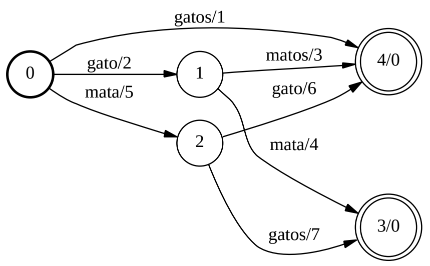

Figure 3: Language model.

### 2.3. Lexicon

Since the algorithm replicates parts of the lexicon throughout the composition transducer, it depends critically on the structure of the lexicon transducer. In fact, equivalent transducers can originate very different search spaces. For example, if epsilon edges are used to implement the lexicon loop, they will be replicated sub-optimally.

To build the lexicon loop transducer, we start with a linear lexicon *WFST* with one initial and one final states, then it is determinized and minimized [3]. Finally, the loop is build by replacing the destination of every edge that ends in a final state, with the initial state.

The algorithm depends on the initial state being "in the loop" for determining the initial state of a pronunciation.

The algorithm does not require the use of a sequential lexicon, in the sense that the composition algorithm will perform as expected even if that condition is not satisfied.

Generally, a lexicon transducer is not sequential due to homophone words and to the fact that the pronunciation of some words can be a subsequence of others. But, it can be turned into one with the addition of dummy disambiguating labels to the pronunciations.

### 2.4. Language model

In this work we consider the language model to be represented by a *WFST*. That is very natural for finite state grammars and word lattices, but is not so for n-gram language models. A "correct" implementation of an $n$-gram language model requires $v^n$ edges, where $v$ is the size of the vocabulary. Such a large number of edges is not acceptable, so the usual approach when representing n-gram language models as "graphs" consists in using epsilon edges to implement the backoff component of the models, approximating the language model with a *WFST* with a number of edges proportional to the number of parameters of the model [4]. The language model does not normally need to be represented as a transducer. It can be represented with an acceptor. Nevertheless, we choose to represent it with a transducer, noting that an acceptor can be represented as a transducer with identical input and output labels in each edge.

## 3. Pushing

As the first step in the direction of an approximation to "on the fly" minimization, we implemented an approximation of push-

[3]We rely on the AT&T FST Tools for determinization of *WFSTs* and minimization of automata.

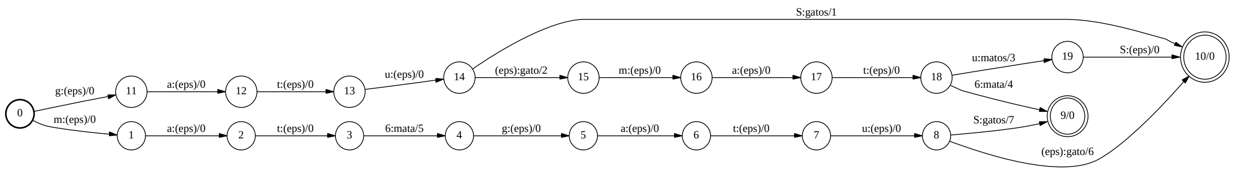

Figure 4: Composition of the lexicon with the language model.

ing. It is useful to push the output labels and the output weights as much towards the initial state as possible. Pushing the output labels will allow further sharing of suffixes by a minimization algorithm, resulting in smaller composition *WFSTs*. Pushing the weights allows an even earlier use of the language model, allowing what is sometimes called "language model look-ahead".

To obtain an "on the fly" implementation of the pushing of the output labels, we can modify the algorithm 1 so that the grammar label is outputed as soon as an edge of the lexicon matches with only one of the arcs leaving the grammar state. In this way, the output label is outputed as soon as possible. The $firstSingleMatch$ function detects this event by essentially checking if the output label is the only one reachable from the destination state and if the origin state has other choices. This function uses the $match(q^g, q^l)$ function that, given a lexicon state and a language model state, returns the intersection of the set of output labels that can be reached from the lexicon state $q^l$, with the set of input labels of the edges that leave the state $q^g$ of the language model. This function is itself implemented using $destSet$.

Our "on the fly" implementation of weight pushing only spreads the language model weights. Our purpose is to spread the weights of the language model throughout the path from the initial state of the word until its identity is known. For the rest of the path the weights should be $\bar{1}$.

To help us keeping track of the pushed weights, we keep a table $p$ of weights indexed with the states of the composition transducer. Every value $p[q]$ in the table is the accumulated partial weight that was pushed from the start of the pronunciation (a state in the form $(i^l, q^g)$) to the indexed state $q$.

Given an edge $a$ from $(q^l, q^g)$ to $d^{lg}$, and the set of edges of the language model that match the lexicon, then its weight $w$ is determinated as $w \leftarrow (p[(q^l, q^g)]^{-1} \otimes f(S)]$. The accumulated partial weight of the destination is set to $p[d^{lg}] \leftarrow f(S)$. Due to the characteristics of the lexicon, specially that every path must output one label, and because we are only pushing inside the word, multiple incoming edges do not push different weights into the state. The function $f$ is typically the additive operation $\oplus$ of the semiring that we are using.

```
input: $q = (q^l, q^g)$ {$q$ is a state of $lg$}
$A \leftarrow \{\}$
for all $a^l = (q^l, l^l, o^l, d^l, w^l)$ in the lexicon do
  $s^l \leftarrow destSet(a^l)$
  if $(s^l = \{\})$ then
    $A \leftarrow A \overset{\oplus}{\cup} \{(q, l^l, \epsilon, (d^l, q^g), w^l)\}$
  else
    for all $a^g = (q^g, l^g, o^g, d^g, w^g)$ in the grammar do
      if $(l^g = \epsilon)$ then
        if $(q^l = i^l)$ then
          $A \leftarrow A \overset{\oplus}{\cup} \{(q, \epsilon, \epsilon, (q^l, d^g), w^g)\}$
        end if
      else if $(l^g \in s^l)$ then
        if $o^l = \epsilon$ then
          $d \leftarrow (d^l, q^g)$
          $w \leftarrow f(match(q^g, d^l))$
        else
          $d \leftarrow (d^l, d^g)$
          $w \leftarrow w^g$
        end if
        $p[d] \leftarrow w$
        $w \leftarrow p[q]^{-1} \otimes w \otimes w^l$
        if $(firstSingleMatch(a^l, a^g))$ then
          $A \leftarrow A \overset{\oplus}{\cup} \{(q, l^l, o^g, d, w)\}$
        else
          $A \leftarrow A \overset{\oplus}{\cup} \{(q, l^l, \epsilon, d, w)\}$
        end if
      end if
    end for
  end if
end for
return $A$
```

Figure 5: Version of the algorithm with pushing of weights and output labels.

## 4. Implementation issues

The algorithms shown were described for clarity and simplicity, and, as shown, have a quadratic or larger complexity. The simplest algorithm for determining the set of edges leaving a state requires at least $n^l \times n^g$ operations, where $n^l$ and $n^g$ are, respectively, the number of edges leaving its lexicon and language model states. By sorting the edges leaving a state of the language model by input label, more efficient versions can be implemented. In particular, we propose a representation of the sets as ranges which allows an efficient implementation based on the linear complexity algorithm for the merging of two sorted lists.

The sets used to label the output epsilon edges of the lexicon can be easily calculated off-line by propagating the non-epsilon output labels backwards throughout the lexicon graph, and accumulating them in the epsilon edges. This approach can be used with medium to large lexica, but, when their size is very big, the memory occupied can be prohibitive. We propose a very compact representation of sets that performs very well in practice for the lexica commonly used in speech recognition.

If we constraint the lexicon to be loop-free, which is reasonable, then we have a finite number of paths from the initial state to the first non-epsilon-output edge. We propose a depth-first traversal of the graph from the initial state to the first non-epsilon-output edge, and numerating each non-epsilon output label sequentially. If there is more than a path to a label, then it

| *LM* | | | *algorithm* | | | *minimization* | |
|---|---|---|---|---|---|---|---|
| *cutoffs* | *edges* | *states* | *edges* | *states* | *mem (MB)* | *edges* | *states* |
| 150 | 89,731 | 14,992 | 286,890 | 204,576 | 28 | 193,971 | 112,625 |
| 100 | 121,179 | 21,408 | 404,923 | 294,544 | 33 | 260,916 | 152,306 |
| 50 | 211,634 | 38,483 | 747,832 | 554,858 | 49 | 452,947 | 263,889 |
| 10 | 817,183 | 128,386 | 3,127,618 | 2,357,306 | 108 | 1,687,611 | 938,934 |

Table 1: Experiments with our algorithm.

will be assigned multiple numbers.

By doing this, we can represent each set as a "usually small" number of numeric ranges. In fact, if that zone of the lexicon graph is shaped like a tree, then each set can be represented by a single range.

Of course, there are pathological *WFSTs* where this representation is of no help. But for the typical lexica used in speech recognition, with linear pronunciations and a small number of pronunciation alternatives per word, this scheme works very well.

## 5. Experimental results

In order to evaluate the proposed algorithm we conducted a series of experiments where we combined a lexicon with trigram language models of various sizes. The experiments were performed in parallel using both the general determinization algorithm and our alternative. A standard 600MHz pentium III pc with 1GB of ram, running Linux, was used for these experiments.

We used an European Portuguese lexicon with 27k words. The lexicon was converted to a linear lexicon and disambiguating labels were added to the end of the pronunciations, as required by the AT&T algorithm. This *WFST* had 281,815 states and 313,962 edges. It was then determinized and minimized for use with our algorithm resulting in an equivalent transducer with 29,620 states and 59,366 edges.

We used various trigram backoff language models, trained from 46 million words from the online edition of the PÚBLICO newspaper, corresponding to the years from 1995 to 1998. The generation of language model of various sizes was done by applying different cutoffs (we applied the same cutoff to bigrams and trigrams). The language models were approximated by *WFSTs* by representing each context by a different state and each n-gram in the model by an edge between contexts. The backoffs were represented by epsilon edges from specific contexts to more general ones.

The *WFSTs* obtained with both algorithms were minimized using an offline version of the transducer minimization algorithm [5]. The output of our two proposed algorithms becomes identical after this minimization, and thus we only show one set of results.

Table 1 shows the memory requirements and the size of the composition transducers obtained with our algorithm. The ratio between the number of arcs in the language model and in the composition (2.1) is similar to the one reported in [2]. The memory required by our algorithm is basically the memory necessary for the storage of the composition transducer. In this aspect, we observed differences of orders of magnitude relative to the AT&T algorithm. With our computer, the larger language model we managed to apply it to had a cuttoff of 200, and it required over 950MB of ram. Our algorithm required less than 26MB in the same task.

## 6. Conclusions

We have shown an algorithm for the integration of the lexicon with the language model with very small memory requirements. We have also shown a way to perform an "on the fly" implementation of pushing that allows the early use of the language model information, and that we plan to use to develop an approximation to "on the fly" minimization.

This algorithm can also be applied to other similar tasks in speech recognitions, such as the integration of the acoustic models (HMMs) in the search graph. This applicability is mostly dependent on the architecture of the HMMs. In the common case of linear architectures, it can be applied almost directly, and we only have to deal with self-loops as a special case.

The idea of associating the set of future labels to output epsilon transitions is also of interest in itself in order to reduce the number of dead-ends in the transducer composition algorithm.

## 7. Acknowledgements

The present work is part of Diamantino Caseiro's PhD thesis, initially sponsored by a FCT scholarship (PRAXIS XXI/BD/15836/98). This works was also partially funded by the POSI program of the "Quadro Comunitario de Apoio III".

## 8. References

[1] M. Federico, M. Cettolo, M. Brugnara, G. F. and Antoniol, ”Language modelling for efficient beam-search”. Computer Speech and Language, 9, 1995.

[2] M. Mohri, M.Riley, D. Hindle, A. Ljolje, F. Pereira, ”Full expansion of context-dependent networks in large vocabulary speech recognition”, Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'98), Seattle, Washington, 1998.

[3] M. Mohri, F. Pereira, M. Riley, ”Weighted Automata in Text and Speech Processing”, Proc. of the ECAI 96 Workshop, 1996.

[4] G. Riccardi, E. Bocchieri, R. Pieraccini, ”Non Deterministic Stochastic Language Models for Speech Recognition”, Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'95), 1995.

[5] M. Mohri, ”Finite-State Transducers in Language and Speech Processing”, Computational Linguistics, 23:2, 1997.