Finite-State Methods in Automatic Speech Recognition

Diamantino António Caseiro
(Mestre)

Dissertação para obtenção do Grau de Doutor em
Engenharia Informática e Computadores

Orientador: Doutora Isabel Maria Martins Trancoso

Presidente: Reitor da Universidade Técnica de Lisboa
Vogais: Doutor Francisco Casacuberta Nolla
Doutor Nuno Manuel de Carvalho Ferreira Guimarães
Doutora Isabel Maria Martins Trancoso
Doutor Arlindo ManuelLimede de Oliveira
Doutor Nuno João Neves Mamede
Doutor João Paulo da Silva Neto

Dezembro de 2003
RESUMO

Nesta tese são propostas duas abordagens para resolver problemas de escalabilidade e adaptabilidade em reconhecimento de fala contínua de vocabulário extenso baseado em transdutores ponderados de estado finito (TPEF). A primeira é baseada num algoritmo de reconhecimento que separa o modelo de linguagem das outras fontes de conhecimento. A segunda abordagem é baseada num algoritmo de composição de transdutores especializado. O algoritmo constrói incrementalmente um transdutor determinístico representando a composição do léxico com o modelo de linguagem enquanto aproxima outras optimizações. Sendo incremental, o algoritmo pode ser incorporado em sistemas dinâmicos de reconhecimento de fala.

Ambas as abordagens foram testadas em sistemas de reconhecimento de fala de vocabulário extenso. A utilização da segunda abordagem num sistema de transcrição de notícias televisivas permitiu uma melhoria de 6 vezes na sua velocidade de reconhecimento.

Nesta tese também foram exploradas várias técnicas de modelação usando TPEF, que foram aplicadas a dois problemas em particular: o alinhamento temporal de corpora de grandes dimensões, quer ao nível da palavra, quer ao nível do fone, usando regras fonológicas para modelar variações de pronúncia. O segundo problema foi a conversão grafema-para-fone usando técnicas diversas: baseadas em conhecimento, dirigidas pelos dados e mistas.
ABSTRACT

This thesis proposes two approaches to address scalability and adaptability problems in weighted finite-state transducer (WFST) approaches to large vocabulary continuous speech recognition. The first one relies on a recognition algorithm which decouples the language model from the other knowledge sources. The second approach is based on a specialized composition algorithm. This algorithm incrementally builds a sequential weighted finite-state transducer representing the composition of the lexicon with the language model, while approximating other optimizations. Being incremental, the algorithm can be embedded in a dynamic speech recognition system.

Both approaches were tested in large vocabulary speech recognition systems. The second one, in particular, was used in a large broadcast news transcription system. A recognition speed improvement of 6 times was observed relative to a previous non-WFST system.

In this thesis various WFST modelling approaches were also pursued. These techniques were applied to two problems in particular: alignment of large speech corpora at both word and phone levels, using phonological rules to model pronunciation variation, and grapheme-to-phone conversion using knowledge-based, data-driven and hybrid approaches.
PALAVRAS CHAVE

Reconhecimento Automático da Fala
Métodos de Estado Finito
Transdutores Ponderados de Estado Finito
KEY WORDS

Automatic Speech Recognition
Finite-State Methods
Weighted Finite-State Transducers
ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to those who have helped to develop this thesis.

I am most grateful to my advisor Prof. Isabel Trancoso with whom it has been both an honor and a pleasure to work. Her open-minded and flexible approach allowed me to pursue my own paths as a graduate student. I thank her for being an outstanding advisor - insightful, helpful, kind and patient.

I would also like to thank the funding agencies which directly and indirectly supported this work. This PhD thesis was initially sponsored by a FCT scholarship (PRAXIS XXI/BD/15836/98). The early part of this work was integrated in the REC project, while more recent developments were done in the framework of the ALERT project. This work was also partially funded by the POSI program of the “Quadro Comunitario de Apoio III”.

Next I thank the many researchers in the L2F Laboratory with whom I had the privilege to collaborate. Their team work allowed me to focus on the theme of the thesis without having to spend excessive effort on accessory, but necessary, activities. The list of researchers with whom I collaborated formally is long, and includes, in alphabetical order: Rui Amaral, Manuela Barros, Pedro Carvalho, Fernando Corte-Real, Isabel Mascarenhas, Hugo Meinedo, João Paulo Neto, Luís Caldas de Oliveira, Sérgio Paulo, Carlos Ribeiro, António Serralheiro, Isabel Trancoso and Céu Viana. I would also like to thank our Mirandese informants: António Alves, Matilde Miguel, and Domingos Raposo.

I thank Dr. Michael Riley and Dr. Mehryar Mohri for many discussions related
with finite-state transducers and search in speech recognition during my internship at AT&T.

I would like to thank the friends that helped me proof read this manuscript: Isabel Trancoso, Gail Rosen, and Isabel Mascarenhas.

I would also like to thank all members of the L2F Laboratory, which provided for a very friendly and stimulating work environment. In particular the graduate students, and other researchers, which were unfortunate enough to share office space with me.

I express much gratitude and love for my family; my father, mother and brother. They always supported and encouraged my option to pursue an academic career.

I would specially like to thank my special friend Gail Rosen, for her love and support.
# TABLE OF CONTENTS

RESUMO

ABSTRACT

PALAVRAS CHAVE

KEY WORDS

ACKNOWLEDGEMENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 Introduction

1.1 Introduction .............................................. 1
1.2 Motivation .............................................. 2
1.3 Thesis Contributions .................................... 4
1.4 Outline ............................................... 5

CHAPTER 2 Finite-State Machines

2.1 Finite-State Automata ..................................... 7
2.1.1 Definition .......................................... 7
2.1.2 Properties and Definitions .......................... 9
2.2 Finite-State Transducers ................................ 12
2.2.1 Properties and Definitions .......................... 14
2.2.2 Sequential and Subsequential Transducers ........... 14
2.3 Weighted Finite-State Automata and Transducers ........ 15
2.3.1 Semi-rings and Weights ............................ 15
2.3.2 Weighted Finite-State Automata .................... 16
2.3.3 Weighted Finite-State Transducers ................. 17
2.4 Summary ................................................ 18
CHAPTER 6 Decoders for WFST Structured Search Spaces

6.1 Traditional Search Approaches .................................................. 72
6.2 Decoupled Decoder ................................................................. 81
  6.2.1 Time-Synchronous Search .................................................. 82
  6.2.2 Stack Decoding ............................................................... 83
  6.2.3 Pruning ...................................................................... 84
  6.2.4 Experiments ................................................................. 86
6.3 Tightly Coupled Decoder ......................................................... 90
  6.3.1 Pruning ................................................................. 90
  6.3.2 Time Markers .............................................................. 91
  6.3.3 Experiments ............................................................... 93
6.4 Summary ................................................................. 93

CHAPTER 7 Integrating the Lexicon with the Language Model

7.1 Introduction ................................................................. 95
7.2 Basic Idea ................................................................. 96
7.3 Composition of the Lexicon with the Language Model ............ 98
7.4 Pushing ................................................................. 104
  7.4.1 Approximations and restrictions ...................................... 109
7.5 Minimization .............................................................. 111
  7.5.1 Minimization of the Language Model ................................ 111
  7.5.2 Suffix Sharing ............................................................ 112
  7.5.3 Tail Sharing ............................................................ 118
7.6 Optimizing the Algorithm .................................................... 121
7.7 Network Factorization ....................................................... 129
  7.7.1 “On-the-Fly” Factorization .............................................. 130
7.8 Experimental Results ........................................................ 131
  7.8.1 Comparison with the explicit determinization approach ... 131
  7.8.2 Evaluating the relative effect of optimizations ................. 134
  7.8.3 Overhead of “On-the-fly” Generation of the Search Space ... 136
  7.8.4 Impact of the Size of Language Models ............................ 138
  7.8.5 Application in a Very Large Task ..................................... 139
7.9 Summary ................................................................. 140
LIST OF TABLES

3.1 Sample TBL rule templates. ................................. 33
4.1 Construction of the inverse context dependency transducer. . 49
6.1 5k LVCSR results using a lexicon loop WFST. ........ 87
6.2 27k LVCSR results using a lexicon loop WFST. ........ 89
7.1 Toy lexicon. ................................................ 96
7.2 State translation table X for the example of Figure 7.30. . 126
7.3 Dimension of search networks. ............................ 133
8.1 Alignment results. ......................................... 152
8.2 Comparison of the current and previous rule-based approaches. 159
8.3 Pair n-gram WFSTs. ....................................... 161
8.4 Performance of the n-gram approach. ...................... 161
8.5 Performance of the n-gram approach, when trained and used after
the stress marking WFST. ..................................... 162
8.6 Performance of the backward n-grams. ..................... 163
8.7 Performance of combining the forward and reverse n-grams. . 163
8.8 Additional SAMPA symbols for Mirandese. ............... 166
LIST OF FIGURES

2.1 Example of a finite-state automaton. ........................................... 9
2.2 Example of a deterministic minimum finite-state automaton. ..... 12
2.3 Example of a finite-state transducer. ............................................ 13
2.4 Example of a deterministic minimum finite-state automaton. ..... 18
3.1 Example of a string-edit distance filter $T$. ................................. 27
3.2 Example of a string-edit distance composition $WFST$. ............... 28
4.1 Example of a context dependency transducer (for two phones: $x$ and $y$). ................................................................. 50
4.2 Finite-state approximation to a tri-gram backoff language model. . 52
5.1 Acoustic model combining several MLPs trained on different feature sets (reproduced from [92]). ........................................ 65
5.2 Phone $S$ topology. ................................................................. 69
6.1 Sample European Portuguese lexicon tree. ................................. 74
6.2 Example of word-conditioned tree search. ................................. 76
6.3 Example of word-conditioned static search network. .................... 78
6.4 Unigram lookahead vs no lookahead in the 5k words large vocabulary task. ................................................................. 88
7.1 Sample Lexicon $L$. ................................................................. 97
7.2 Sample Language Model $G$. ..................................................... 97
7.3 Composition $L \circ G$. ........................................... 97
7.4 All paths generated by the composition $L \circ G$. ................. 97
7.5 Tagged lexicon loop $L^* = \min(L)^*$. .......................... 99
7.6 Composition $L^* \circ G$. ........................................... 100
7.7 Function to explicitly copy WFSTs. ............................... 102
7.8 Definition of INITIAL and FINAL functions for “on-the-fly” composition. 102
7.9 Non-coaccessible paths avoiding algorithm. ......................... 103
7.10 Algorithm for creating edges with weight and output label pushing. 107
7.11 Function for fixing prefix edges. ................................. 108
7.12 Sample language model fragment $G_2$. .......................... 112
7.13 Minimal deterministic lexicon $L_{det} = \min \det(L)$. ............ 112
7.14 Composition fragment. ........................................... 113
7.15 Desired composition fragment. ................................... 113
7.16 Suffix sharing lexicon WFST $L_{suf}$. ............................ 114
7.17 $L_{comp} = \pi_1(L_{det}) \circ L_{suf}$. ................................. 115
7.18 Suffix sharing algorithm. ....................................... 116
7.19 Suffix sharing algorithm (cont.) .............................. 117
7.20 Function FINDINSUFFIXLEX. .................................. 117
7.21 Sample language model $G_3$. ................................. 119
7.22 Composition $L^* \circ G_3$ with pushing. ......................... 119
7.23 Desired composition $L^* \circ G_3$, with complete sharing of suffixes. . . 119
7.24 Tail-sharing composition $L^* \circ G_3$. ................................. 120
7.25 Efficient generation of edges for $\epsilon$-free composition. .......... 122
7.26 Use of ranges to tag tree lexicon. ........................................ 123
7.27 Before preprocessing. ........................................................ 124
7.28 After preprocessing. ........................................................... 124
7.29 Problematic lexicon for non-idempotent semi-rings. ............... 125
7.30 Illustration of creating new states to fix problematic lexica. ....... 125
7.31 Algorithm for creating edges with weight and output label pushing. 127
7.32 Algorithm for creating edges with weight and output label pushing. 128
7.33 Function for fixing edges. .................................................... 129
7.34 Explicit determinization vs suffix sharing algorithm. ............... 133
7.35 Performance of various minimization approximations. ............. 135
7.36 Average number of active states per frame for various minimization approximations. ........................................................... 135
7.37 Impact of pushing approximations in a small language model. ... 137
7.38 Impact of pushing approximations in a medium language model. .. 137
7.39 “On-the-Fly” vs. Static network. ........................................... 138
7.40 Variation of “On-the-Fly” overhead with the WER. ................. 139
7.41 Performance variation with the increase of the language model. ... 140
8.1 Example of an optional rule specification. .............................. 146
8.2 Example of forbidden rule specification. .................................. 147
CHAPTER 1

Introduction

1.1 Introduction

Large vocabulary continuous speech recognition is a very hard task, which is the reason why state of the art systems use multiple sources of linguistic knowledge to better solve the problem. Those knowledge sources refer to various modelling levels, such as the acoustic, phonetic, lexical and syntactic levels. Due to the diversity of knowledge sources which must be integrated, algorithms for recognizing large vocabulary continuous speech are traditionally very complex. The integration of new knowledge sources can be problematic, since each new knowledge source adds to the complexity of the recognizing algorithm. Frequently, new, or out of the ordinary knowledge sources are not tightly integrated in the main algorithm, but are rather used in later stages of processing to help choose the best sentence, from alternatives provided by the main algorithm.

At the AT&T Laboratories, an approach was pioneered which greatly simplified the integration of knowledge sources in speech recognition. The approach is based on the notion of weighted finite-state transducers (WFSTs). A WFST is a finite-state machine which allows the modelling of weighted relations. The key to the approach is the use of WFSTs to model each knowledge source. These multiple
WFSTs can then be combined using very general algorithms into a single WFST, which can then be used by a speech recognition algorithm.

This approach simplifies the integration of new knowledge sources, since the recognition algorithm is now independent of the particular knowledge sources used. The problem of integrating those knowledge sources shifted from the implementation of the recognition algorithm, to the simpler problem of finding suitable WFST representations.

An additional advantage of the approach is that very effective optimizing algorithms can be applied to the WFST resulting from the integration of the knowledge sources. Hence, the approach allows the flexible development of very efficient systems, performing on par, or even surpassing traditional systems.

The research presented in this thesis was done on the general framework of WFST techniques. It addresses some problems found by us, and other researchers, while building a WFST-based speech recognition system.

1.2 Motivation

The original theme of the thesis was the integration of machine translation techniques in a dictation system, with the purpose of improving the recognition accuracy. After a first review of the literature, in search for possible approaches to the problem, WFST techniques seemed a very promising and elegant approach to the problem, since they can be used to model both natural language processing aspects of machine translation, and speech recognition knowledge sources. We decided to proceed along that path. Our original research plan was first to build a WFST-based dictation system, and then gradually introduce machine translation components into the system.

The thesis, in its final form, grew out of problems encountered in the construction of the WFST-based dictation system. Those problems were deemed important
on its own to change the theme of the thesis. The change of theme was proposed to the Phd Coordination Committee ("Comissão de Coordenação do Programa de Doutoramento") nominated by the Department of Informatics ("Departamento de Informática"). The committee approved the change, and suggested focusing the work on the study of search problems in speech recognition.

We encountered three main problems with the use of WFST techniques in speech recognition:

- The first problem is that a fundamental WFST optimization algorithm used in large vocabulary speech recognition, WFST determinization, requires a large quantity of memory relative to the size of the WFST being optimized. This problem imposes a limit in the size (and consequently, on the quality) of the systems which can be built. In the future, with the availability of ever more powerful hardware, this problem may lose relevance, since it only affects the development environment, nevertheless it was a major obstacle in our adoption of WFST techniques.

- The second problem, also memory related, is that the WFST resulting from the compilation of all knowledge sources, even after extensive optimization, can be very large, thus requiring large amounts of memory. We consider this to be a more serious problem than the previous one. Specially if our long term goal is the ubiquity of speech interfaces in computers and electronic devices; the speech recognition component of the interface should not be substantially more expensive (in monetary and computationally terms) than a keyboard.

- The third problem is related to the use of dynamic knowledge sources: In general, WFST techniques rely on using general algorithms for optimizing one WFST network representing all knowledge sources in the system. This optimization is computationally expensive and is done offline. This means that the original knowledge sources are not available in runtime. Hence, it
may be troublesome to preserve the optimality of the network, when dynamically adjusting the knowledge sources. For example, when adapting the language model probabilities or when adding new words or pronunciations to the vocabulary.

Besides those problems, in this thesis we also used the expressiveness of WFSTs to address some research problems related with modelling the European Portuguese language. Those problems are: 1) How to explicitly model pronunciation rules, and which rules should be modelled explicitly; 2) Model grapheme-to-phone conversion.

1.3 Thesis Contributions

The major contributions of this thesis are:

- The decoupled decoder presented in Chapter 6, which was our first approach to the memory related problems. This decoder separates the language model from the other knowledge sources represented by WFSTs.

- The specialized composition algorithms, presented in Chapter 7, which performs the composition of the lexicon with the language model, both represented as WFSTs. These algorithms allow the “on-the-fly” construction of the decoder search space; furthermore, they allow the exact generation of a deterministic composition WFST and approximate the pushing and minimization operations.

Besides these contributions, which are directly related with the main goals of the thesis, other contributions were also obtained:

- The Mirandese grapheme-to-phone conversion system presented in Chapter 8, which is the first such system for that language.
• The hybrid grapheme-to-phone conversion system, also presented in Chapter 8, which combines a data-driven technique with knowledge based stress modelling.

• The time marker mechanism presented in Chapter 6, which allows the segmentation of the utterance to be specified in the search space WFST, and that was used in various segmentation tasks described in Chapter 8.

1.4 Outline

This thesis is organized into 9 chapters. Following is a brief description of each remaining chapter:

• Chapter 2: Finite-State Machines

  This chapter provides the basic mathematical background about finite-state automata, transducers and their weighted counterparts. It also introduces the notation used throughout the thesis.

• Chapter 3: FSTs in Written Language Processing

  This chapter provides an overview of the application of finite-state techniques, and transducers in particular, in written language processing. The coverage is meant as an illustration of finite-state techniques used in various areas of written language processing, some of which are relevant for spoken language processing.

• Chapter 4: FSTs in Spoken Language Processing

  This chapter provides an overview of the application of finite-state transducers to spoken language processing. The coverage is merely illustrative in the area of speech synthesis, and more exhaustive in the area of speech recognition.
• Chapter 5: Linguistic Resources

This chapter describes the linguistic resources necessary to accomplish and test the algorithms proposed in the thesis. Those resources include corpora, lexica, statistical models and various development tools.

• Chapter 6: Decoders for WFST Structured Search Spaces

The 6th chapter presents two decoders which were developed in the context of this thesis. The first one is the core of our first approach to the memory problems addressed in the thesis. The second one relies on the algorithms in Chapter 7 to solve the problems identified in Section 1.2.

• Chapter 7: Integration of the Lexicon with the Language Model

This chapter addresses the problem of optimizing the composition of the lexicon with the language model, the major topic of this thesis, and presents various algorithms which perform this composition, while optimizing the resulting network for use in speech recognition.

• Chapter 8: Other Applications

This chapter addresses other applications of WFSTs to speech processing. The first one is the problem of aligning large speech corpora at the word and phone level. This application motivated us to develop alignment support in our decoder, and to develop a phonologic rule specification language. The other application is the use of WFSTs to implement grapheme-to-phone conversion modules, useful not only for speech synthesis but also for speech recognition.

• Chapter 9: Conclusions and Future Directions

This chapter is a summary of the contributions and major finding of the thesis. It concludes with a discussion on possible future work.
CHAPTER 2

Finite-State Machines

In this chapter we provide the basic mathematical background about finite-state machines and weighted transducers in particular.

This chapter serves the dual purpose of introducing the notation used throughout this thesis, and of describing the main properties of finite-state machines.

It is not our purpose to present a self contained course on finite-state methods. For further information about finite-state automata we refer the reader to [61]; for information about finite-state transducers, including weighted finite-state transducers, we refer the reader to [97].

We start this chapter with a description of finite-state automata, then proceed to finite-state transducers. Finally, in Section 2.3, the weighted counterparts of finite-state automata and transducers are presented.

2.1 Finite-State Automata

2.1.1 Definition

Formally, a finite-state automaton (FSA) is a tuple \( A = (\Sigma, Q, i, F, E) \) where:

- \( \Sigma \) is the set of input symbols or labels. We refer to this set as the \textit{alphabet},
- \( Q \) is the \textit{set of states},
• $i \in Q$ is the initial state,

• $F \subseteq Q$ is the set of final states,

• $E$ is the finite set of edges $Q \times (\Sigma \cup \{\epsilon\}) \times Q$, each edge $e = (q, l, d)$ in $E$, is characterized by an initial state $q$, an input label $l$, and a destination state $d$.

The definition of automaton allows for $\epsilon$ edges. These edges allow the transition between states without consuming an input symbol (in speech recognition these edges are sometimes called skip arcs).

In the following presentation, we will sometimes refer to the set of edges leaving a state in the form of a transition function $\delta$ mapping $Q \times \Sigma$ to $2^Q$:

$$\forall (q_0, a) \in Q \times \Sigma, \delta(q, a) = \{q' : (q, a, q') \in E\}.$$

We will denote by $\delta^*$ the extension of $\delta$ to strings of labels $(Q \times \Sigma^*)$. $\Sigma^*$ is the set of finite strings obtained by concatenating elements of $\Sigma$, being also called the free monoid of $\Sigma$.

A path $\pi$ in $A$ from $q_0 \in Q$ to $q_m$ is a set of successive transitions between $q_0$ and $q_m$:

$$\pi = \{(q_0, a_0, q_1), (q_1, a_1, q_2), ..., (q_{m-1}, a_{m-1}, q_m)\}, \text{ with } \forall i \in [0, m-1], (q_i, a_i, q_{i+1}) \in E.$$

This definition of path will allow us to define the language accepted or recognized by the automaton:

A string $w \in \Sigma^*$ is recognized by $A$ iff $\delta^*(i, w) \cap F \neq \emptyset$. Meaning, that there exists a path from the initial to a final state labelled with $w$.

The language recognized by $A$ is the set of strings:

$$L(A) = \{w \in \Sigma^* : \delta^*(i, w) \cap F \neq \emptyset\} \quad (2.1)$$

The family of languages $L$ that can be recognized by finite-state automata is called the family of regular languages. This family is equal to the smallest family
of languages over $\Sigma$ that contains the empty set, the singleton sets, and that is closed under concatenative closure (Kleene star), concatenation, and union [83].

Finite-state automata are sometimes represented as directed labelled graphs. Figure 2.1 illustrates a finite-state automaton $A$, with initial state 0 (shown in a bold circle) and final states 3 and 4 (shown in double circles). In the figure, the $\epsilon$ symbol is represented as “..”. The alphabet of $A$ is $\Sigma = \{\text{dark}, \text{small}, \text{dog}, \text{mice}\}$, and the language it recognizes is $L(A) = \{\text{"dark dog"}, \text{"dark mice"}, \text{"dark small dog"}, \text{"dark small mice"}, \text{"dark small small dog"}, \text{"dark small small mice"}, \ldots, \text{"dark small* mice"}\}$; this language includes infinite strings in the form "dark small* dog" and "dark small* mice", were small* means any number of repetitions of the word "small".

### 2.1.2 Properties and Definitions

**Graph Properties**

In virtue of the graph-like nature of finite-state automata, they may have some properties of graphs:

- An automaton is *acyclic* if it contains no cycle, and *cyclic* otherwise. The language of an acyclic automaton is finite, since its graph has a finite number
of paths.

- A state $q$ is **accessible** if there exists at least one path from the initial state to $q$, and is **coaccessible** if there exists a path from the state to a final state. It is **useful** if it is both accessible and coaccessible.

- An automaton is **trim** if all its states are useful.

While not strictly automata algorithms, *single-source shortest path algorithms* are very useful graph manipulation tools. In this thesis, we will sometimes refer to these algorithms in the form of the function $\text{bestpath}$ and its generalization $n\text{bestpath}$. In [35], a good overview of these algorithms is presented.

These graph properties and algorithms apply to all kinds of finite-state machines presented in this chapter.

**Closure Properties**

If $L(A)$ and $L(B)$ are the regular languages defined by the automata $A$ and $B$, then it is possible to compute the following FSAs:

**Union** $A \cup B$, such that $L(A \cup B) = L(A) \cup L(B)$,

**Concatenation** $A \cdot B$, such that $L(A \cdot B) = L(A) \cdot L(B)$,

**Kleene star** $A^*$, such that $L(A^*) = L(A)^*$,

**Intersection** $A \cap B$, such that $L(A \cap B) = L(A) \cap L(B)$,

**Complementarily** $\bar{A}$, such that $L(\bar{A}) = \Sigma^* - L(A)$,

**Reverse** $\text{reverse}(A)$, such that $L(\text{reverse}(A))$ contains string that are the reverse, or mirror image, of strings in $L(A)$. For example, if "$abc" \in L(A)$ then "$cba" \in L(\text{reverse}(A))$. 

10
These closure properties of FSAs are very powerful, and allow us to think declaratively about the FSAs, in terms of set operations over their languages. Other, more expressive, formalisms available in language processing do not satisfy all of them: context-free grammars, for example, are not closed under neither general intersection nor complementarily. Intersection is very useful in practice, as it allows the use of FSAs to represent constraints, that can be incrementally combined.

Decidability Properties

Given the FSAs $A$ and $B$ and the string $w$, the following properties are decidable:

**Membership** $w \in L(A)$?

**Emptiness** $L(A) = \{\}$?

**Totality** $L(A) = \Sigma^*$?

**Subset** $L(A) \subseteq L(B)$?

**Equality** $L(A) = L(B)$?

Once again, these properties are not generally available for more expressive formalisms. For example, only the first two are decidable for context-free grammars.

Equivalence Properties

**Determinism** We say that an automaton $A$ is deterministic if, for any element $a \in \Sigma$ and $q \in Q$, there exists at most one transition labelled with $a$ leaving $q$. A deterministic automaton can be defined as the tuple $(\Sigma, Q, i, F, \delta)$, where $\Sigma, Q, i$, and $F$ are defined as before, and $\delta$ is defined as a mapping $\delta : Q \times \Sigma \rightarrow Q$ and extended to $\delta^* : Q \times \Sigma^* \rightarrow Q$. 

11
Figure 2.2: Example of a deterministic minimum finite-state automaton.

Every non-deterministic automaton is equivalent to a deterministic one. And, given a non-deterministic automaton, there exists an algorithm to find an equivalent deterministic one [4, 61].

\textbf{\epsilon-transitions} One important equivalence result is that any non-deterministic automaton with \epsilon transitions is equivalent to a non-deterministic one without \epsilon transitions.

\textbf{Minimality} Given a deterministic FSA \( A \) there are algorithms that find an equivalent FSA with the minimum number of states [61, 168]. This FSA is unique up to a renumbering of the states.

Figure 2.2 shows the deterministic minimum equivalent FSA of the example in Figure 2.1.

\section{2.2 Finite-State Transducers}

A finite-state transducer (\textit{FST}) is a model similar to a finite-state automaton, but extended to produce output strings when matching an input string. It is a tuple \( T = (\Sigma, \Delta, Q, i, F, E) \) where:

- \( \Sigma \) is the set of input labels or \textit{input alphabet},
- \( \Delta \) is the set of output labels or \textit{output alphabet},
- \( Q \) is a finite set of \textit{states},

12
Figure 2.3: Example of a finite-state transducer.

- $i \in Q$ is the initial state,

- $F \subseteq Q$ is the set of final states,

- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times Q$, is the finite set of edges. Each edge $e = (q, l, o, d)$ is now characterized by an initial state $q$, an input label $l$, an output label $o$, and a destination state $d$.

This particular definition of $FST$ corresponds to letter transducers. Some authors use definitions which allow for edges to be labelled with input and/or output strings of symbols. Those definitions can be converted to letter transducers by expanding each string-labelled edge in a path of edges labelled with single symbols or, eventually, $\epsilon$.

An $FST$ can be interpreted as an $FSA$ labelled with pairs of symbols. The underlying finite-state automaton of the $FST$ $(\Sigma_t, \Delta_t, Q, i, F, E)$ is the $FSA$ $(\Sigma_t \times \Delta_t, Q, i, F, E')$, where $(q_1, (a, b), q_2) \in E'$ iff $(q_1, a, b, q_2) \in E$.

When using this interpretation, all the properties of $FSAs$ hold for the underlying automaton of a transducer.

The real power of $FST$s is that they can be interpreted as a mapping $\tau(T)$ from the set of strings in $\Sigma^*$ to the power set of strings in $2^{\Delta^*}$. The mappings that can be represented by $FST$s are called rational transductions.

Figure 2.3 shows an example of an $FST$. In each edge, the input and output symbols are separated with ".". This transducer maps strings such as "dark dog" into "light dog", and "dark small mice" into "light big mice".
2.2.1 Properties and Definitions

Like FSAs, FSTs also have very powerful closure and algorithmic properties. Given transducers $T$ and $R$, it is possible to compute:

**Union, Concatenation, Kleene star and Reverse** which are defined similarly as for FSAs,

**Inverse** The inverse transducer $T^{-1}$ implements the inverse mapping, such that $\tau(T)^{-1}$ maps strings from $\Delta^*$ into $2^{\Sigma^*}$,

**Projection** We define the first and the second projection of an FST $T = (\Sigma, \Delta, Q, i, F, E)$ as the FSAs $\pi_1(T) = (\Sigma, Q, i, F, E_1)$ and $\pi_2(T) = (\Delta, Q, i, F, E_2)$, in which the edges in $E_1$ and $E_2$ are obtained by removing, respectively, the output and the input labels from the edges in $E$,

**Composition** The composition transducer $T \circ R$ is such that, if $\tau(T)$ maps $\Sigma^*$ into $2^\Delta^*$, and $\tau(R)$ maps $\Delta^*$ into $2^\Sigma^*$, then $\tau(T \circ R)$ maps $\Sigma^*$ into $2^\Gamma^*$. Note that in this thesis, we use a matrix-like notation for composition where $T \circ R$ means that the output of $T$ is taken as the input of $R$.

However, FSTs are not, generally, closed under intersection.

2.2.2 Sequential and Subsequential Transducers

Sequential and subsequential transducers are two very important classes of transducers.

A *sequential* transducer is a tuple $T = (\Sigma, \Delta, Q, i, F, \delta, \sigma)$ where:

- $\Sigma, \Delta, Q, i,$ and $F$, are defined as for general transducers,
- $\delta$ is the state transition function which maps $Q \times \Sigma$ to $Q$,
- $\sigma$ is the output function which maps $Q \times \Sigma$ to $\Delta^*$. 

14
The main characteristic of sequential transducers is that multiple edges with the same input label are not allowed to leave the same state. This property resembles the determinism property of FSAs. However, not all transducers can be converted to an equivalent sequential one. In fact, the expressive power of sequential transducers is inferior to general transducers. The reason is that general transducers can be ambiguous, meaning that the same input string can generate different output strings, while sequential transducers cannot output those multiple strings, since they contain only one path for each input string. Sequential transducers can only model functions, not general relations. In this thesis, the term deterministic is sometimes used to refer to sequential transducers.

Subsequential transducers are a generalization of sequential transducers, which introduce the possibility of generating an additional output string at final states [138]. Hence, subsequential transducers can model some ambiguity at final states. Formally, a subsequential transducer is a tuple $T = (\Sigma, \Delta, Q, i, F, \delta, \sigma, \rho)$, where $\rho$ is the final output function.

2.3 Weighted Finite-State Automata and Transducers

Before defining weighted finite-state automata (WFSAs) or weighted finite-state transducers (WFSTs), we start by defining the concepts of semi-ring and weight.

2.3.1 Semi-rings and Weights

A semi-ring [4] is a system $(K, \oplus, \ominus, \overline{0}, \overline{1})$, where $K$ is a set of elements, and $\oplus$ and $\ominus$ are binary operations on $K$, satisfying the following properties:

- $(K, \oplus, \overline{0})$ is a monoid, that is, it is closed under $\oplus$, $\oplus$ is associative, and $\overline{0}$ is an identity,
• \((K, \otimes, \bar{1})\) is also a monoid, and we assume that \(\bar{0}\) is an annihilator, i.e.,
  \[ a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}, \]

• \(\oplus\) is commutative,

• \(\otimes\) distributes over \(\oplus\).

Members of the semi-ring are called weights. This definition of weights allows for
the use of WFST algorithms in many applications where weights are not simple real
numbers. However, in speech processing three semi-rings are particularly useful:

The probability semi-ring \(([0, 1], +, \times, 0, 1)\) allows the interpretation of weights
as probabilities,

The log probability semi-ring \((\mathbb{R}, \oplus_l, +, \infty, 0)\), where \(\oplus_l\) is defined as \(a \oplus_l b =
-\log(e^{-a} + e^{-b})\), is the counterpart of the probability semi-ring in the
logarithmic domain,

The tropical semi-ring \((\mathbb{R}, \min, +, \infty, 0)\) is used as an efficient approximation
to the log probability semi-ring.

The logarithmic domain is used in speech processing to avoid numerical ac-
curacy problems. In this thesis, \(-\log()\) is used instead of \(\log()\) so that we can
interpret these weights as costs.

Although weights are often used to represent probabilities, weighted automata
or transducers are different from stochastic machines. In particular, the stochastic
constraint that the probabilities of all edges following a state sum to one may not
be preserved by closure operations such as union.

2.3.2 Weighted Finite-State Automata

Weighted finite-state automata are automata which have a weight associated with
each edge. Formally, \(A = (\Sigma, Q, i, F, E, \lambda, \rho, K)\), where:
• $\Sigma$, $Q$, $i$ and $F$ are defined as for FSAs,

• $K$ is a semi-ring,

• $E \subset Q \times \Sigma \times Q \times K$, is the finite-set of edges. Each edge $e = (q, l, d, w)$ is now characterized by an initial state $q$, an input label $l$, a destination state $d$, and a weight $w$,

• $\lambda$ is the initial state weight,

• $\rho : F \rightarrow K$ is the final (weight) function, which assigns a weight to each final state, in the spirit of subsequential transducers.

The weight of a path $\pi$ in the automaton, $\sigma(\pi)$, is defined using an output function $\sigma : E^* \rightarrow K$, which computes the product ($\otimes$) of all weights along the path.

A weighted automaton $A$ can be used to define a weight function $S(A)$, consisting of accumulating the weight of all paths which match a given input string. Formally, given a string $w$, $S(A)$ is defined as:

$$ S(A)(w) = \bigoplus_{\pi \in w \otimes f, \ f \in F} \lambda \otimes \sigma(\pi) \otimes \rho(f) \quad (2.2) $$

Weighted automata can be treated as transducers from strings of input symbols to weights. And, like transducers, not all properties of FSAs apply to weighted automata, for example, not all weighted automata can be determinized. In alternative, transducers can also be interpreted as weighted automata over a suitable string semi-ring.

### 2.3.3 Weighted Finite-State Transducers

Weighted finite-state transducers are transducers which have a weight associated with each edge. Formally, $T = (\Sigma, \Delta, Q, i, F, E, \lambda, \rho, K)$ where:
Figure 2.4: Example of a deterministic minimum finite-state automaton.

- $\Sigma, \Delta, Q, i, \text{ and } F$ are defined as for FSTs,

- $\lambda, \rho$ and $K$ are defined as for WFSAs,

- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times Q \times K$, is the finite-set of edges. Each edge $e = (q, l, o, d, w)$ is now characterized by an initial state $q$, an input label $l$, an output label $o$, a destination state $d$, and a weight $w$.

A WFST $T$ allows the computation of a weighted transduction $\tau(T)$. Given a string $w$, its weighted transduction $\tau(T)(w)$ is a set containing pairs associating each possible output string with its weight.

Figure 2.4 illustrates a WFST in the probability semi-ring, " / " is used to separate the weight from the other labels of the edge.

Transducer composition can be generalized to WFSTs [125]. The algorithm for finding an equivalent (sub)sequential WFST, when it exists, is described in [97]. Minimization algorithms are shown in [97] and [98].

2.4 Summary

In this chapter, we introduced the notation used throughout this thesis and presented the basic mathematical background about finite-state machines.
CHAPTER 3

FSTs in Written Language Processing

3.1 Historical Perspective

Finite-state techniques were among the first computational approaches to language processing, for example in [65], Joshi and Hopely describe a parsing program designed at the University of Pennsylvania, between the years 1958–59, as part of a project directed by Zellig S. Harris. The techniques used are remarkably similar to more “modern” approaches: the program was essentially a cascade of finite-state transducers.

But the attention of the linguistics community soon moved away from finite-state approaches, as they were deemed inadequate for the study of human language. Chomsky’s seminal work, “Syntactic Structures” [31], published in 1957, included a demonstration that natural languages could not be modelled by finite-state probabilistic devices (he called them “finite-state Markov processes”). The problem is that finite-state devices cannot represent unlimited nested dependencies that occur in natural languages. This work was so influential that, for decades, computational linguists directed their research away from either finite-state or probabilistic approaches and devoted their efforts to more powerful computational formalisms.
Therefore, finite-state technology has, for the most part, been abandoned by the linguistics community, but it thrived in other areas. From the late 1960s, early 1970s, Thompson’s regular expression compilation algorithm [154], and the grep family of UNIX tools, had an immense impact in Computer Science. In that area, soon text processing and finite-state automata became almost synonymous. During the 1970s, the foundations of speech recognition were also starting to be set at various research institutions, using, among others, probabilistic finite-state models.

Finite-state methods reentered the mainstream computational linguistics in the 1980s through the back door of morphology and phonology, and regained respectability. Many researchers started thinking that perhaps some other linguistic phenomena could also be fully understood in finite-state terms.

In the late 1980s, early 1990s, there was also a renewed interest in empirical methods, motivated by the growing availability of texts in electronic format and by the funding available from agencies such as Defense Advanced Research Projects Agency (DARPA) for both speech and natural language processing. Finite-state is, of course, not synonymous with either empirical or statistical, but the success of statistical finite-state approaches in areas such as speech recognition, no doubt, helped finite-state approaches to spread to other areas.

Nowadays, finite-state techniques are used in many areas of written language processing, either as the underlying computational model or as efficient approximations to more powerful models. In the following sections we describe some finite-state approaches in the areas of morphology and phonology, lexicon representation, spelling correction, part of speech tagging, and syntax. Our aim with this overview is not provide the complete picture of how finite-state approaches are currently being used, but instead provide a view of some of the more important techniques, specially those that have the potential of being integrated with speech processing approaches, either for synthesis, or for recognition/understanding.
3.2 Morphology and Phonology

3.2.1 Rewrite Rules

The use of rules to describe phonological transformations has been relatively common since the publication of the classic work of Chomsky and Halle, “The Sound Pattern of English” [32] in 1968. They formalized phonological grammars as ordered sequences of rewrite rules, applied sequentially to convert abstract phonological representations into surface forms. Those rules have the following syntax \( A \rightarrow B/L\_\_ R \), where \( A,B,L \) and \( R \) can be arbitrarily complex strings or feature matrices. Their meaning is that \( A \) is replaced by \( B \) when it has the left context \( L \) and the right context \( R \). Such rules are context-dependent, and are thus, more powerful than regular expressions or finite-state methods.

This phonological rule paradigm was very well accepted in the linguistic community and became the standard theory used for generation.

However, the model defined by context-dependent rules is of similar computational complexity as the context-dependent grammars used to model syntax. This similarity is contradictory with the intuition that phonological and morphological phenomena ought to be cognitively and computationally simpler than syntax or semantics.

In 1972, Johnson [63] observed that, in practice, phonologists applied the rules either left to right or right to left, never applying a rule to its own output. He showed that this constraint implied that the input/output relation specified by each rule could be modelled as a finite-state transducer. Furthermore, due to the closure property of transducer composition operation, a sequence of rules could be modelled by a single transducer. He did not present a method for converting rules into transducers, but showed that the conversion was possible. Johnson’s work was not widely known at the time, and his results only reached the mainstream computational linguistics community after being rediscovered almost ten years later.
by Kaplan and Kay [66, 67].

In the 1980s, Kaplan and Kay rediscovered and improved upon Johnson’s results. During this decade, at research institutes such as Xerox Park, much effort was applied to build the finite-state infrastructure necessary for the compilation of rules. Kaplan and Kay were successful in finding an effective compilation method; their method was described in an unpublished manuscript that circulated for years in the community as a draft, until it was finally published in 1994 [67].

Various other methods have since been proposed for the compilation of rewrite rules. For example, Karttunen [69] proposed an algebraic formulation of rewrite rules in the Xerox finite-state calculus, and a more efficient compilation algorithm was presented by Mohri and Sproat [107]. This last work also addressed the problem of compiling weighted rules into weighted finite-state transducers.

Kaplan and Kay proposed various modes of application of rules, such as left-to-right or right-to-left, each with a slightly different behavior in the presence of overlapping expressions. Many modes of application have been proposed by other researchers, for example: Karttunen’s directed replacement [70], which makes only the longest possible replacement at each point; and Kempe and Karttunen’s parallel replacement [79], which allows multiple replacements to apply simultaneously to the same input, without interfering with each other.

The compilation techniques are varied, but they usually employ a composition cascade of transducers that start by introducing auxiliary symbols to mark the occurrence of the replacement expression and/or rule contexts. Those auxiliary symbols help the application of the rule constraint by identifying the presence of the necessary context.

### 3.2.2 Two-level Morphology

In his 1983 PhD thesis [84], Koskenniemi invented a new finite-state method for describing phonological constraints. Instead of applying the rules in sequence, with
the output of one rule being used as the input to the next one, he proposed the parallel application of rules. He called his system *two-level morphology*, and the rules *two-level rules*.

A two-level rule imposes a constraint to a particular lexical/surface correspondence in a surrounding context in which it can be allowed, required, or prohibited. It can refer to the lexical, the surface, or both levels simultaneously. The relations described by two-level rules are also regular, but restricted to equal-length relations. Deletions are simulated with the use of a special *zero* symbol, that must be taken into account by other rules. The restriction to equal-length relations allows rule transducers to be combined using intersection, an operation that is not possible for more general finite-state transducers.

According to Karttunen [72], Koskenniemi became acquainted with Kaplan and Kay’s rediscovery in 1980, but he proceeded to find an alternative to generative rewrite rules, because he was not convinced that they would ever be practical for efficient morphological analysis. At a first glance, modelling generative rules using transducers would solve the analysis problem, due to their inversion property. But, if a generative rule is simply inverted, it might produce spurious analysis for a given surface form. Karttunen [72] illustrates the problem with two rules, \( N \to m/\_p \) and \( p \to m/m\_ \), that when applied to the lexical form “kaNpat” generate unambiguously “kammat”. But when the rules are inverted and applied to “kammat”, three lexical analyzes are produced: “kaNpat”, “kampat” and “kammat”. The solution to this over-analyze problem — composing the rules with the lexicon, also represented as a finite-state transducer — only appeared much later [73] in 1992, in the context of two-level morphology.

In Koskenniemi’s PhD thesis, the semantic of two-level rules was well defined, but no compilation method was proposed. Koskenniemi’s original implementation of the rules did not depend on finite-state algorithms or rule compilations. In the 1980s, much of the research of Koskenniemi, Kaplan, Kay, and Karttunen related
with the compilation of rewrite rules also applied to two-level rules, with the first compiler for two-level rules appearing around 1985-87 [86, 74]. Kaplan and Kay’s 1994 paper [67] also showed how to compile two-level rules.

3.2.3 Other approaches

Besides these two classical approaches to morphology and phonology, various others have been proposed and presented in finite-state terms; for instance, one-level morphology/phonology [14, 165, 166, 167], optimality theory [48, 51, 47, 71, 53, 7], and multi-tape approaches to morphology of Semitic languages [81, 82].

3.3 Lexicon Representation

The lexicon is a critical component in natural language systems, the nature of which differs from system to system: some systems require a large set of word forms, including inflected forms, while others require only a set of base forms and rely on morphology to obtain derived forms. Yet, almost all systems require some way to store and manipulate sets of words. Finite-state approaches are particularly well suited to represent finite sets of strings, in particular, because these sets are representable with acyclic automata, and various specialized creation and optimization algorithms exist for this kind of automata, which are much more efficient than the general algorithms. Of particular relevance are algorithms for incremental construction of minimal, deterministic acyclic FSA from lists of sorted or unsorted strings [7, 146, 93, 94, 39, 38, 130].

3.4 Spelling Correction

In this section, we will describe some finite-state approaches to spelling correction. The description of these approaches will allow us to introduce some general
techniques that can be applied to other tasks in written and spoken language processing.

The purpose of spelling correction is to find lexical alternatives for words deemed incorrect in a text. Words can be considered incorrect because either they do not belong to the language of the text, or because they are correct words but are incorrect in the context which they occur. In this section, we focus on the first problem only.

Given a misspelled word or string, a spell correction program must find in a dictionary the words that most resemble it. The resemblance or similarity can be measured in various ways, but most frequently, the resemblance measure uses common letter n-grams or the string-edit distance[41].

In one of the first approaches to spelling correction based on finite-state transducers, Oflazer [118] presented an approach to spelling correction of agglutinative languages (such as Turkish) based on two-level morphology and a dynamic programming search algorithm. The main difficulty of spelling correction of agglutinative languages is the very large number of morphemes that can be concatenated to build a word. A typical nominal or verbal root may have many thousands of valid forms.

His approach consisted of finding candidate roots of the word from a dictionary and then systematically generating all the possible words “resembling” the misspelled word. Since the morphology of Turkish is purely suffixing, the search for the root of the word only has to consider one edge of its string.

The generation modelling was done using two-level morphology, which produced a graph with edges labelled with morphemes, such that all candidate words were encoded as paths. Then dynamic programming search was done on the graph to find candidate paths. The search algorithm used the string-edit distance to reject substrings deemed too different.

The same approach was applied by Daciuk [37] to the problem of spell correction
in Polish.

In [117], Ofazer introduced the concept of error-tolerant finite state recognition. Error-tolerant recognition extends the recognition of a string by an automaton in order to allow slight deviations from strings in the regular set it encodes. As in [118], a two-level morphology system was used to produce the automaton representing all possible words. But now the search was more fine grained on the grapheme instead of the morpheme level.

In these approaches, the string-edit distance is performed as a subtask embedded in the search algorithm. Yet, the string-edit distance computation can itself be represented using weighted finite-state transducers.

In [99], the problem of string alignment using the string-edit distance is presented as an example of application of the AT&T FSM library.

Since string-edit distance is a very useful concept in natural language processing and speech processing, we will describe how it can be implemented using finite-state transducers, following the approach described in [99].

Given two strings \( a \) and \( b \), their minimum string-edit distance \( d(a, b) \) is defined by the following recurrence:

\[
\begin{align*}
    d(a[0], b[0]) &= 0 \\
    d_{\text{sub}}(a[i], b[i]) &= d(a[i - 1], b[i - 1]) + w(a[i], b[i]) \\
    d_{\text{del}}(a[i], b[i]) &= d(a[i - 1], b[i]) + w(a[i], \epsilon) \\
    d_{\text{ins}}(a[i], b[i]) &= d(a[i], b[i - 1]) + w(\epsilon, b[i]) \\
    d(a[i], b[i]) &= \min\{d_{\text{sub}}(a[i], b[i]), d_{\text{del}}(a[i], b[i]), d_{\text{ins}}(a[i], b[i])\} \tag{3.1}
\end{align*}
\]

Where \( w(a, b) \) is a matrix containing the distances between elements of the alphabet \( \Sigma \cup \{\epsilon\} \).

If \( A \) and \( B \) are identity transducers representing the strings \( a \) and \( b \), then all possible sequences of edit operations (allowing only one operation at each index
Figure 3.1: Example of a string−edit distance filter $T$.

position $i$) that transform one onto the other can be represented as the composition $A \circ T \circ B$, where $T$ is a transducer representing the distance between element. It has one state 0, and looping transitions $E_t$ as defined in equation 3.2.

$$\forall a, b \in \Sigma, (0, a, b, 0, w(a, b)) \in E_t$$
$$\forall b \in \Sigma, (0, \epsilon, b, 0, w(\epsilon, b)) \in E_t$$
$$\forall a \in \Sigma, (0, a, \epsilon, 0, w(a, \epsilon)) \in E_t$$  \hspace{1cm} (3.2)

The minimum string−edit distance can now be computed as the best path in transducer $A \circ T \circ B$.

Figure 3.1 shows a string−edit distance transducer $T$ with the alphabet $x, y$, and with substitution cost 3, and insertion and deletion costs of 5. The string−edit distance of strings $a = "xyxx"$ and $b = "xxxy"$, is the best path in the transducer shown in Figure 3.2, which results from $A \circ T \circ B$.

This computation of string−edit distance is a straightforward, but naive algorithm, and better algorithms exist (see [55] or [110] for an overview). Yet, it has a few strengths:

- The composition can be performed “on−the−fly” as needed by the graph traversal algorithm which searches for the minimum distance. When “on−
the-\textit{fly} composition is used, algorithms with better runtime complexity than $O(|A||B|)$ are possible. For example, by choosing a traversal algorithm based on Dijkstra\textquotesingle s algorithm, it is possible to obtain a complexity of $O(|A| \times (1 + K))$ for some common distance matrixes ($K$ is the distance between the strings).

- $T$ can encode contextual distances, thus allowing easy generalization to complex errors such as transpositions and one to many and many to one substitutions.

- $A$ and $B$ are not restricted to sequences: they can be arbitrary transducers, allowing an integrated formulation of the spelling correction problem.

The integrated formulation of the spelling correction problem is as follows: if $A$ is a misspelled word and $B$ is a lexicon transducer, then the word (in the lexicon) which most resembles our string is:

$$\pi_2(bestpath(A \circ T \circ B))$$  \hspace{1cm} (3.3)
If more than one alternative is required, then the single-source shortest path algorithm (*bestpath*) should be replaced by an algorithm which computes the $n$ single-source shortest paths. In positive weighted graphs, this can be accomplished by adapting Dijkstra’s algorithm [35].

The lexicon $B$ can be as simple as a list of words, or as complex as a full morphological module.

This approach can even be extended to sentence level contextual spelling correction, if $A$ is a sentence and $G$ a statistical language model (see section 4.2.2), then the best sentence level suggestion for correction is:

$$
\pi_2(*bestpath*(A \circ T \circ B^* \circ G))
$$

(3.4)

In Chapter 8 of this thesis, we present an application of this general WFST based string-edit distance approach to the problem of aligning a string of graphemes to its respective string of phones in European Portuguese. This technique allowed us, in particular, to account for some many-to-many substitutions which are characteristic of the language.

### 3.5 Part of Speech Tagging

Part of speech tagging consists of tagging the word forms of a text with their syntactic or grammatical categories. Those categories are traditionally called *Part Of Speech* (POS). Although some words belong to a single category, most can belong to multiple categories, depending on usage. The goal of part of speech tagging consists of disambiguating each word in their textual context in order to facilitate other, more complex, textual analysis.

Many approaches to POS tagging have been proposed, ranging from knowledge based techniques, to statistical or non-statistical empirical ones.

Empirical techniques have met with considerable success in this task. In the following sections, we will focus on two empirical techniques, and show how they
can be understood in finite-state terms.

3.5.1 Stochastic Taggers

Statistical techniques for POS tagging have been proposed since the 1960s. In 1965 Stolz et al. [153] were, to the best of our knowledge, the first to use some probabilistic information to help solve this problem. Their approach was based on four phases that used various sources of information: the first phase consisted of a dictionary lookup to tag common unambiguous words; the second used morphological information; the third was an ad-hoc phase that applied rules tailored for particular words; and finally, the fourth phase probabilistically selected the most likely tag given the contextual information.

The first completely statistic tagger was described by Bahl and Mercer in 1976 [10]. During the 1980s, many other stochastic approaches were published, for example [90, 52, 33, 44].

The most common stochastic approach consists of assigning the tags that maximize the formula 3.5.

\[
P(w_i|t_i)P(t_i|t_{i-1}...t_{i-k}),
\]

(3.5)

In this formula, \(w_i\) is the word at position \(i\), \(t_i\) is the tag at position \(i\) and \(k\) is the length of the context.

The formula has a direct correspondence with Hidden Markov Models (HMMs), because it can be encoded as an HMM containing one state for each different context \(t_{i-1}...t_{i-k}\), whose links connect states \(t_{i-1}...t_{i-k}\) with states \(t_{i}...t_{i-k-1}\) with probability \(P(t_i|t_{i-1}...t_{i-k})\), and whose observation probabilities are given by \(P(w_i|t_i)\).

The class of stochastic models based on formula 3.5, or on slight variations of it, are called HMM taggers, even though, when tagged text is used for training, the observations are not hidden (sometimes the term "Visible Markov Model" is used to stress that fact).
HMM taggers are attractive due to their good accuracy. Their main inconveniences include: their non-determinism that requires search for tagging, their large number of parameters and size, the difficulty of interpreting and adapting its behavior, and the need for hand-tagged training text for bootstrapping the training process.

It is straightforward to convert an HMM to a weighted finite-state transducer. One technique is to consider \( P(t_i|t_{i-1}...t_{i-k}) \) as an \( n \)-gram language model of tags, and convert it to a WFST \( A \) using the technique shown in section 4.2.2. \( P(w_i|t_i) \) can be converted to a WFST \( B \) with only one state and one looping edge for each \((word,tag)\) tuple with input label \( word \), output label \( tag \) and weight \( P(word|tag) \). Finally, the HMM WFST can be built as \( H = B \circ A \).

A radically different approach was followed by Kempe [76, 77] who tried to approximate an HMM tagger using non-weighted finite-state transducers. His goal was not to emulate the exact behavior of the HMM, but to extract a sequential transducer from it that performs tagging as accurately as possible. Such a transducer could then be composed with other transducers encoding correction rules for frequent errors, or with further steps of text analysis also encoded as transducers. In [76] he presented two approximation algorithms, which he called respectively \( n \)-Type and \( s \)-Type approximations, and managed to obtain 95.95% tagging accuracy with a transducer extracted from a 96.77% accurate HMM tagger. In [77], he introduced look-back and look-ahead in an approximation called \( b \)-Type. The transducers obtained are sequential when look-back or look-ahead or none are used. But when both look-back and look-ahead are used, they are not generally sequential, and may present various tagging alternatives, with the number of alternatives decreasing as the context size increases. In his implementation, when faced with multiple alternatives, he arbitrarily selected the first one found and managed to obtain 97.34% accuracy when approximating a 97.35% accurate HMM tagger.
3.5.2 Transformation-Based Learning

Eric Brill in his PhD thesis [21] proposed a novel method for extracting linguistic models from corpora. The technique, Transformation-Based error-driven Learning (TBL), was first proposed as an effective way to learn POS rules [20], and it has since been used in other natural language processing tasks such as spelling correction, dialogue act tagging, etc.

The TBL rule learning process starts with an initial tagging, obtained, for example, by assigning the most likely tag to each word. Then it enters a loop, where:

1. A candidate set of rules is produced.

2. The improvement score of each rule is obtained from the training corpus.

3. The best rule is selected.

4. The best rule is applied to the training corpus.

The loop iterates until the best improvement that can be obtained is below some threshold. The improvement score consists of the number of errors corrected by the rule subtracted by the number of new errors introduced.

The candidate set of rules is obtained by instantiating rule templates in the corpus. In Table 3.1, a few templates are illustrated. The rules originally referred to tags only, but in later work, Brill [22] improved the performance by adding rule templates sensitive to actual word forms.

Due to the greedy nature of the algorithm and the improvement criterion, the algorithm tended to select more general rules first.

One advantage of TBL relative to other approaches such as HMMs for POS tagging is that the rules are more easily interpretable by humans. One other attractive property of TBL derived POS rules, very relevant to our work, is that
\[
\begin{array}{|c|c|}
\hline
\text{Template} & \text{Meaning} \\
\hline
A \ B \ \text{PREVTAG} \ C & \text{Change A to B if previous tag is C} \\
A \ B \ \text{PREV1OR2TAG} \ C & \text{Change A to B if previous 1 or 2 tag is C} \\
A \ B \ \text{PREV1OR2OR3TAG} \ C & \text{Change A to B if previous 1, 2 or 3 tag is C} \\
A \ B \ \text{NEXTTAG} \ C & \text{Change A to B if next tag is C} \\
A \ B \ \text{NEXT1OR2TAG} \ C & \text{Change A to B if next 1 or 2 tag is C} \\
A \ B \ \text{SURROUNDTAG} \ C \ D & \text{Change A to B if between C and D} \\
A \ B \ \text{PREVBIGRAM} \ C \ D & \text{Change A to B if previous tags are C D} \\
A \ B \ \text{NEXTBIGRAM} \ C \ D & \text{Change A to B if next tags are C D} \\
\hline
\end{array}
\]

Table 3.1: Sample TBL rule templates.

they can be converted to deterministic finite-state transducers for very efficient tagging.

Roche and Schabes, in their 1995 paper [137], showed the conversion of TBL rules to a non-deterministic finite-state transducer and how to determinize that transducer. Their technique consists of:

1. Convert each rule to a transducer \( T \) that performs the same substitution locally.

2. Convert each transducer \( T \) to a transducer \( T_2 \) that operates globally on any string. \( T_2 \) performs all substitutions possible and keeps unmatched portions of the string untouched. They defined a \( \text{LocExt} \) operator to perform the conversion \( T_2 = \text{LocExt}(T) \).

3. Compose all the local extensions together following the rule order.

4. Determinize using a transducer determinization algorithm inspired by Mohri [96].
3.5.3 Knowledge-based Techniques

There is vast research on the use of finite-state techniques to implement knowledge-based approaches to part-of-speech tagging.

While empirical techniques tend to select one tag for each word, most knowledge-based techniques start with multiple tags for each word and use hand-crafted rules to remove (or add) tags. It is common for the end result to include some ambiguity which might only be resolved in the syntactic or semantic stages of analysis.

Among the more common finite-state knowledge-based techniques used in POS tagging are local grammars, specified using negative constraints, and intersection grammars. We will describe these techniques in deeper detail in Section 3.6, since they are also used to model syntax.

3.5.4 Hybrid Techniques

One advantage of compiling a statistically based or rule based system to the common framework of weighted finite-state transducers is the prospect of combining the strengths of both approaches. Tzoukermann and Radev did that in [160].

Their system performs disambiguation of the source text after its tokenization and morphological analysis, which are also done using weighted finite-state transducers.

The disambiguation is performed in two stages. First a linguistic disambiguation transducer is applied. It is constructed from local grammars expressing negative constraints, such as non-agreement of noun-pronoun. Secondly, statistical disambiguation is performed using a language model encoded as a weighted finite-state transducer. In order to smooth the probabilities and reduce the problem of sparse data, the language model was not based on words or tags but on genotypes – the set of tags which can be taken by a particular word.

The use of weighted transducers helps the system to cope with ambiguity. For
example, the morphological analysis transducer produces not only multiple analysis for a given input string, but also the probability of each one, thus allowing the system to prefer strings with higher frequencies.

3.6 Syntax and Parsing

Even though, as pointed by Chomsky [31], some syntactic phenomena cannot be directly modelled using finite-state devices, the use of such devices in syntax can still be very useful and efficient. Most local aspects of syntax as well as some complex ones can be described by finite-state machines. Their application ranges from local part of speech disambiguation to deeper syntactic analysis. Some systems have been used with success to perform superficial parses. The goal of a superficial parse is not to recover complete, exact parses. Instead they aim to reliably and efficiently recover syntactic information from unrestricted text, while sacrificing depth of analysis. In this section we will describe some approaches which differ in the depth of analysis performed. What they have in common is that they group segments of a text in various kinds of syntactic structures and are not limited to classifying each word in a POS class.

Most likely, the first application of finite-state techniques to parsing was taken in the Transformations and Discourse Analysis Project directed by Zellig S. Harris at the University of Pennsylvania in the years 1958 and 1959. The parser developed in that project was recently described by Joshi and Hopely in [65]. According to their description, parsing was done in 7 phases of which only the last one was not finite-state. The first 3 phases performed POS tagging and rule-based disambiguation; phase 4 computed noun phrases, phase 5 dealt with adjuncts such as adverbal and prepositional phrases, phase 6 dealt with verb clusters, and phase 7 dealt with nesting. The phases were combined with what was essentially a finite-state transducer cascade.
3.6.1 Chunks

Some of the simpler syntactic approaches extract chunks from the text. A chunk is a sequence of words which might not always correspond to established syntactic classification such as noun or propositional phrases, but which might be useful for some purposes. Bourigault [19] used this approach to identify terminological noun phrases in French. First he defined chunks as words such as verbs, prepositions, determiners, etc, that cannot belong to common noun phrases. Chunks were defined as sequences of words between chunks, and finally specific POS patterns were used to extract technical term candidates.

In [33], Church build a noun chunk recognizer based on HMMs, that was itself applied to the output of an HMM tagger and marked chunks by inserting open “[” and close “]” brackets between pairs of tags. He modelled the insertion of brackets by three states “[”, “]” and “][]”, and modelled chunks and chinks by two other states.

Abney [1] uses chunks as the basic unit to perform deeper syntactic parsing.

3.6.2 Finite-State Cascades

A finite-state cascade is a common architecture used to organize the extraction of syntactic information [2]. It consists of a sequence of levels where the inputs to one level are the outputs of the previous one, and recursion is not allowed. The number and kind of levels used varies from system to system, yet most finite-state cascade include as the first level POS disambiguation followed by a chunk assembling level. A simplex clause level is also common. Simplex clauses are clauses in which embedded clauses have been flattened and converted to siblings at the same level.

The conversion of finite-state cascades to finite-state transducers consists of compiling each level into a transducer, and composing the various levels in sequence.
3.6.3 Local Grammars

Local grammars have been championed by Gross and other lexicographers, as fundamental to the construction of large-coverage grammars capable, for example, of parsing unrestricted newspapers texts. Local grammars are descriptions of local constraints, such as restrictions on the surrounding sequences of a set of words. Finite-state automata are very convenient representations for such constraints [54].

Various finite-state techniques and algorithms were developed to create parsers for local grammars. One technique consists of representing the sequences that are locally unacceptable, or alternatively, the set of allowed sequences, using finite-state automata. Various efficient algorithms exist to apply these particular constraints [135, 95]; and they are used, for example, to implement POS disambiguation.

More powerful algorithms have also been proposed. In 1994, Roche [136] presented two parsing algorithms capable of parsing not only finite-state grammars but also context-free grammars and their extensions.

In the first algorithm, the grammar is a rational transduction \( f \), represented by a transducer \( T \). The input of the parser is the set \( s_0 = \{ [P] \text{sentence}[P] \} \) containing only the input sentence delimited by the phrase marker \([P]\). The analysis consists of computing \( s_1 = f(s_0) \) and iterating \( s_{i+1} = f(s_i) \) until a fixed point \( s_p = f(s_p) \) is found. The resulting set, \( s_p \), contains bracketed strings representing parsing trees. When the input is syntactically ambiguous, the set contains more than one element. As each one of the sets \( s_i \) is represented by a directed acyclic graph (DAG) \( A_i \), the computation consists of iterating \( A_{i+1} = A_i \circ T \).

To detect the fixed point, Roche used a modified version of the composition algorithm that verifies if the result was equal to its input automaton \( A_i \).

The second algorithm follows the same general idea but is performed deterministically thanks to the decomposition of the transducer \( T \), that is not in general sequential, into a bimachine.
A bimachine is a finite-state device that consists, basically, of a left sequential function composed with a right sequential function. Both functions can be implemented with (left or right) sequential transducers. All transducers can be decomposed into a bimachine.

Given the bimachine $B = (T_l, T_r)$ functionally equivalent to $T$, the computation performed in each iteration consists now of composing $A_i$ with the left-sequential transducer first, $A' = A_i \circ T_l$, and then with the right-sequential one, $A_{i+1} = \text{reverse}(\text{reverse}(A') \circ T_r)$.

The deterministic version is usually much more efficient because its composition only generates useful states. That does not happen with the non-sequential one. On the other hand, the size of the decomposed sequential transducers can be exponential relative to the original transducer.

Using these techniques, Roche managed to build a system with a grammar of more than 200,000 lexical rules, and capable of parsing French sentences in less than a second.

The construction of very large lexicalized grammars poses various problems in terms of management of the large number of rules. Besides compilation algorithms, properly designed authoring tools are a necessity if such rules are to be efficiently written and maintained by linguists and lexicographers. One such system is the Intex System [147]. This system is based mostly on finite-state devices and can be used for research at various linguistic levels, including morphology, part-of-speech tagging and syntactic analysis.

This system was designed as a text processor able to parse large corpora. It includes large coverage dictionaries and grammars represented using finite-state transducers. Furthermore, since it is an authoring tool, it allows the user to add and create more dictionaries and grammars. The system has been used to build large-scale linguistic resources in many languages, including English, French, Italian, German, Spanish and Portuguese.
3.6.4 Intersection Grammars

Intersection grammars [164, 68, 163] are based on Koskenniemi’s two-level rules and encode syntactic constraints using transducers. These grammars are also used in a broad range of syntactic levels, from mere POS disambiguation to partial or large scale parsing. The most distinctive feature of these grammars is that their rules are compiled to finite-state transducers which are then combined using intersection.

3.6.5 Approximation to More Powerful Formalisms

A different approach to parsing with finite-state transducers consists of using the transducers as approximations to more expressive computational formalisms. Due to their widespread use in modelling syntax, methods that approximate context-free grammars or automata equivalent devices such as pushdown automata or recursive transition networks (RTN) [172], are of particular interest.

Finite-state approximations can be obtained by imposing restrictions, such as memory limitations, on a more powerful parsing algorithm in order to restrict its recognition power to finite-state languages. Examples of approaches thus motivated are [126] which is based on a LR-parser, and [64] which is based on a left-corner parser.

In general, two kinds of approximations are described in the literature, either the finite-state approximation is a subset of the context-free grammar or a superset that overgeneralizes it in a systematic way. Overgeneralization techniques are useful when the finite-state grammar is used as a preprocessor, while subset techniques are of psycholinguistic interest.

It is important to know the cases in which a particular approximation technique achieves exact results, for example, the approach in [126] is exact for left-linear or right-linear grammars, while the method in [111] is exact in the more general case of strongly regular grammars.

This quest for finite-state approximations to context-tree grammars is a very
active research topic in the finite-state community.

3.7 Discussion

In this chapter we presented an overview of how finite-state techniques can be used, and are being used, in written language processing. They are sometimes used not as a tool to solve all problems, but as tools particularly suited for some tasks. As examples of application, finite-state methods are commonly found in the core of information extraction systems [60] because they can very efficiently extract the information required for many tasks. Even very large scale natural language processing systems, such as the SYSTRAN machine translation system [143, 144], are starting to use finite-state methods embedded at various levels, including morphology, conceptual description, and transfer dictionary encoding.

From the point of view of speech recognition, finite-state methods in natural language processing are very interesting for various reasons. One is that many of these methods work as well with an input word graph or lattice as with a single sentence, making them suitable to process recognition alternatives produced by a speech recognition component of a speech understanding system. Some methods can even be embedded in the speech recognition search space, allowing the creation of tightly integrated speech understanding systems. One final reason is that finite-state algorithms are very general, and an algorithm designed for processing at a particular linguistic level, for example syntax, can be used in speech recognition to process a different level. One example is the use of local grammars to enforce pronunciation constraints.
CHAPTER 4

FSTs in Spoken Language Processing

In this chapter we present an overview of the use of methods based on finite-state transducers in spoken language processing.

Since the main focus of the thesis is speech recognition, the overview of WFST techniques in speech synthesis is more superficial than the description of methods used in speech recognition. The speech recognition overview is much deeper and describes not only WFST techniques but introduces also some general concepts necessary for better understanding the rest of the thesis, and comparison with state-of-the-art techniques.

4.1 Text to Speech

Text to speech (TTS) synthesis entails the conversion of a written sentence or text into a spoken utterance. The problem can be subdivided in two main subproblems: an analysis subproblem that extracts linguistic information such as phoneme sequences or accentual parameters from written text, and a generation subproblem that uses the extracted information to generate a sound file.
4.1.1 Extraction of Linguistic Information

The first main subproblem can be divided in various subtasks, some of which are:

Document structure detection analyzes the global structure of the document, identifying elements such as titles, paragraphs and sentences which can have implications for prosody.

Text normalization converts symbols, abbreviations, numbers and other non-orthographic elements to a normalized transcription.

Linguistic analysis performs some syntactic and semantic analysis necessary either for pronunciation or prosody.

Homograph disambiguation disambiguates the correct sense of words, whenever it has an impact on the pronunciation.

Morphological analysis analyzes unknown words in their component morphemes in order to derive their correct pronunciation.

Letter-to-sound conversion uses dictionaries combined with rules or data driven classification methods to determine the pronunciation of words.

Prosody generation is the task of generating speech parameters such as duration, volume and F0 contours from the original text.

Looking at this list of tasks, we realize that some of them are application of techniques that were described in the previous chapter, and can be effectively modelled using finite-state transducers. In fact, some of those techniques were proposed in the context of TTS systems.

For example, linguistic analysis is normally based on POS tagging and superficial parsing. Homograph disambiguation can be done using POS tagging followed by decision trees or rules. Rule based letter-to-sound conversion is sometimes based
on the same kind of rules used for phonology, including rewriting rules and two-
level rules. Other approaches to letter-to-sound conversion such as decision trees
can also be converted to finite-state transducers [150].

A good description of some of these techniques can be found in [148] where the
AT&T multilingual speech synthesis system is described. This was indeed one of
the first systems to make extensive use of finite-state transducers.

4.1.2 Speech Generation

Among the main approaches to generating a speech sound wave from the param-
ters obtained in the previous analysis phase are:

Articulatory synthesis which uses a physical model of speech production that
includes physiological articulators.

Formant synthesis that uses a source-filter model characterized by formant fre-
quencies.

Concatenative synthesis that is based on the concatenation of pre-recorded
speech segments.

To our knowledge, the only approach to which finite-state transducers have been
applied is concatenative synthesis. In this approach, they are used to model the
search space generated by alternative concatenation hypotheses of variable length
segments.

In [173], Yi et al. proposed a transducer structure to efficiently perform unit
selection in concatenative speech synthesis. Their approach consists of searching
for the best path in the cascade $W \circ L \circ P \circ S$, where $W$ is the sequence of words, $L$
is a pronunciation lexicon, $P$ implements phonological rules, and $S$ maps phones to
sound segments. The key aspect of their approach is the structure of $S$, that allows
connections between every sound segment in the database thought a constraint
subnetwork. This subnetwork weights connections based on phonological context classes. In this approach, the concatenative cost of two segments is determined using only their phonological context and not the speech signal itself.

Bulyko and Ostendorf [25] proposed a different approach in which they defined *splicing costs*, which are segment-specific boundary costs that measure how well one segment concatenates with any other. Besides this novel cost component, their cost measure also includes the concatenation cost between segments and the distance to the target (desired) segment. Vector quantization (VQ) is used to reduce the complexity associated with computing concatenation costs. The database is also represented using a transducer, but with a very different structure: each concatenation unit is represented by a state, states representing the same target are grouped in clusters, and concatenation costs are represented with a VQ concatenation sub-network. Each segment-representing state has links to and from states in the VQ sub-network which represent its boundaries. Concatenation costs are represented in the VQ sub-network as links between these boundary-representing states. This system also includes prosody prediction [24], by combining application specific prosodic templates with a prosodic prediction module, implemented as decision trees compiled to transducers.

### 4.2 Speech Recognition

With few exceptions, almost all approaches to speech recognition are based on probabilistic finite-state models such as HMMs or n-gram language models. Yet, until the development of WFST approaches, the integration of different knowledge sources was not done in a unifying way, each particular knowledge source requiring explicit support in the code of the search algorithm (decoder) of the recognizer. This led to rigid systems, hard to modify.

WFST approaches brought to speech recognition a unified form of representa-
tion of knowledge sources. On one hand, this led to a simplification of the decoder, which became independent of the particular knowledge sources used. On the other hand, the recognizer itself became much more flexible, and from the point of view of research, the effort of trying out new ideas was much reduced. Another very important aspect is that WFST optimization algorithms, such as determinization and minimization, can be used to obtain very efficient systems, on par or surpassing traditional techniques.

We start this section by describing the probabilistic framework which supports most modern approaches to automatic speech recognition, and showing how WFSTs fit in that framework.

The presentation of WFST approaches starts with a description of how the major components of a large vocabulary speech recognition system can be represented as WFSTs. Then it proceeds to a description of how those components are integrated and how they are used by the decoder.

To illustrate the flexibility achieved with WFSTs, the following sections present various topics, including: multi-pass systems, dynamic decoders, combination of multiple systems, and pronunciation modelling.

### 4.2.1 Probabilistic Framework

In probabilistic approaches to speech recognition, the search problem is stated as the problem of finding the most likely sentence given an utterance. This consists of maximizing the following equation:

$$\hat{W} = \arg \max_W P(W|X)$$  \hspace{1cm} (4.1)

where $\hat{W}$ is the most likely sequence of words, and $X$ is the utterance. From the point of view of speech recognition, the sentence is a merely a sequence of words, and the utterance is normally represented as a sequence of feature vectors extracted
from the speech signal at regular intervals.

Using Bayes theorem, Equation 4.1 can be rewritten as:

$$\hat{W} = \arg \max_P \frac{P(W)P(X|W)}{P(X)}$$  \hspace{1cm} (4.2)

and, since the maximization is carried out with the variable $X$ fixed, the term $P(X)$ can be ignored:

$$\hat{W} = \arg \max_P P(W)P(X|W)$$  \hspace{1cm} (4.3)

Equation 4.3 is called the *fundamental equation* of speech recognition. The two terms of the equation have distinct roles and are estimated using different techniques:

- The *language model* $P(W)$ is normally modelled using $n$-gram language models.
- The *acoustic model* $P(X|W)$ is usually modelled using a suitable structured HMM network. Each path in this network models a particular realization of the word sequence.

Strictly speaking, the computation of $P(X|W)$ involves summing the probability of all paths in the network which have some probability of generating the word sequence $W$:

$$P(X|W) = \sum_{b \text{ path}} P(X|b)$$  \hspace{1cm} (4.4)

Most decoders simplify this computation by using the best path instead of the sum, as shown in Equation 4.5. This approximation is called the *Viterbi approximation*.

$$P(X|W) = \max_{b \text{ path}} P(X|b)$$  \hspace{1cm} (4.5)

Both the HMM-based acoustic model $P(X|W)$, and the language model $P(W)$ can be represented with graph-like models. In large vocabulary continuous speech
recognition (LVCSR) systems, these graphs can be very complex since they integrate various sources of linguistic knowledge. In particular, the acoustic model HMM network is usually assembled from smaller HMM models representing sub-word segments, such as phones.

In the following section, we will see that many different knowledge sources can be represented as WFSTs. The algebraic properties of WFSTs can then be used to seamlessly integrate those knowledge sources in WFST representations of both the acoustic model network and the language model. Furthermore, these two networks can be themselves integrated in a single WFST compatible with this probabilistic framework, and suitable for very efficient decoding.

4.2.2 WFST Representation of Knowledge Sources

HMM Transducer

The topology and transition matrix of HMM sub-word units can be compiled to WFSTs with a similar topology. The input label of each edge is a symbol representing the HMM distribution of the destination state. Its weight is the transition probability, and the output label is $\epsilon$, except for the edges which leave the initial state (or alternatively those which arrive at final states), which output a symbol representing the model. Every path that traverses the transducer must output one and only one symbol. A transducer $H$ containing all HMM sub-word units can be built using a concatenative closure of the union of the individual sub-word models WFSTs.

Context Dependency

In a context independent system, each HMM unit can represent a specific phone. But most state-of-the-art recognition systems are context dependent. This means that the HMM unit used to represent a phone is selected depending on the particu-
lar context of the phone (for example, in triphone systems each phone has different units for each combination of previous and next phones). Context dependent systems allow much more detailed modelling.

It is useful to distinguish two types of context dependent systems: \textit{intra-word} and \textit{cross-word}. Context dependency of the first type, in which context dependent units are used only inside the pronunciation of the word, poses no problems to a traditional hierarchical decoder, because it can be implemented with simple substitutions. One only has to replace the units inside the pronunciation with context dependent versions. Cross-word context modelling is much more difficult to implement, because now the pronunciation of a word depends on the previous and/or the next word. The use of cross-word units has a severe impact on the structure of the search space. A traditional decoder has to deal explicitly with the cross-word pronunciations, and in some cases this is only done in an additional pass.

The transformation from context independent to context dependent units can be modelled by interposing a transducer $C$ between the HMM transducer and the lexicon. This transducer implements the context dependent to independent relation\footnote{That is, a transduction from the HMM units that are context dependent to the context independent phones which constitute the lexicon.} [101]. To build the transducer $C$, we build its inverse by connecting the edges as show in Table 4.1 for the triphone case. For each triphone\footnote{$a-b+c$ should be read as the version of $b$ that has $a$ on the left and $c$ on the right.} $a-b+c$, right biphone $b+c$, left biphone $a-b$ and uniphone\footnote{We name a context independent unit \textit{uniphone} when used in a system that also contains context dependent units.} $b$, an edge is built by connecting states that represent particular contexts. In Figure 4.1, a complete context dependency transducer is illustrated. The reader may notice that the input and output labels on the edges are not synchronized; the purpose of this delay between the input phone and the output of its context-dependent version is to make the transducer deterministic without changing the represented relation.
Table 4.1: Construction of the inverse context dependency transducer.

**Pronunciation Lexicon**

We may regard a pronunciation as a transducer from phone sequences into a word. In most systems, a pronunciation is modelled as a phone sequence associated with a word (although some systems use more complex models [3]). The pronunciation can be modelled as the trivial linear transducer with one edge for each phone in the sequence. Each edge has the corresponding phone (or model) as input label and unitary weight. One of the edges in the sequence may have the pronunciation probability and must have the word as output; all other edges should have epsilon output. The linear transducer corresponding to the complete lexicon $L$ is the concatenative closure of the union of individual pronunciation transducers. In large vocabulary continuous speech recognition the lexicon is commonly organized as a tree. We can transform the linear transducer to a tree form by sharing prefixes, or we can share prefixes in an even more general way by using the generic weighted
transducer determinization algorithm [100].

Language Models

In speech-to-text systems, the last level corresponds to sentences (sequences of words), and so, the language model can be viewed as an acceptor, because no transduction between languages is required. It might not be so, in language understanding or information extraction systems where it can be a transducer to map the sentences to a higher, application dependent, level. For example, in [30] is proposed the use of transducers to map from English sentences to logic forms in a language understanding task.

In this thesis, however, we focus on the speech-to-text problem and treat the language model as an acceptor. There is a long tradition of using finite-state acceptors in speech recognition. They are used in limited domains and in dialog systems to specify the acceptable sentences. In large vocabulary applications, they are used to constrain the search in later passes of multiple pass systems. However, the main methodology used in first-pass systems consists of word $n$-gram language
models.

One simple way to convert an $n$-gram language model to an automaton $G$, is to create a state for each possible context, and place weighted edges between the contexts, each edge labelled with the respective word and weighted with the probability of the $n$-gram. This exact conversion requires a number of edges equal to the number of all possible $n$-grams, and not only to the number of $n$-grams observed in the training data. Hence, this conversion is only possible for small vocabulary and low order language models.

The alternative is to resort to finite-state approximations. One approximation consists of creating a state for each context found in the training data. This state is the origin of all the edges that model $n$-gram probabilities with that context in the language model. Backoffs to lower order contexts are implemented as an $\epsilon$ edge from the higher order context to the lower. This is an approximation because the resulting automaton is non-deterministic and can have multiple paths for the same $n$-gram, each with a different probability value. From the recognition's point of view, this can be a problem, because if the backoff path probability is higher than the explicit probability, the former will be erroneously preferred. This approximation was initially proposed in [127] for approximating bigram language models. Given that it works very well in practice, this approximation is being widely used with some variations, including [131, 161, 169, 155]. In Figure 4.2, we show an example of a finite-state approximation to a trigram backoff language model with a vocabulary of two words $a$ and $b$. Each state is labelled with the context it represents.

4.2.3 Integration of Knowledge Sources

Having represented the previous components as transducers, the problem of decoding is reduced to searching for the best path in the transducer obtained with the composition of all the components in the system. This composition possibly
Figure 4.2: Finite-state approximation to a trigram backoff language model.

includes $X \circ H \circ C \circ L \circ G$, where $H$ is the HMM transducer, $C$ is the context-dependency transducer, $L$ is the lexicon, $G$ is the language model, and $X$ is a WFST representation of the utterance. $X$ has a state for every frame $t$, with one edge for every distribution connecting consecutive states. The output label of each edge is the distribution identifier, and the weight is the probability of the feature vector at the frame given the distribution.

This integration of components is interesting from a theoretical point of view, because it makes the complete search space explicit. But is not used in practice, since it is computationally very wasteful. Specially the creation of the transducer $X$ which involves computing the probability of every vector given every distribution. The composition of $X$ with the rest of the system is best done “on-the-fly” in the decoder.

We continue with the description of how the component WFSTs are combined
in practice in most systems.

**Component WFST Composition**

In [105], the composition of the component WFSTs includes all the levels from the acoustic model to the language model, resulting in an HMM-to-word WFST \((C \circ L \circ G)\). A phone-based speech decoder constrained with this network, is then used to recognize the utterance. Later, in [103], the composition was extended to the level of the distribution \((H \circ C \circ L \circ G)\). Because the resulting network \(N\) is enormous and contains many long and linear paths\(^4\), it is factorized into two transducers \(N = H' \circ F\), where \(H'\) replaces each different linear path with a unique identification code, and \(F\) is the result of replacing the linear paths with an edge labelled with the respective code. Those paths are represented as HMMs in the decoder and are used instead of phones; the decoder is obviously constrained by the \(F\) network during recognition.

During the composition of the various component WFSTs, the operations of determinization and minimization are used to optimize and reduce the size of the intermediate transducers. The size of the resulting transducer is generally not much larger than the language model (in [103], it was respectively 2.1 and 2.5 times bigger than the bigram and trigram language models) and allows efficient use of cross-word acoustic models (they reported that the use of cross-word triphones only increases the transducer size by 2.5%).

This approach of building static optimized HMM level networks was adopted by other research groups outside of AT&T where it was developed, and is the most known WFST approach to large vocabulary speech recognition. In the rest of the thesis we will refer to it as the *explicit determinization* WFST approach.

\(^4\)In this context, a linear path is a path where its states, other than the first and the last, have a self-looping transition, one incoming transition, and one outgoing transition.

53
Composition of the lexicon with the language model

The integration of the speech recognition components, although conceptually simple, is a difficult task because of the size of the involved transducers. In particular, the determinization of the composition of the lexicon with the language model, may not terminate or may require very large amounts of memory. Since the determinization algorithm terminates when applied to a unambiguous transducer (this is a sufficient condition), the composition of the lexicon and the language model must be made unambiguous.

In the explicit determinization approach [105] the composition transducer is disambiguated by disambiguating the lexicon and language model WFSTs. The lexicon is represented by a linear transducer (with no sharing of pronunciations paths) which is disambiguated by concatenating dummy symbols to the end of ambiguous pronunciation paths. A lexicon transducer can be ambiguous due to homophones, and due to the existence of words whose pronunciations are a prefix of others, which results in ambiguity in its closure transducer. Another important aspect of the lexicon WFST structure is that the output word should be produced by the first edge of the pronunciation – this is necessary to avoid the generation of non-coaccessible states during composition.

Backoff ε edges of the language model should also be replaced by disambiguating input symbols. These symbols will turn the language model into a deterministic WFST. This is a required step, even when the determinization algorithm treats the ε symbol like an ordinary label, because it prevents the algorithm from trying to determinize suffixes of different words arriving at the same backoff state. This behavior has a large cost in terms of temporary memory and quality of the resulting composition.

The disambiguating lexicon and language model WFSTs are then composed, and the resulting transducer is determinized using the general weighted transducer determinization algorithm. All disambiguating labels should be removed from the
integrated network once other optimizations such as minimization are performed.

Very recently, it was proposed a new disambiguating algorithm which can be
applied to arbitrary WFSTs [5]. This algorithm removes the need for task specific
disambiguating heuristics. As an example, the authors applied it to a highly am-
biguous class based n-gram language model, which could then be composed with
the lexicon and determinized.

4.2.4 Multiple Pass Decoders

Multiple pass decoders use a computationally cheap first pass system to generate
lattices or n-best lists containing recognition alternatives. The information from
those alternatives is then used to perform other refined searches using adapted or
computationally expensive models. This process of reevaluating the cost of lattice
hypotheses using better knowledge sources is called rescoring.

Most traditional first pass search algorithms can be adapted to generate lattices
with little computational overhead. Some of the usual approaches [140, 139, 9] are
not compatible with WFST techniques, due to the extensive optimization of the
search space done by these techniques. Nevertheless, at least two WFST techniques
have been proposed: in [17] it was proposed recording all the search space used by
the search process and converting that state-level graph to a word lattice, while
in [89] it was proposed generating approximate phone lattices which in turn are
converted to word lattices.

One way of building second pass decoders consists of using the lattice $W$ gen-
erated in a first pass (which is an acyclic word graph, representable by WFSTs)
to constraint the search space [102]. This can be constructed, for example, as
$H_2 \circ C_2 \circ L \circ G \circ W'$, where $H_2$ and $C_2$ are more expensive acoustic and context
dependency models, and $W'$ is similar to $W$, but with weight $\bar{I}$ in every edge.

When the language model $L$ is replaced by a new one, $L_2$, it is possible to simply
replace the language model score in the original lattice: $W \circ -L \circ L_2$, where $-L$
has the same topology as \( L \) but negated weights\(^5\). Then the search for the best hypothesis can be done efficiently in the word graph itself, as \( \text{bestpath}(W \circ -LoL_2) \), instead of having to repeat the acoustic search. This technique can be used with \( n \)-gram language models [17], if backoff edges in \( L \) output a special symbol \(<\text{BO}>\) and backoff edges in \(-L\) have that symbol as input. This avoids the generation of redundant backoff paths from occurring when non-deterministic \( n \)-gram language models are composed.

### 4.2.5 Dynamic Decoders

One of the major problems of static \( WFST \) approaches is the need for large computational resources for: 1) building large \( WFST \) systems, and 2), using those large systems. Although the most common solution to this problem consists of using small static first pass decoders, combined with additional search passes, some dynamic forms of knowledge sources integration have also been proposed.

While developing the foundations of \( WFST \) approaches, Riley, Pereira, Mohri, and other researchers at AT&T proposed some dynamic techniques for component \( WFST \) integration. In [132], the use of “on-the-fly” composition was proposed as a way to integrate context-dependent phone models into a speech recognition system. In that work, a non-deterministic lexicon \( WFST \) outputting the word in the first edge of each pronunciation was used to avoid the generation of non-coaccessible paths during composition. No search space optimizations were reported. Later [106], they reported that this technique only expands 1.6% of the total number of arcs in the ATIS Task. In that publication, various local determinization algorithms were also described, which could be used to dynamically optimize the search space, but no direct application to speech recognition was reported.

In 2001, Dolfing and Hetherington [46] proposed the dynamic integration of a small optimized static system with a larger language model. This approach

---

\(^5\)We are assuming the use of log probability weights and the tropical semi-ring.
resembles resoring, but is done in a single pass. They started by factorizing a large language model $G_l$ into a small model $G_s$ and a difference model $G_{l-s}$, such that $G_f = G_s \circ G_{f-s}$. Then the small language model was statically composed with the other knowledge sources, including the lexicon $L$, phonological rules $P$, and the acoustic model $H$, in a network $N = H \circ C \circ P \circ L \circ G_s$ that was optimized offline. The full language model information was integrated in runtime with the difference language model $N \circ G_{f-s}$, using a specialized “on-the-fly” composition algorithm, which updated the language model score of hypotheses whenever they crossed a non-$\epsilon$ output edge in $N$.

In 2002, Willett and Katagiri [171] also proposed factorizing the language model as $G_{tri} = G_{uni} \circ G_{tri-uni}$, where $tri$ and $uni$ referred to trigram and unigram language models, respectively. $G_{uni}$ was used in the construction of an offline optimized search network $N = H \circ C \circ L \circ G_{uni}$. That network was then used by a decoder similar to the Single Lexicon Network approach described in Section 6.1: whenever a hypothesis crossed a non-$\epsilon$ output edge in $N$, its language model score was updated with the difference to the trigram score, and the language model context associated with the hypothesis was updated with a new state in $G_{tri-uni}$.

This last approach is similar to the approach proposed in our first decoder and described in Chapter 6. Our approach was originally published in 2000 [27], and by the time of Willett and Katagiri’s publication we had stopped developing it. We find the fact that other researchers arrived at a similar solution to the problem is an indication that the idea has merit and deserves more development; nevertheless, we believe that the “on-the-fly” composition algorithm described in Chapter 7 is a better solution.

4.2.6 Combination of Multiple Systems

The combination of the output of multiple recognition systems can improve recognition accuracy. In [50], the ROVER (Recognizer Output Voting Error Reduction)
system was presented that combined the output of various systems that participated in that year LVCSR Hub 5-E Benchmark Test Evaluation, managing an average relative error reduction of 10 to 12%. Since then, it has become common to build slightly different versions of a recognition system\(^6\), and to use similar techniques to improve the overall performance.

Outputs can also be combined using finite-state techniques, as shown in [102], where various systems outputs were combined by performing the finite-state intersection of lattices generated by each system and then finding the lowest cost path through the resulting network.

A different approach to the combination of multiple systems was taken by Hazen et al. [56], with the goal not of improving recognition accuracy in a particular domain, but rather of combining systems developed either for different languages or for different domains, in a single unified system. This allowed, for example, the recognition of utterances in multiple languages, or about various subjects. Their approach consists simply of combining various specialized integrated search networks \(N_i\) into a single network \(N\) using the union operation \(N = k_1N_1 \cup k_2N_2 \cup \ldots \cup k_iN_i\), were \(k_i\) are scaling factors necessary to account for differences in acoustic modelling, when networks from different languages are combined.

### 4.2.7 Pronunciation Modelling

Pronunciation modelling is one of the areas where the flexibility of WFSTs has been used by various researchers to address various problems with the simple "lexicon as a list of pronunciations" model which is the most common approach in speech recognition.

In [16], Boulianne et al. integrated cross-word phonology transducers in a French large vocabulary recognition system. Their goal was to explicitly model

---

\(^6\)By using, for example, different acoustic models, language models or different speaker adaptation techniques
French coarticulation phenomena such as liaison. The presented approach consisted of modelling the lexicon \( L \) as the composition of three transducers \( L = P \circ S \circ L' \), where \( P \) ("phonology") restricted realization of base form liaison into surface form phones, according to whether the first phoneme beginning the next word was a vowel. The transducer \( S \) dealt with silence between words, and \( L' \) was a lexicon where word internal phonological rules were applied and problematic words were marked with special symbols, that were taken into account by the transducers \( S \) and \( P \). With this cross-word pronunciation modelling, a reduction of error rate from 30% to 27.7% was observed in a 5k words task.

The Spoken Language Systems Group of the Massachusetts Institute of Technology has devoted much effort in the area of pronunciation modelling with the publication of various PhD theses and many papers about this subject.

In his PhD thesis [109], Mou presents an integrated framework for lexical modelling based on WFSTs. His research involved sub-lexical and supra-lexical modelling which were integrated in a speech recognition system as \( H \circ C \circ M \circ L \circ G \), where \( M \) is a sub-lexical model WFST and \( G \) includes supra-lexical modelling.

His research on sub-lexical modelling was based on the ANGIE hierarchical model [142, 87], which describes sub-word structure using context-free grammar rules. ANGIE explicitly models morphology, syllabification, phonemics and phonetics. Mou presented various finite-state approximations to the information in ANGIE, with the purpose of allowing the direct use of that information in a recognition system.

Mou also investigated various methods to model supra-lexical information with WFSTs. His research was based on the TINA natural language understanding system [141], and involved the integration of probabilistic natural language based models in speech recognition.

In a parallel effort, Hetherington [59] presented a rule system specially tailored to efficiently encode pronunciation rules for use in speech recognition. Using this
rule system, a pronunciation modelling WFST $P$ can be built and integrated explicitly in the recognition cascade: $H \circ C \circ P \circ L \circ G$.

In his PhD thesis [13], Bazzi addresses the problem of out-of-vocabulary (OOV) words in speech recognition. The main problem with these words is the number of recognition errors incurred by the system when they are present on the speech signal. Bazzi’s goal was to avoid these errors by explicitly detecting OOV words. He presents various WFST based methods to model OOV words, including the use of sub-word networks $L_{oo}v$ in parallel with the lexicon. These sub-word networks were integrated as $C \circ P \circ (L \cup L_{oo}v) \circ G$, where $P$ is Hetherington’s. To better model these words, he introduced various constraints in $L_{oo}v$, including: length constrains, the constraint that words in $L$ cannot occur in $L_{oo}v$, phonotactics, etc. He also developed more sophisticated integration methods, in which different classes of OOV words were modelled and included in the language model, thus allowing the detection of OOV words by their linguistic context.

In Chapter 8, we present some work that was carried out, in the context of this thesis, to introduce alternative pronunciation rules in the automatic alignment of read and spontaneous speech. The goal of that work was, on one hand, to improve the time alignment, for browsing and speech synthesis purposes. And, on the other to, allow an understanding of which phenomena of spoken European Portuguese can be reliably modelled by the rules.

### 4.3 Summary

From the present overview, it is patent that WFSTs provide a uniform modelling representation for many knowledge sources used in speech synthesis and in speech recognition. This uniformity of representation, combined with very general and well founded WFST manipulation algorithms, allows the creation of very flexible and efficient speech processing systems. Browsing through the latest proceedings
of the more prestigious conferences in the field, we see that the number of research
groups trying these techniques is growing, which means that the advantages of
WFST techniques are being recognized in the community.
CHAPTER 5

Linguistic Resources

The research described in this thesis was guided not only by theoretical considerations but also by experimental results. All algorithms and techniques presented were tested in controlled experiments.

The experiments were based on various kinds of resources, including: linguistic resources such as corpora and lexica, statistical resources comprised of acoustic and language models, and multiple development tools. Most of those resources were neither developed specifically for this thesis, nor developed by the author.

In this chapter we describe all resources used in this thesis and assign credit to the respective authors.

In the next section we describe the corpora, but since each corpus typically defines a domain of application, we also describe associated lexica and statistical resources. In section 5.2, we describe the development tools used.

5.1 Corpora

5.1.1 The BD-Público Domain

Some of the earliest speech recognition experiments done in this work were based on the BD-Público Corpus [113]. This corpus combines text collected from the
online edition of the Público newspaper, with recordings of read excerpts of the newspaper. It is similar in goal and size to the Wall Street Journal (WSJ0) corpus [124].

The recordings were done in a sound-proof room at INESC, using a high quality microphone. The recording population was selected from students of Instituto Superior Técnico (IST).

Text Corpus

The text component of the corpus was collected from the web, and consisted initially of 6 months daily editions from September 1995 to March 1996. After pre-processing, the size of the text corpus was around 10 million words. In a later phase, the text corpus was extended with the editions from 1996, 1997 and 1998. In this work, we used this extended version with around 45 million words of text.

Vocabulary and Lexica

The BD-Público corpus includes 2 different vocabulary sets, one with 27k words and a smaller subset with 5k. Those sets were obtained from the analysis of the available texts. From the list of 27k words, a pronunciation dictionary or lexicon was build, as described in [113].

Acoustic Model

The statistical acoustic model used in the experiments was originally developed for a dictation task [112]. This model estimates context-independent phone posterior probabilities using three MLPs (Multi-Layer Perceptrons) from the acoustic data at each frame. The 3 streams of probabilities are then combined using an appropriate algorithm [92]. The processing stages are represented in Figure 5.1. The MLPs use the same basic structure and are trained with streams of features extracted from the speech signal with different methods: PLP [58], Log-RASTA [58] and
MSG [80]. For the first two processes, the features are log-energy and PLP/Log-RASTA 12th order coefficients and their first temporal derivatives summing up to 26 parameters. The MSG method uses 28 coefficients. Each MLP classifier incorporates local acoustic context via a multi-frame input window of 7 frames. The resulting network has a single hidden layer with 500 units and 39 output units (38 phones for European Portuguese plus silence).

Figure 5.1: Acoustic model combining several MLPs trained on different feature sets (reproduced from [92]).

5.1.2 The Alert Domain

The latest recognition experiments were performed in the context of the European project Alert. An European Portuguese broadcast news (BN) corpus was collected as part of this project. The corpus comprises two subsets: one to be used for speech recognition training and testing and the other to be used for the development of topic detection algorithms.

From that data, only the training part of the speech recognition corpus (around 60h), and a subset of 256 sentences, randomly selected from the official test corpus, were used in the present thesis.

The performance observed while testing with the 256 utterance subset was actually worse than using the full test set. Yet its use allowed us to quickly obtain
the performance curves needed to compare different optimization schemes. The size of the reduced test corpus was chosen to be of the same order of magnitude as the one used in the BD-Público experiments.

Text Corpus

The text corpus used for language modelling was initially based on the BP-Público corpus which was later extended with texts from other sources:

- Texts from CETEMPúblico [134], another news corpus based on the same newspaper which contains the editions from 1991 to 1998, with a total of 180 million words, but partially overlapping with BD-Público.

- Texts from recent editions of the Público newspaper.

- Texts from other newspapers collected from online editions.

- Manual transcriptions of the newscasts selected for development of the speech recognition system.

The collection of online editions still goes on at the time of writing of this thesis, yet, most language models presented in this thesis were based on the corpus as it was in January of 2002, containing around 430 million words.

Vocabulary and Lexicon

In the Alert project a vocabulary size of 57k words was chosen for various reasons. On one hand, 64k is a common vocabulary size used in similar tasks in other languages, thus being a good starting point for development. On the other hand, some development tools used at the start of the project were restricted to 64k different words or pronunciations.

From the newspaper texts, words which occur more than 50 times (around 100k) were classified according to syntactic classes. From that set, a subset of 56k was
selected based on the weighted frequency of their occurrence in the text according to syntactic classes.

This set was augmented with all words in the training transcriptions giving a total of 57,564 words. The margin to 64k was reserved for eventual new words in the training corpus which was not completely transcribed at the time.

The percentage of words on the development set which were not in this vocabulary was around 1.4%.

The pronunciation dictionary was assembled by collecting pronunciations from lexica developed previously by our research group for different projects. A grapheme-to-phone system was used to generate pronunciations for new words.

Recently, a larger vocabulary and lexicon with over 80k words was developed. This larger vocabulary was not used in this thesis.

**Probabilistic Acoustic Models**

The development of acoustic models started even before the transcription of the training set was complete. An acoustic model originally developed in the BP-Público domain was initially used to align the training material as it was collected. Then began a bootstrapping process which still continues. In this process, the aligned material is used to train acoustic models (MLPs) better suited to the task, which are in turn used to better align the training corpus. This process is not completely automated, as the beginning of each new iteration is an opportunity to tune various parameters of the system, including, among others, the size of the MLPs and corrections to the transcriptions. The acoustic model used in this thesis corresponds to the 6th alignment.

**5.1.3 Other Resources**

Although speech recognition experiments were conducted mainly on the two previous corpus, some other experiments required the use of other linguistic resources:
• All phonetic transcriptions described in this thesis use the SAMPA phonetic alphabet for EP\(^1\) which was defined in the framework of the SAMLA European project and includes 38 phonetic symbols. Although the phonetic alphabet used by both the BD-Público and Alert lexica uses different symbols, they refer to the same phonetic segments.

• The alignment experiments described in Chapter 8 were performed using the spoken book corpus which was then being collected in the scope of the IPSOM project [145].

• The Corgi corpus [158] was used to evaluate the quality of phone and word level alignments. Phone level manual alignments of portions of the corpus were used to develop and evaluate alternative pronunciation rules.

• The lexicon of the Português Fundamental ("Fundamental Portuguese") corpus was used to train and evaluate the grapheme to phone systems discussed in Chapter 8.

5.2 Tools

5.2.1 AUDIMUS

AUDIMUS is a hybrid system [114] that combines the temporal modelling capabilities of Hidden Markov Models with the pattern discriminative classification capabilities of multilayer perceptrons. In this system, a Markov process is used to model the basic temporal nature of the speech signal, while the neural network is used as the acoustic model, estimating context-independent posterior phone probabilities given the acoustic data at each frame. This approach differs from that used by most recognizers due to the estimation of the posterior probability of the word sequence given the acoustic data.

\(^{1}\)http://www.l2f.inesc-id.pt/~imt/ampa.html
The AUDIMUS system comprises tools for extracting parameters from the speech signal as well as tools for training and evaluating multilayer perceptrons. It also includes a speech recognizer. At the start of this thesis, the recognizer was based on the NOWAY decoder [128], considered one of the fastest hybrid decoders reported in the literature. In recent versions of the AUDIMUS system [91], the NOWAY decoder has been replaced by the decoder developed in this thesis and described in Chapter 6.

The probabilistic acoustic models described in previous sections were all trained and executed using this system.

**Phone Models Topology** All HMM WFSTs \( (H) \) used in this thesis were built from phone models with the most common topology used in AUDIMUS: a sequence of states with no self-loops to enforce the minimal duration of the model, and one final state with a self-loop. Figure 5.2 shows the transducer corresponding to phone \( S \) (\( S \) identifies the respective output unit of the MLP which models its probability).

![Phone S topology diagram](image)

Figure 5.2: Phone S topology.

It can be seen in the figure that the weights of the edges that leave a node do not sum up to one as required by the HMM definition. This particular topology, with a number of states equal to half the average phone duration and transition probabilities equal to 0.5, allows a better modelling of phone duration by approximating a pseudo-Poisson distribution [133].
5.2.2 Other Toolkits

WFST Manipulation

The creation and manipulation of WFSTs was done using two toolkits: the command line AT&T FSM Tools\(^2\) [99] and our own Finite-State Toolkit (FSTk) which is described in Appendix A. The compilation of regular expressions to WFSTs was based on the \texttt{HParse} tool from HTK\(^3\) [174].

Language Model Manipulation

Initially n-gram language model were generated with the CMU-Cambridge LM Toolkit\(^4\) [34], later the SRILM toolkit\(^5\) [152] was also used to generate, interpolate and prune [151] language models.

5.3 Summary

In this chapter we described the experimental framework in which the research in this thesis was developed. This thesis benefited from the availability of various development tools, many of them are free for research purposes. This thesis could only be set in the context of the computational processing of the European Portuguese language, thanks to efforts of researchers, many of them from our research group, which developed fundamental linguistic resources such as corpora and lexica.

\(^2\)http://www.research.att.com/sw/tools/fsm/
\(^3\)http://htk.eng.cam.ac.uk/index.shtml
\(^4\)http://svr-www.eng.cam.ac.uk/~prc14/toolkit.html
\(^5\)http://www.speech.sri.com/projects/srilm/
CHAPTER 6

Decoders for \textit{WFST} Structured Search Spaces

In this chapter we describe two decoders, which were developed to allow the use of weighted finite-state transducers as the primary means of integration of knowledge sources in speech recognition.

The development of the first decoder was motivated by the difficulty of optimizing the search space using the \textit{WFST} operations of determinization and minimization which require very large amounts of memory. That was a severe restriction to the size of systems that we could build using the hardware available at the time of its development. This first decoder is characterized by the decoupling of the language model composition from the rest of the search space.

The second decoder uses a different approach to solve that problem: it relies on specialized composition algorithms (described in Chapter 7) to build the search space, either statically or “on-the-fly” in runtime. This approach also addresses other issues such as the size of the system in runtime and the problem of integrating dynamic knowledge sources in runtime.

We start this chapter with the description of various traditional approaches to decoding in large vocabulary speech recognition. Then we present our initial decoder, which decouples the language model from the search network. We then
proceed to the description of our current decoder. We finalize with a short summary highlighting the main aspects of the decoders.

6.1 Traditional Search Approaches

Search in large vocabulary speech recognition is an area which has received much attention from the speech recognition community. As such, many different search techniques have been proposed and studied. In this section, we do not aim at performing a comprehensive review of all that research effort, since very good overviews and descriptions are available in the literature. For example, [108], [62] and [8] are very good references on the subject. The aim of this section is merely to present some approaches which are representative of the major techniques. We also present approaches which were influential in our own work.

Traditional approaches to large vocabulary speech recognition deal with the search problem by embedding a suitable structured search space in the decoder algorithm. The search algorithm cannot be easily separated from the structure of the search space.

As suggested in [8], most approaches to search can be categorized along two orthogonal axes. On one hand, decoders are either time synchronous or not, depending on whether they consider simultaneously hypotheses ending in the same time frame. On the other hand, decoders can be dynamic, if its search space is built dynamically or static otherwise.

We begin this section by describing general techniques that are common to most of them. And then proceed to the description of both dynamic and static time-synchronous approaches and an asynchronous dynamic one.

Pruning The large vocabulary recognition task is so daunting that all practical decoders, including WFST based decoders, need to systematically ignore unlikely hypotheses and thus prune large regions of the search space. Pruning is a non-
admissible technique, meaning that when it is used, the search algorithm might miss the optimal solution. But it is a crucial aspect of all recognition algorithms, and the control of pruning allows tradeoffs between recognition accuracy and search time. Errors due to pruning or other non-admissible techniques are called search errors.

Recombination of hypotheses  The number of possible sentences grows exponentially with the length of the utterance. This growth can be linearized when language models with bounded history are used [123]. The key is using the principle of hypotheses recombination, which says that it is safe to discard the worse of two competing hypotheses, with different word sequences, if those sequences end with a common subsequence which makes them indistinct in the language model. In this situation, the worse hypothesis will always be a worse prefix of any solution. This situation occurs typically in $n$-gram language models, where hypotheses recombination can be implemented by comparing the $n - 1$ most recent words.

Lexicon tree  The pronunciation lexicon, which maps sub-word units to words, is organized as a tree in most large vocabulary decoders. This tree lexicon is used to reduce the computational effort incurred with the large size of the lexicon by sharing common word prefixes (see Figure 6.1).

Language model lookahead  When a tree lexicon is used, the language model probability cannot be incorporated at the start of the pronunciation of a word since its identity is not yet known. To use language model information as soon as possible, the decoder can use instead the best language model probability of all words which share a given branch of the tree. Sometimes the language model probabilities are factorized throughout the lexicon tree so that the probability accumulated in a path from the root of the tree to an intermediate branch is the same as the best language model probability of the words which share the branch.
Since the typical effect of language model lookahead is the reduction of search errors when tighter pruning is used, search may be accelerated by increasing the pruning. This typical effect results of two factors. The first is that it provides some information about future scores of partial hypotheses. The other factor, is that the language model probability is introduced more gradually throughout the pronunciation path, with less abrupt changes to the probability of hypotheses.

**Word-conditioned tree search**

Some of the most popular time-synchronous approaches use dynamic programming search over lexical trees, which are used as the primary representation of the search space [115]. When a bigram language model is used, multiple tree copies have to be made and dynamic programming search can be organized in essentially two ways [122]:

- Using *time-conditioned trees* which consists of creating a copy of the lexicon tree for each frame. Dynamic programming search is performed inside each tree, and when a hypothesis reaches a leaf of the tree at time \( t \), it is moved to the root of the tree conditioned by frame \( t \).
• Using \textit{word-conditioned trees} where a tree copy is made for each possible word (Figure 6.2). When a hypothesis reaches a leaf of the tree corresponding to word \textit{w}, it is moved to the root of the tree conditioned by word \textit{w}.

The potential search space size of this method is enormous; in the worst case, there can be one time-conditioned tree copy per frame, or, alternatively, as many word-conditioned tree copies as the size of the vocabulary. In practice, however, their number can be reduced to reasonable levels by using pruning techniques.

Word-conditioned tree organization of the search space has been the preferred technique when cross word acoustic models or language model lookahead are used. We will focus this section on the word-conditioned technique. Nevertheless, later in this Section, we describe a time-asynchronous technique which uses time-conditioned lexicon trees.

The word-condition tree search approach can be generalized to arbitrary \textit{n}-gram language models by conditioning each lexicon tree with a particular language model history.

Each lexicon tree forms a large HMM network when each node is expanded or replaced by the respective phone HMM or sub-word unit. During search, each state of this large HMM can contain a different partial hypothesis. When a state in the search space contains a hypothesis we say that the state is \textit{active}.

The time-asynchronous search progresses frame by frame. At each frame, two iterations are performed over the list of active hypotheses:

• In the first one, the active hypotheses are propagated among inner states of each lexicon tree HMM.

• In the second one, hypotheses which reach a leaf state of a lexicon tree HMM are propagated to the root state of another language model history conditioned tree. This tree is created if it does not exist yet. Language model factorization is performed when each tree is created.
Each lexicon tree contains pronunciations for all the vocabulary words. This helps the creation of new lexicon trees since they share the same structure.

Language model factorization is efficiently performed by propagating the language model probabilities from the leaves to the root of the tree [121]. The maximum language model probability of all pronunciations \( lm[s] \) which share a given node \( s \) is computed recursively as the best such value of all its children nodes, \( lm[s] = \max_{r \in \text{children}[s]} lm[r] \). The language model factorization associated with the branch from parent node \( s \) to children node \( r \) is computed as \( lm[r]/lm[s] \). In [121], the structure of the tree was optimized for language model lookahead by grouping sequences of parent nodes with only one child, so that each inner node of the resulting tree has at least two children.

**Static search network**

In [49] Federico et al. demonstrated that it is feasible to use a static search network in LVCSR. Their original approach uses an interpolated bigram language model and
multiple lexicon trees. The smoothing method used in the language model consists of the following interpolation:

\[ P(z|y) = f'(z|y) + bo(y)P(z) \]  \hspace{1cm} (6.1)

where \( f'(z|y) \) is estimated from bigrams appearing in the training data, \( P(z) \) is the unigram probability, and \( bo(y) \) is an interpolation parameter.

The approach is also based on language model history conditioned trees, yet it uses a number of techniques to achieve a tight representation of the search space:

- Each lexicon tree contains only the words which followed its language model history.

- Only language model histories which exist explicitly in the language model have an associated tree. The other histories are modelled using weighted null transitions to a unigram tree. These transitions reflect the interpolated structure of the language model patent in Equation 6.1.

- Language model lookahead is used to optimize each tree.

- Hypotheses recombination is implemented in the structure of the network with explicit null transitions from each leaf to the root of a destination tree. This other tree is conditioned by the word whose pronunciation ends at the leaf.

Figure 6.3 illustrates the search space obtained with these techniques.

A reduction of 3 times of the size of this search network was obtained by applying the classical automata minimization algorithm described in [4]. Figure 6.3 does not show this optimization.

In [23], an efficient approximation to the minimization step is proposed by sharing the pronunciations which arrive at the same lexicon tree. This sharing is
possible because, on one hand, all pronunciation paths linking to the same LM-history conditioned lexicon-tree are instances of the same word. And, on the other, long branches starting at a node where the identity of a word is first established are given probability 1 as a consequence of the language model lookahead optimization. Such *tails* can be shared among different trees while preserving most of the overall tree structure of the network.

**Single lexicon network**

In [43], Demuynck et al. proposed the use of a single lexicon network to perform search. This lexicon network allows not only the sharing of pronunciation prefixes, but also of suffixes. The main motivation for sharing suffixes was reducing the size of the network when cross-word phones are used.

The lexicon network is built starting from a lexicon tree. Then the word identifying information is moved towards the root to the first non-shared arc. The leaves of the tree are shared, and suffixes are shared recursively in a reverse tree structure.

Since the lexicon network is neither time nor word conditioned, multiple hypotheses can share the same state in the network. Each hypothesis is conditioned
by a language model history. Hypotheses with different language model histories can share the same network state, but hypotheses with the same one are merged.

Because hypotheses with different language model histories simultaneously coexist in the network, one cannot easily incorporate language model lookahead information. As an approximation, the authors proposed the factorization of the unigram probability in the prefix part of the network where the word identity is not known.

**Start-synchronous search**

The decoders presented so far are time-synchronous, meaning that all hypotheses in the decoder at a given time are partial hypotheses which span from the beginning of the utterance to the same time instant. Most asynchronous search techniques have been called *stack decoders* in the speech recognition community. The name *stack decoder* was coined because such decoders are based on a priority queue data structure (erroneously called stack). The queue contains hypotheses which span from the beginning of the utterance to some latter instant, not necessary the same for all hypotheses. The hypotheses are all complete in the sense that they end in the end of a word.

Search in a stack decoder is conceptually very simple – it is a best first algorithm that consists of repeatedly removing the best hypothesis from the queue, expanding it, and placing the newly created hypotheses in the queue. This loop proceeds until a solution which spans from the beginning to the end of the utterance is found. The expansion of the current hypothesis consists of performing an acoustic match between all words in the vocabulary and the signal starting at the end of the hypothesis. This acoustic match is sometimes done in two steps, the first one, called *fast match*, uses simplified acoustic models to select a small list of candidate words which are then used to perform a more expensive *detailed match*. The objective function used to sort hypotheses in the queue is $f(h) = g(h_0) + b^r(h_{i+1}^{[X]}), $ where
\( g(h^t_0) \) is the cost of the hypothesis \( h \) from time 0 to time \( t \) where it was matched with the acoustic signal, and \( b^*(h^X_{t+1}) \) is an estimate of the cost to complete the match to the ending time of the utterance \( [X] \). If \( b^* \) never overestimates the cost, then the algorithm becomes a version of the \( A^* \) algorithm [116] and is guaranteed to find the best solution.

An advantage of stack decoders relative to most time-synchronous approaches is that the language model is decoupled from the acoustic search, thus allowing the use of language models with long histories. The main problem when designing a stack decoder is finding an admissible way to compare hypotheses with different lengths. That comparison is needed to sort hypotheses in the queue and to prune away bad hypotheses. It is very difficult to find admissible \( b^* \) functions in single pass systems.

One way to deal with hypotheses with different lengths consists of expanding the shortest one first. This approach is sometimes called multi-stack search since a different queue is kept for each frame, containing the hypotheses which end in that frame.

The NOWAY system [129] is based on this multi-stack principle. One of its characteristics is that all hypotheses which end in the same frame are expanded simultaneously using a time-synchronous Viterbi search based on a time-conditioned lexicon tree. This motivated the designation of this approach as start-synchronous search.

Regarding the search algorithm, various pruning techniques are implemented in NOWAY: beam pruning is implemented by keeping a different reference score for each time frame, which is updated greedily or by backtracing from complete hypotheses; phone deactivation pruning [128] uses posterior probability phone estimates to prune unlikely phones, and each stack has a maximum number of hypotheses.

Some forms of language model lookahead are also implemented by spreading
language model scores in the time-conditioned search tree: spreading unigram probabilities, spreading the best language model score given the various hypotheses which are to be extended, and using the exact language model score if the number of hypotheses is below a given threshold.

6.2 Decoupled Decoder

The main design characteristic of our initial decoder is that the language model is decoupled from the search network, avoiding the problem of the memory required for its integration. The search network is a distribution-to-words WFST which is build outside of the decoder. This network is frequently build as $H \circ L$, and does not include the language model $G$. The language model score is explicitly added by the decoder when a search space edge which outputs a word is crossed during search.

In terms of probabilistic acoustic modelling, the decoder can use either a hybrid model that combines the temporal modelling capabilities of HMMs with the pattern classification capabilities of multilayer perceptrons (MLPs), or it can work with Gaussian mixture distributions.

The decoder is mainly time-synchronous, but we have also performed experiments in which it operated as a stack decoder in the spirit of the start-synchronous approach (Section 6.1). In both modes of operation, the language model is decoupled from the search. This separation has the disadvantage that full language model lookahead is not used. We mitigate this problem by allowing the use of the unigram probability as an early approximation.

The hybrid decoder has the following main inputs:

- The output of the neural network.
- A prior probability vector, to convert the probabilities estimated by the neural network to scaled likelihoods.
• An n-gram language model, represented explicitly or using a WFST.

• A distribution-to-word WFST that is built outside of the decoder and that is usually the composition of the acoustic models (their HMM topology), the lexicon and a lattice.

6.2.1 Time-Synchronous Search

The time-synchronous search is a token passing implementation of the Viterbi algorithm [175], however, instead of one hypothesis, our tokens contain hypsets.

We define a hypset $h_s^t$ as a set containing hypotheses that end in the same state $s$ of the WFST and at the same time $t$.

In our decoder, a hypothesis is a sequence of words with an associated cost, which is the combination of the acoustic cost of the best match of that sequence with the utterance and the cost (-LogProb) of the sequence in the language model.

We associate a token with each active state of the network. Initially, a hypset containing only the empty hypothesis is placed in the token of the initial state of the WFST. Then the search starts: at each time instant, the hypsets are propagated through non-epsilon edges and merged at the destination state; then, the epsilon edges are traversed and the hypsets merged.

The merging of hypsets can be a relative time consuming operation. We try to reduce the impact of that operation by optimizing some frequent cases: in general, the network has a significant percentage of linear paths (sequences of states with only one successor, or with only a self-loop and a successor); and most of the output labels are epsilon.

We do some bookkeeping and share hypsets among tokens. Whenever possible, the hypset is processed as an opaque entity that is shared and propagated from state to state, the differences in acoustic costs being kept separately at each token. Only when two or more different hypsets converge in a state, or when one hypset traverses an edge with a non-epsilon output label, are the hypsets really merged,
and each individual hypothesis has to be changed. This bookkeeping is specially suited for the common case of tree lexica or the more general case of determinized WFSTs.

The language model score is applied to the hypotheses of the hypset as soon as an edge with an output label is traversed. We adopt a cache to reuse hypsets that were previous extensions of the same hypset with the same word. This speeds up the propagation when the source state of that edge stays active in successive time instants.

During the merging of hypsets, the Viterbi approximation is enforced, and only the best hypothesis in the same context is kept. When using n-gram language models, two hypotheses are considered in the same context if they have the same \( n - 1 \) last words. When WFST language models are used, their context is the WFST state. Keeping a limited context also enforces hypotheses recombination.

### 6.2.2 Stack Decoding

A stack decoding mode heavily influenced by the NOWAY decoder was also implemented. In this mode, the search proceeds at two levels: an outer word level which searches over the sequences of words (hypotheses), and an inner level search that extends a set of hypotheses (hypset) with another word.

#### Word Level Search

Underlying the search at the word level is an array \( H[t] \), indexed in time, that, for each time instant, keeps the set of hypsets that ended at that time.

In the beginning of the search, we build a hypset containing only the empty hypothesis and ending at the initial state of the WFST. This pair is placed at \( H[0] \) and the search starts.

This word-level search consists only of a loop where the hypsets that end at each successive time instant are expanded. Notice that only one expansion is performed
in each time instant, and all hypotheses ending at the same time are expanded simultaneously.

The expansion of the hypsets consists of extending its hypotheses with another word. At the end of the utterance, we select the solution as the best hypothesis in a hypset ending at a final state of the WFST. The word expansion or state level search is time-synchronous.

State Level Search

The state level search makes use of the distributions-to-words WFST build using the automata algebra. Because the search progresses word by word, the final edge of each word in this WFST must be marked with a special end-of-word (EOW) label.

At the beginning of the expansion, the hypsets in the current time instant (all \( h_i^t \in H[t] \)) are placed at corresponding word initial states (that is, all states following EOW edges from the final state of the hypset). Then a time-synchronous search is performed. During this search, when an EOW edge is traversed, the hypset is not propagated to the destination state, but is instead merged with the corresponding hypset at \( H[t] \), to build a hypset that will be later expanded at time \( t \).

6.2.3 Pruning

The use of pruning is fundamental for the efficiency of the search. We use three types of pruning:

Beam Pruning

When the decoder operates in a non time-synchronous mode, we cannot know the value of the best state at a given time instant. To perform beam pruning we keep an array, indexed in time, of best-so-far costs that are updated as the search
progresses. Whenever the best cost in a hypset is over a given threshold (beam) of the best-so-far at that time, the state containing the hypset is deactivated. During the merging of hypsets, every hypothesis over the threshold is removed from the resulting hypset. When a hypothesis is found that has a lower cost than the best-to-far at that time, the value in the array is updated.

When in time-synchronous mode, an additional pass is made at each time instant over the active states and those found outside the beam threshold are deactivated.

This kind of pruning acts as a fundamental termination condition for the state level search in the stack decoding mode, as it proceeds until all states are deactivated.

Hypset window

An additional form of pruning is used which consists of limiting the maximum number of hypotheses in each hypset and keeping only the best $m$.

Phone Deactivation Pruning

When using hybrid models, the decoder has access to the posterior probabilities that are estimated by the neural network, and so phone deactivation pruning [128] can be adopted.

This form of pruning takes advantage of the fact that the MLP directly estimates the posterior probability of each phone (the probability $P(q|x)$ of the phone $q$ given the acoustic vector $x$). This pruning consists of flooring the probability of a given phone to a very low value when it is below a given threshold. This has the effect of allowing the MLP to deactivate some phones. There is usually an optimal value for the threshold: if too large, then the search will be faster but more error prone; if it too low, then the search will be slower with no advantage regarding the accuracy (sometimes the accuracy is even worse due to the MLP having difficulty modelling
low probabilities).

### 6.2.4 Experiments

Some experiments were done to evaluate the performance of the decoder, using the same framework as the AUDIMUS system [112], a hybrid LVCSR system developed for European Portuguese. As stated in Chapter 5 we used as many components of that system as possible, in order to compare our decoder directly with NOWAY. The experiments performed with the decoupled decoder were all done in the BD-Público domain, first using a 5k vocabulary and, in later experiments, a 27k vocabulary.

**Lattice Decoding**

Since the decoder was being build simultaneously with the FSTk library used to manipulate WFSTs in runtime, it was necessary to perform very controlled experiments with the purpose of detecting implementation errors and to calibrate various aspects of the decoder.

Those experiments were set in the 5k words task of the BD-Público domain, and were based on a search space build using, lattices (or word graphs) of word hypotheses generated by the NOWAY decoder.

This was a rescoring experiment where lattices were used as an additional constraint to the decoder. They had an average of 136 word arcs per spoken word. We performed some experiments with the expansion of words to phones and found that the use of language model probabilities as soon as possible allows the use of tighter beams. No language model lookahead information was used. The best performance was obtained by placing the arc with the word output at the beginning of the pronunciation and sharing the suffixes of the words arriving at the same state.

The lowest error rate obtained with a reasonable beam (50.0) was 16.3%. Some experiments were performed to find the best values for the language model scale
(5.0) and word insertion penalty (0.0). This WER is similar to the best error rate obtained by the NOWAY decoder using the same models (16.5%).

5k Words Large Vocabulary Task

We performed some experiments with large vocabulary recognition using either the stack decoder mode, or a time-synchronous search based on a lexicon loop network WFST. Using these networks the time-synchronous decoder becomes similar to the approach described in Section 6.1.

We observed that the time-synchronous search is faster than the stack decoder by a factor of 10 times. In table 6.1, we present the results obtained with the lexicon loop WFST. The results where obtained using a 600Mhz Pentium III PC running Linux. The better results of the time-synchronous search are not surprising, given that our stack decoder implementation has the unnecessary overhead of propagating hypsets instead of simple hypotheses.

The time-synchronous was tested with and without unigram lookahead information in the search network. As can be seen in Figure 6.4, unigram lookahead generally provides a very significant search speedup.

<table>
<thead>
<tr>
<th>WER</th>
<th>xRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.9</td>
<td>1.0</td>
</tr>
<tr>
<td>19.2</td>
<td>1.7</td>
</tr>
<tr>
<td>17.2</td>
<td>3.0</td>
</tr>
<tr>
<td>17.0</td>
<td>4.2</td>
</tr>
<tr>
<td>16.8</td>
<td>6.9</td>
</tr>
<tr>
<td>16.3</td>
<td>10.1</td>
</tr>
</tbody>
</table>

Table 6.1: 5k LVCSR results using a lexicon loop WFST.
Figure 6.4: Unigram lookahead vs no lookahead in the 5k words large vocabulary task.
27k Words Large Vocabulary Task

Given the positive results obtained with the decoupled decoder we proceeded to extent it to a more demanding task in the BD-Público domain, with a 27k vocabulary and a trigram language model with 17 Million $n$-grams. In this task, the acoustic model (MLP) was also improved.

Soon we faced some problems as it became very difficult to achieve few search errors in a reasonable fast system. Table 6.2 shows the results of some experiments. In this task, the NOWAY decoder was able to obtain as little as 10.4 WER at 6.7 xRT.

<table>
<thead>
<tr>
<th>Beam</th>
<th>Hypset</th>
<th>WER</th>
<th>xRT</th>
</tr>
</thead>
<tbody>
<tr>
<td>75</td>
<td>15</td>
<td>49.8</td>
<td>3.5</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>15.0</td>
<td>7.3</td>
</tr>
<tr>
<td>110</td>
<td>15</td>
<td>13.2</td>
<td>10.0</td>
</tr>
<tr>
<td>125</td>
<td>15</td>
<td>12.4</td>
<td>16.6</td>
</tr>
<tr>
<td>135</td>
<td>15</td>
<td>12.3</td>
<td>19.3</td>
</tr>
<tr>
<td>150</td>
<td>15</td>
<td>12.2</td>
<td>26.0</td>
</tr>
<tr>
<td>135</td>
<td>20</td>
<td>12.1</td>
<td>27.6</td>
</tr>
<tr>
<td>150</td>
<td>20</td>
<td>11.9</td>
<td>37.8</td>
</tr>
</tbody>
</table>

Table 6.2: 27k LVCSR results using a lexicon loop WFST.

The main problem was balancing the maximum size of the hypset with the beam. To further control the size of the hypsets, a hypset beam was tried. This beam pruned hypotheses relative to the best one in the hypset. This additional beam was not very effective and complicated the decoder with another parameter to tune.

At this point, we decided that an “on-the-fly” composition algorithm of the lexicon with the language model would be the best way to integrate the language model in the decoder and stopped investing in the idea of multiple hypotheses per token.
6.3 Tightly Coupled Decoder

Having decided to explicitly include the language model in the search WFST, a new decoder was developed based on the time-synchronous mode of operation of the previous one.

Very few changes were required to the search algorithm itself since it is implemented using principles of data abstraction. The decoder manipulates tokens using a well-defined programming interface independent of the particular implementation. The main change is that the new token implementation only contains one hypothesis and not a hypset.

The new decoder is also parameterized to allow the search space to be specified as a single compiled distribution-to-words WFST or to receive various components such as the acoustic WFST, the lexicon and the language model WFST and perform their composition “on-the-fly” during recognition. In particular, the “on-the-fly” composition of the lexicon with the language model can be done with any one of a variety of algorithms presented in Chapter 7.

6.3.1 Pruning

The pruning techniques used in the decoder were improved from the previous version. The new version now uses now 3 forms of pruning: phone deactivation pruning, a new form of beam pruning, and histogram pruning. Hypset window pruning is, of course, no longer used.

A new form of beam pruning was introduced, which differs from the previous one in its eagerness. It consists of pruning hypotheses while they are being generated by using the cost of the best hypothesis so far as a reference. When a hypothesis with cost $c_t$ at time $t$ is propagated through an edge with input label $d$ and weight $w$, its cost in the next frame is updated with two components: a transition weight $w$ that incorporates linguistic constraints; and an acoustic weight $distr(d, t + 1)$.  

90
Because the transition weight is often of the same order of magnitude as the beam, we obtain significant improvements by performing the pruning test twice: first the cost \( c_t + w \) is tested and then \( c_t + w + \text{distr}(d, t + 1) \). If the first test fails, some computations are avoided, including both the computation of \( \text{distr}(d, t + 1) \) and the bookkeeping associated with the expansion of the hypothesis.

This form of eager pruning assumes that \( \text{distr}(d, t) \) always returns a positive value, which is true if the distribution is modelled using Gaussian mixtures, since the cost is by definition \( -\log() \) of a probability value. But when scaled likelihoods are used, as is our case, \( \text{distr}(d, t) \) can take negative values. Yet, that does not seem to be problematic, and in all experiments performed, we observed a significant speed improvement, and no accuracy penalty, when using this form of eager pruning.

The function of histogram pruning is to reduce peak resource usage, time and memory, to a reasonable limit. It consists of establishing the maximum number \( m \) of hypotheses that are expanded at each frame. Whenever their number is over the limit, only the \( m \) best are kept and the other are pruned. If the value of \( m \) is set to a reasonable value (such as 100000), then it has virtually no negative effect on the accuracy of the decoder, while preventing it from virtually stopping when there is a severe acoustic mismatch relative to the training conditions.

6.3.2 Time Markers

Time markers are a generalization of the EOW label introduced in Section 6.2.2. They are special labels which are used by the decoder to keep track of time in the decoding hypotheses. These markers are interpreted by the decoder as regular \( \epsilon \) labels, but when a token traverses such a label, a structure containing the current frame or time instant is added to the hypothesis.
Segmentation

Time markers have many uses, the most common one being allowing time segmentation of the speech signal at user specified levels. In particular, word or phone segmentations are achieved by placing a time marker label at the end of each word or phone model.

Confidence Features

The decoder can be used to extract features which are the basis of the computation of confidence measures. Time markers play a role in the collection of those features, since the features are associated with each hypothesis whenever a time marked is crossed.

Lattice Generation

Time markers are also used as the basis for generating lattices. In lattice generation mode, a lattice WFST is build in the following way: each hypothesis keeps a lattice state which is the last lattice state the hypothesis has crossed, and whenever it crosses either a non-$\epsilon$ output edge, or a time marker edge, going to state $s$, an edge is added to the lattice going from the last recorded state in the hypothesis to a new lattice state $(t, s)$. Regularly, garbage collection is performed to remove dead-end paths from the lattice WFST. At the end of the utterance, $\epsilon$ removal is applied to convert this WFST into a word lattice.

This mechanism generalizes various other algorithms. For example, if time markers are placed at the end of each phone, it implements the algorithm described in [89], where a word lattice is generated based on a phone level lattice. If they are placed after each non-$\epsilon$ input edge then it becomes the algorithm in [17].

92
6.3.3 Experiments

Since the most crucial aspect of the tightly coupled decoder is the construction of
the search space, the experiments done with it are described in the next chapter.

6.4 Summary

We have presented two decoders that use weighted finite state transducer algebra
to include knowledge sources early in a one-pass search.

The first decoder operates mainly time-synchronously for use with finite-state
problem specific grammars and for large vocabulary tasks using a lexicon loop
network. Its main characteristics is the decoupling of the language model from the
search thanks to the use of multiple hypotheses per state.

The second decoder is a more traditional time-synchronous algorithm and relies
on transducer operations to build the search space. Combined with the “on-the-
fly” composition algorithms presented in the next chapter, this decoder can be the
basis of a dynamic WFST recognition system.
CHAPTER 7

Integrating the Lexicon with the Language Model

7.1 Introduction

In this chapter we propose a WFST composition algorithm, specially tailored to integrate the lexicon with the language model. The algorithm is designed to compose the lexicon with the language model while simultaneously optimizing the resulting transducer. Furthermore, it is our goal to perform the computation “on-the-fly”.

The optimizations we apply to the composition are determinization, minimization and pushing.

Our “on-the-fly” algorithm is exact for some local knowledge integration and optimization operations such as composition and determinization. The other operations, which have a global nature, are approximated.

In the rest of this chapter, we introduce the basic idea behind our algorithm and gradually present how the various optimization operations were developed and incorporated.
7.2 Basic Idea

We base our algorithm for the lexicon and language model integration on the following theorem [97].

**Theorem 7.1** The composition of sequential transducers is sequential.

This result is a straightforward consequence of the definitions of sequential transducer and transducer composition.

Based on this theorem, we can avoid the explicit determinization of the composition transducer, if the lexicon and the language model are represented as sequential transducers. One useful aspect of this theorem is that composition can be performed "on-the-fly".

To exemplify, we use the toy lexicon listed in Table 7.1, which can be represented by the sequential transducer $L$ shown in Figure 7.1, and the sequential language model $G$ shown in Figure 7.2.

<table>
<thead>
<tr>
<th>Word</th>
<th>Phone Seq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMO</td>
<td>a m o</td>
</tr>
<tr>
<td>ATO</td>
<td>a t o</td>
</tr>
<tr>
<td>OLA</td>
<td>o l a</td>
</tr>
<tr>
<td>OTO</td>
<td>o t o</td>
</tr>
</tbody>
</table>

Table 7.1: Toy lexicon.

Looking at their composition $L \circ G$, shown in Figure 7.3, we observe that it is indeed sequential. At a first glance, it seems that the composition algorithm is all that is needed to obtain a deterministic composition. Yet, in the figure only the useful states are shown, but non-useful states are also generated when the usual transducer composition algorithm is used.

Figure 7.4 shows all the states generated by the algorithm, including the non-useful states 2, 5 and 6.
Figure 7.1: Sample Lexicon \( L \).

Figure 7.2: Sample Language Model \( G \).

Figure 7.3: Composition \( L \circ G \).

Figure 7.4: All paths generated by the composition \( L \circ G \).
The occurrence of non-useful states is not apparent to casual users of automata toolkits because most implementations of transducer composition (and of automata intersection) employ a post-processing step that removes such states. Non-useful states are either non-accessible or non-coaccessible. The generation of non-accessible states can be avoided "on-the-fly" by generating the composition transducer state by state while performing a graph transversal starting at the initial state. The removal of non-coaccessible is more problematic, as it requires either a post-processing step or the use of unbounded lookahead, neither of which is compatible with our goal of an efficient "on-the-fly" composition algorithm.

In order to avoid the generation of non-coaccessible states during composition, we propose a pre-processing step in which every \( \epsilon \) output edge of the lexicon WFST is tagged with the set of non-\( \epsilon \) output labels directly accessible from that edge. Figure 7.5 shows how the lexicon looks like after being tagged with these sets. The sets can be easily obtained by a post-order traversal of the lexicon WFST, in which the set assigned to an \( \epsilon \) output edge is obtained as the union of the sets assigned to all edges leaving its destination state (non-\( \epsilon \) output edges are assigned a singleton set). In our proposed composition algorithm, these sets are used to avoid following \( \epsilon \) output edges of the lexicon when they do not lead to a useful path.

### 7.3 Composition of the Lexicon with the Language Model

In traditional continuous speech large vocabulary recognition systems, the search space is sometimes described in terms of copies of the lexicon, while in WFST approaches an algebraic description is preferred. In the following presentation of the specialized composition algorithm, we shall also describe the search space graphically in terms of lexicon copies thanks to the simplicity of the composition algorithm. We hope that, since the specialized composition algorithm is equivalent
to using explicit determinization for search space optimization, this description will help the reader to better understand the search space generated by WFST approaches. Of course, the equivalence is only in the sense that the resulting networks represent the same relation, and both are deterministic. The precise topology, distribution of weights, and the location of output labels along paths may be different.

Let us start by analyzing the search space which would be produced by our composition algorithm in a paradigmatic situation: In this situation, the lexicon WFST is organized in a loop, such that the initial state is both the initial and final state of all pronunciations.

Figure 7.5 shows a minimal deterministic tagged lexicon loop $L^*$ representing our toy lexicon, and Figure 7.6 shows its composition with the language model $G$. In these figures the set of all words is represented as {...} and the initial/final state of the lexicon loop is shown twice for clarity.

Each state of the composed WFST corresponds to a pair of states from the argument transducers (for example, state (1,0) corresponds to state 1 of the lexicon and to state 0 of the LM). To help the discussion, we designate states in the form $(i_l, q_o)$, where $i_l$ is the initial state of the lexicon and $q_o$ is a language model state, as anchor states. Assuming that the lexicon includes pronunciations for all words
in the language model, anchor states (and indirectly, the language model) shape the global structure of the search space: every edge in the language model will correspond to a path between anchor states in the composition transducer.

To help the local characterization of the search space, we call states in the lexicon that are between the initial state and a non-ε output edge, as prefix states (shown inside a gray rectangle in Figure 7.5). And we designate states in the lexicon between a non-ε output edge and the final state as suffix states (shown inside a white rectangle).

Analyzing the composed WFST, we see that two types of replication of the lexicon occur: one that starts in anchor states and replicates lexicon prefix states until non-ε output edges are found, and another which consists of suffix states and progresses until the end of the pronunciation.

In the first type of replication, edges (and states) of the lexicon are filtered by the input labels (words) of edges leaving the language model state, so that only edges of the lexicon that match a language model label in its tagging set or in its output label are copied. In the second type, the filtering is done by the labels of incoming edges to the language model state which corresponds to the destination anchor state. Three replication regions are marked in Figure 7.6, one prefix region, in gray, and two suffix ones in white rectangles.

It is obvious that, due to the selective cloning of the lexicon WFST, its topology
is crucial in determining the shape, size, and performance of the search space.

Algorithm

In this section we present a first version of the composition algorithm that uses the sets associated with the lexicon $WFST$ to avoid the generation of non-coaccessible paths.

Since our goal is an “on-the-fly” algorithm, it must be designed to allow the incremental construction of the composition network. We followed the design of the AT&T FSM Library [99] for “on-the-fly” generation of $WFST$s. In this design, the essential interface to a $WFST$ $T$ consists of three function:

- INITIAL($T$) which returns the initial state of $T$.
- FINAL($T$, $q$) which returns the final cost $\rho(q)$ of state $q$ if it is final or 0 otherwise.
- ARCS($T$, $q$) which returns the set of edges leaving state $q$.

These functions are sufficient for a graph traversal algorithm to be able to explore the $WFST$ without generating non-accessible states. As an example, Figure 7.7, shows an algorithm which performs the explicit copy of a $WFST$. This particular algorithm uses a queue $S$ to store unvisited states.

Our specialized composition algorithm differs from the generic one only in the ARCS function, the INITIAL and FINAL functions are similar and are shown in Figure 7.8.

Figure 7.9 presents an ARCS function that uses the sets associated with edges in the lexicon to avoid the generation of non-coaccessible paths.

The argument to the function is a state in the composition $WFST$ that corresponds to a pair of lexicon and language model states. The function uses two embedded loops to match the set of edges leaving the lexicon state with the set of edges leaving the language model state.
COPY($T_{in}$)
1 $S \leftarrow E_{out} \leftarrow \{\}$
2 $i_{out} \leftarrow \text{INITIAL}($$T_{in}$$)$
3 $Q_{out} \leftarrow \{i_{out}\}$
4 ENQUEUE($S$, $i_{out}$)
5 if $\emptyset \neq \text{FINAL}(T_{in}, i_{out})$
6 then $F \leftarrow \{i_{out}\}$
7 $\rho_{out}[i_{out}] \leftarrow \text{FINAL}(T_{in}, i_{out})$
8 else $F \leftarrow \{\}$
9 while $S \neq \{\}$
10 do $q \leftarrow \text{HEAD}(S)$
11 DEQUEUE($S$)
12 $A \leftarrow \text{ARCS}(T_{in}, q)$
13 $E \leftarrow E \cup A$
14 for each $(q, l, o, d, w) \in A$
15 do if $d \notin Q$
16 then $Q \leftarrow Q \cup \{d\}$
17 ENQUEUE($S$, $d$)
18 if $\emptyset \neq \text{FINAL}(T_{in}, d)$
19 then $F_{out} \leftarrow F_{out} \cup \{d\}$
20 $\rho_{out}[d] \leftarrow \text{FINAL}(T_{in}, d)$
21 return $T_{out}$

Figure 7.7: Function to explicitly copy WFSTs.

INITIAL($q$)
1 $(q_l, q_g) \leftarrow q$
2 return INITIAl($q_l$) and INITIAl($q_g$)

FINAL($q$)
1 $(q_l, q_g) \leftarrow q$
2 return FINAL($q_l$) $\otimes$ FINAL($q_g$)

Figure 7.8: Definition of INITIAL and FINAL functions for “on-the-fly” composition.
ARCS\((T, q = (q_t, q_g))\)
1   \(A \leftarrow \{\}\)
2   if \(q_t = i_t\)
3      then for each \((q_g, \epsilon, o_g, d_g, w_g)\)
4         do \(A \leftarrow A \cup (q, \epsilon, o_g, (q_t, d_g), w_g)\)
5      for each \(e_l = (q_l, l, o_l, d_l, w_l)\)
6         do if \(\text{DestSet}(e_l) = \Delta_l\)
7            then \(A \leftarrow A \cup \{(q, l, \epsilon, (d_l, q_g), w_l)\}\)
8            else for each \(e_g = (q_g, l_g, o_g, d_g, w_g)\)
9               do
10                  if \(l_g \in \text{DestSet}(e_l)\)
11                     then if \(o_l = \epsilon\)
12                        then \(A \leftarrow A \cup \{(q, l, \epsilon, (d_l, q_g), w_l)\}\)
13                        else \(A \leftarrow A \cup \{(q, l, o_g, (d_l, d_g), w_l \otimes w_g)\}\)
14      return \(A\)

Figure 7.9: Non-coaccessible paths avoiding algorithm.

The algorithm begins by dealing with the presence of \(\epsilon\) input edges in the language model (lines 2 to 4). When they occur, a corresponding edge is created in the composition WFST connecting only anchor states. Here, it is clearly assumed that the lexicon loops through the initial state.

Then the algorithm loops through the lexicon edges. Given a particular lexicon edge, the first test verifies if the edge is in the suffix region of the lexicon (line 6), and if so, the edge is cloned into the composition WFST (line 7). If the lexicon edge does not belong to the suffix region, a loop iterates over the language model edges to find the matching ones.

A language model edge is only considered if its input label is among those that can be reached from the lexicon edge (line 10); this is the test that avoids the generation of non-coaccessible edges. Function DestSet returns the set of non-\(\epsilon\) output labels reachable from an edge in the lexicon.

If the lexicon and language model edges match, two particular situations can occur: 1) either the lexicon edge has \(\epsilon\) output, in which case the lexicon edge will
be cloned to the composition \( WFST \) (line 12), or 2) the lexicon edge has a non-\( \epsilon \) output, in which case a transition will occur in the language model component of the destination state, causing the language model output to be produced and the language model weight to be incorporated in the composition \( WFST \) edge (line 13).

### 7.4 Pushing

In the composition algorithm, as described so far, the language model weights are incorporated into the composed network only in non-\( \epsilon \) output edges.

This distribution of language model weights throughout paths is bad for speech recognition because the language model score is incorporated in a single step, and is delayed until the identity of a word if determined. As we saw in chapter 6, it is usually much better to spread the language model score throughout the pronunciation path, even before that identity is completely known.

When explicit weighted determinization is used to optimize the composition of the lexicon with the language model, weights are naturally spread throughout paths. Furthermore, Mohri and Riley presented in [104] a global weight pushing algorithm which globally spreads weights in a \( WFST \). That algorithm defines a potential function on states which allows weights to be changed locally without affecting the global weight of paths.

A potential function \( V : Q \rightarrow K - \bar{0} \) can be used to re-weight the \( WFST \) by updating its initial weight, final weights and transition weights as follows:

\[
\lambda \leftarrow \lambda \otimes V(i) \quad (7.1)
\]

\[
\forall f \in F, \rho(f) \leftarrow V(f)^{-1} \otimes \rho(f) \quad (7.2)
\]

\[
\forall (q, l, o, d, w) \in E, w \leftarrow V(q)^{-1} \otimes (w \otimes V(d)) \quad (7.3)
\]

This re-weighting is possible if the semi-ring \( K \) is divisible, meaning that for each \( a, b \in K \) such that \( a \oplus b \neq \bar{0} \) there exist \( a_1 \in K \) such that \( a = (a \oplus b) \oplus a_1 \).
Any such $a_1$ can be selected as the remainder of the division of $a$ by $(a \oplus b)$, and we can write $a_1 = (a \oplus b)^{-1} \otimes a$.

The re-weighting algorithm proposed in [104] consists of a generalization of the shortest distance from each state $q$ to a final state, and is defined as:

$$V(q) = \bigoplus_{\pi \in P(q)} w[\pi]$$ (7.4)

where $P(q)$ is the set of all paths from state $q$ to a final state, and $w[\pi]$ is the accumulated weight of path $\pi$.

Our composition algorithm can be improved to spread language model weights throughout the replication of the prefix region of the lexicon in a similar way as the weighted determinization algorithm. The modification is explained below:

Whenever the algorithm verifies that a word from the language model is present in the set associated with an $\epsilon$ output edge of the lexicon, it keeps track of the edge with the best matching weight in the language model. The difference between this best weight and the language model weight which was spread from the previous anchor state, is combined with the weight of the lexicon edge in order to produce the weight of the composed edge. To help this computation, the language model weight spread from the last anchor state is kept in each state of the composed WFST.

On the other hand, when only one language model edge matches a lexicon edge, we can produce the output label immediately. This has the effect of pushing the output labels toward the initial state. A binary value is associated with each composed state indicating if the output label was produced or not and is used to avoid its production more than once.

These two improvements of the algorithm do not change the topology of the network, only the distribution of weights and output labels throughout its paths.
Algorithm

Figure 7.10 presents a modified version of the ARCS function that provides pushing of weights and output labels. The algorithm is similar to the previous one, but now, while matching lexicon and language model edges, it must also collect information necessary for the implementation of pushing. For each destination state \( d \), two values are now stored: \( pushedCost[d] \), which keeps the language model weight spread from the last anchor state, and \( pushed[d] \) which keeps the binary value indicating if the output label was produced or not. Besides this information, the algorithms also collects in \( L \) and \( O \) the sets of input and output labels of language model edges matching the current lexicon edge. Those sets are used to determine if the output label can be pushed or not.

The function \texttt{pushPrefixEdges} uses those auxiliary values to update newly created edges in the prefix region of the network in order to implement pushing of weights and labels.

As before, \( \epsilon \) input edges in the language are dealt with in lines 2 to 4. Also, edges in the suffix region of the composition network are produced, just as before, in lines 6 and 7.

Edges in the prefix region of the network are computed in lines 8 to 21. The main difference from the previous algorithm is that a prefix edge might be changed later in function \texttt{pushPrefixEdges} due to pushing. Also, if the output label was already pushed, that information is propagated to the destination state (line 17).

If it was not pushed already, then auxiliary variables \( L \), \( O \) and \( \hat{p} \) are updated for the destination state. \( L \) contains the set of labels \( l_g \) of language model edges matching the current lexicon edge \( e_i \), while \( O \) stores the corresponding output labels. If, after matching against all language model edges, the set \( L \) is singleton, then the corresponding composition edges should have the output labels changed from \( \epsilon \) to labels from the set \( O \) (if the language model is ambiguous, multiple edges might have to be created, each one outputting a different member of \( O \)). The
ARCS\((T, q = (q_l, q_g))\)

1. \(A \leftarrow \{\}\)
2. if \(q_l = i_l\)
3. then for each \((q_g, \epsilon, o_g, d_g, w_g)\)
4. do \(A \leftarrow A \cup (q, \epsilon, o_g, (q_l, d_g), w_g)\)
5. for each \(e_l = (q_l, l_l, o_l, d_l, w_l)\)
6. do if \(\text{DESTSET}(e_l) = \Delta_l\)
7. then \(A \leftarrow A \cup \{(q, l_l, \epsilon, (d_l, q_g), w_l)\}\)
8. else if \(q_l = \epsilon\)
9. then \(A' \leftarrow O \leftarrow L \leftarrow \{\}\)
10. \(\hat{p} \leftarrow \tilde{0}\)
11. for each \((q_g, l_g, o_g, d_g, w_g)\)
12. do
13. if \(l_q \in \text{DESTSET}(e_l)\)
14. then \(d \leftarrow (d_l, q_g)\)
15. \(A' \leftarrow A' \cup \{(q, l_l, \epsilon, d, w_l)\}\)
16. if \(\text{pushed}[q]\)
17. then \(\text{pushed}[d] \leftarrow \text{true}\)
18. else \(L \leftarrow L \cup \{l_g\}\)
19. \(O \leftarrow O \cup \{o_g\}\)
20. \(\hat{p} \leftarrow \hat{p} \oplus w_g\)
21. \(A = A \cup \text{PUSHPREFIXEDGES}(A', L, O, \hat{p})\)
22. else for each \((q_g, l_g, o_g, d_g, w_g)\)
23. do
24. if \(o_l = l_g\)
25. then \(d \leftarrow (d_l, d_g)\)
26. if \(\text{pushed}[q]\)
27. then \(A \leftarrow A \cup \{(q, l_l, \epsilon, d, w_l)\}\)
28. else \(\hat{w} \leftarrow w_l \otimes w_g \otimes \text{pushedCost}[q]^{-1}\)
29. \(A \leftarrow A \cup \{(q, l_l, o_g, d, \hat{w})\}\)
30. if \(d_l \neq i_l\)
31. then \(\text{pushed}[d] \leftarrow \text{true}\)
32. return \(A\)

Figure 7.10: Algorithm for creating edges with weight and output label pushing.
weights of language model edges matching with \( e_l \) are accumulated in \( \hat{p} \) (line 20); when the final value of \( \hat{p} \) is known, those edges will have their weight updated.

The transition between prefix and suffix regions of the network occurs in lines 21 to 27. There, the language model component of the composition state will change to the destination state, as before (line 21). If pushing has already produced the output label, it will not be produced now, and, as the language model weight was also fully spread in previous edges, only the lexicon weight will be used (line 23). If label pushing has not occurred yet, then the output label will be produced (line 25). Furthermore, only the portion of the language model weight not yet spread in previous edges (\( w_g \otimes pushedCost[q]^{-1} \)) will be used in the edge weight, together, of course, with the lexicon weight (line 24). The information that the output label was already produced is also propagated to the destination state but only if it is not the initial state (lines 26 and 27).

The composition edges produced by matching a particular lexicon edge \( e_l \) with the language model edges are accumulated in set \( A' \). Only when all those edges are collected do we have the necessary information to perform pushing over edges

```latex
\begin{align*}
\text{pushPrefixEdges}(A', L, O, \hat{p}) & \quad 1 \quad A \leftarrow \{\} \\
& \quad 2 \quad \text{for each } (q, l, o, d, w) \in A' \\
& \quad 3 \quad \text{do if } \hat{p} \neq 0 \\
& \quad 4 \quad \text{then } \hat{w} \leftarrow w \otimes \hat{p} \otimes pushedCost[q]^{-1} \\
& \quad 5 \quad \text{pushedCost}[d] \leftarrow \hat{p} \\
& \quad 6 \quad \text{else } \hat{w} = w \\
& \quad 7 \quad \text{if } L = \{\hat{l}\} \\
& \quad 8 \quad \text{then } pushed[d] \leftarrow \text{true} \\
& \quad 9 \quad \text{for each } \hat{o} \in O \\
& \quad 10 \quad \text{do } A \leftarrow A \cup \{(q, l, \hat{o}, d, \hat{w})\} \\
& \quad 11 \quad \text{else } A \leftarrow A \cup \{(q, l, o, d, \hat{w})\} \\
& \quad 12 \quad \text{return } A
\end{align*}
```

Figure 7.11: Function for fixing prefix edges.
in the prefix region of the network. The edges are updated in function PUSHPREFIXEDGES, shown in Figure 7.11.

PUSHPREFIXEDGES iterates over a set of collected edges. If the edge is a prefix edge (lines 3 to 5, in Figure 7.11), its weight will be updated with the “difference” between the language model weight spread to its origin $q$, and the language model weight spread to its destination $d$. If the edge is a transition or a suffix edge, then its weight will stay unchanged (line 6).

Lines 7 to 11 deal with output label pushing. Pushing is done if all language model edges which matched against $e_l$ had the same input labels (that is, when $L$ is a singleton). When this occurs, edges will be created producing each of the collected output labels.

### 7.4.1 Approximations and restrictions

The presented pushing algorithm resembles traditional language model lookahead in various aspects. One aspect is that only the language model component of weights is pushed. This restriction might be mitigated if the lexicon is pre-pushed. This may lead to a suboptimal distribution of weights in the composition network.

We do not predict this to be a major problem since that even when weights are used in the lexicon, the pronunciation path weights are usually normalized relative to all paths of the same word, and in most cases all such paths are preserved in the composition. Yet, since there is no experience in our research group on the use of pronunciation weighting in the lexicon (mostly because of lack of hand labelled data), no experiments have been done to ascertain the impact of this approximation. All pronunciations in our lexica are assigned the same probability (1.0).

Another similarity with language model lookahead is that pushing is not performed across word boundaries. This restriction, in practice, is not very important, because the main effect of pushing across word boundaries can be achieved if the
language model is itself previously pushed. If we assume that the lexicon contains
pronunciations for all the words, no leftover weight will be available at anchor states
to propagate back across pronunciation paths.

A requirement of “on-the-fly” algorithms is that the computation of the value
pushedCost[d] associated with a particular state d, should always produce the same
value, regardless of which lexicon edge was matched with language model edges for
its computation. This is the main reason why the lexicon weight is not included in
the pushed value, as the existence of lexicon edges with different weights but with
the same destination state d could result in different values of pushedCost[d].

If the sets associated with edges arriving at d were different, then pushedCost[d]
might also be different. If all those edges have \( \epsilon \) output, then all sets will be equal
because they are, by definition, the union of the sets of all edges leaving d. The
situation in which all arriving edges have non-\( \epsilon \) output does not pose any problems
either, because in that case pushedCost[d] will always be \( \bar{1} \). The situation in
which same edges have non-\( \epsilon \) output and others \( \epsilon \) is more problematic. Yet, such
situation would imply that some paths would output different numbers of non-\( \epsilon \)
labels between the initial and the final states of the lexicon (before performing its
closure \( L^* \)). This does not happen in typical lexica, or at least in the lexica we are
interested in.

We can thus summarize the approximations and restrictions of our algorithm:

- Only the language model weight is pushed.
- No pushing is performed across word (pronunciation) boundaries.
- The lexicon closure \( L^* \) must loop through the initial state.
- Every path in the lexicon \( L \) between the initial and the final state must output
  one and only one non-\( \epsilon \) label.
7.5 Minimization

So far, in this chapter, we have presented how a sequential lexicon and language model can be composed for “on-the-fly” generation of a deterministic integrated network for speech recognition. We also showed how this network can be built in order to approximate the optimizations achieved by pushing of weights and output labels.

The other main global optimization operation is minimization; its main effect is to reduce the size of the network. Therefore, it is a very important optimization, specially when a static network is desired.

Minimization is difficult to approximate in an “on-the-fly” algorithm because of its global nature. The difficulty comes from the fact that classical minimization algorithms build equivalence classes of states in which the equivalence if based on the set of paths from each state to a final state being the same, while typical “on-the-fly” algorithms know nothing from that state onwards. They are only required to know at least one path up to a particular state. Nevertheless, the specific structure of the composition transducer allow us to exploit and remove various sources of redundancy.

7.5.1 Minimization of the Language Model

In our efforts to approximate minimization, we start by minimizing the language model in a preprocessing step. The rational is that if the language model is not a minimal deterministic transducer, then large portions of the composition transducer will be redundant. Furthermore, we will assume that anchor states in this composition with the minimal language model are all distinct, in the sense that, if classical minimization is applied to the composition transducer, all anchor states will belong to different equivalence classes. This is clearly not true in general: one example is the extreme case where all words in the lexicon are homophone, and all
edges of the language model WFST have output label $\epsilon$.

### 7.5.2 Suffix Sharing

One other source of redundancy in the composition network is the fact that pronunciation suffixes of words arriving at the anchor state are not completely shared. Not even when the lexicon is a minimal deterministic WFST. In the following discussion, we will consider the suffix of a word to consist of all states and edges in the composition transducer which are after the production of its output label and before the next anchor state.

Figure 7.12 illustrates a sample language model fragment $G_2$ that, when composed with the minimal lexicon $L_{det}$, shown in Figure 7.13, generates the network shown in 7.14.

The sharing that takes place in the edge connecting states 3 and 4 is due to the
structure of the lexicon \( WFST \). When the lexicon is deterministic and the identity of the word is produced in the last edge of the pronunciation, no suffix sharing may take place among words arriving at the same anchor state. If the lexicon is deterministic and minimal, as the one shown, then some sharing of suffixes may occur, even among different words.

However, as the example in Figure 7.14 shows, the suffix sharing will not be, in general, complete. In order to closely approximate minimization, we would like to obtain the network shown in Figure 7.15, in which state 6 is merged with state 2.

The redistribution of weights and output labels resulting from pushing allows further sharing of suffixes in the composition network.

In order to achieve better suffix sharing than possible with a deterministic and minimal lexicon \( WFST \) \( (L_{det}) \), we will also employ an equivalent “suffix-sharing” right-sequential lexicon \( WFST (L_{suf}) \), which has the topology illustrated in Figure 7.16. The topology is such that \( \text{reverse}(L_{suf}) \) is sequential, and all non-\( \epsilon \) output labels leave the initial state.

The idea behind our suffix-sharing algorithm is to use \( L_{suf} \) as a guide to generate
the suffixes. One problem we face after producing an output label \( o \), and arriving at lexicon state \( q_{det} \), is knowing which state \( q_{suf} \) in \( L_{suf} \) to use in order to continue the path.

The first step to establish correspondences between states in \( L_{suf} \) and \( L_{det} \) is to compose them \( L_{comp} = \pi_1(L_{det}) \circ L_{suf} \), as illustrated in Figure 7.17. \( L_{comp} \) is, by construction, equivalent to both \( L_{suf} \) and \( L_{det} \).

Using \( L_{comp} \), the correspondence problem can be solved by selecting all suffix states \( q_{suf} \) such that the state \( (q_{det}, q_{suf}) \) exists in a path which outputs \( w \) in \( L_{comp} \). The placement of the non-\( \varepsilon \) output labels in \( L_{suf} \) after the initial state makes the search for such paths trivial.

The following suffix sharing algorithm generates the desired composition.

**Algorithm**

The suffix sharing composition algorithm is shown in Figures 7.18, 7.19 and 7.20. In the algorithm, the subscripts \(_{det} \), \(_{suf} \) and \(_{comp} \) are used to distinguish the lexica. For example, \( i_{det} \) is the initial state of the deterministic lexicon, \( E_{det} \) is its set of edges, \( Q_{suf} \) is the set of states of \( L_{suf} \), and \( F_{suf} \) is its set of final states.

The algorithm is similar to the previous versions presented. The main difference is that the lexicon component \( q_l \) of a composition state \( (q_l, q_o) \) can be a state from
either $L_{det}$ or $L_{suf}$. No overlap is assumed in the set of states of the different lexica: $Q_{suf} \cap Q_{def} = \emptyset$, consequently the array $pushed[q]$ is no longer needed, since it can be determined that the output label was not yet produced by testing if $q_l \in Q_{def}$.

In lines 2 to 4 of Figure 7.18, the algorithm deals with $\epsilon$ edges in the language model. In lines 5 to 9 it deals with edges that belong to the suffix region of the network. Those edges are just replicated from the suffix lexicon until the end of the pronunciation is found (line 7); there the algorithm switches back to using the deterministic lexicon (line 8).

Lines 12 to 22 deal with $\epsilon$ output edges in the deterministic lexicon. As before, non-coaccessible paths are avoided by only considering language model edges which match the set tagging the lexicon edge. Also as before, $L$ accumulates the inputs of the matching language model edges, in order to determine if the output label should be produced. In that event, the destination state of the new edge should be changed to reflect the switch to the suffix lexicon, this is why the destination state $d_q$ is now stored with the output label $o_q$ in $O$. The function $\text{PUSHPREFIXEDGES}$, shown in Figure 7.19, must now change the destination state of edges which produce output labels (lines 8 to 10).

115
\textbf{ARCS}(T, q = (q_l, q_g))

\begin{enumerate}
\item $A \leftarrow \{\}$
\item if $q_l = i_{det}$
\item \hspace{1em} then for each $(q_g, \epsilon, o_g, d_g, w_g)$
\item \hspace{2em} do $A \leftarrow A \cup \{(q, \epsilon, o_g, (q_l, d_g), w_g)\}$
\item \hspace{1em} if $q_l \in Q_{suf}$
\item \hspace{2em} then for each $e_l = (q_l, l_l, o_l, d_l, w_l) \in E_{suf}$
\item \hspace{3em} do if $d_l \in F_{suf}$
\item \hspace{4em} then $A \leftarrow A \cup \{(q, l_l, \epsilon, (i_{det}, q_g), w_l)\}$
\item \hspace{4em} else $A \leftarrow A \cup \{(q, l_l, \epsilon, (d_l, q_g), w_l)\}$
\item \hspace{1em} else for each $e_l = (q_l, l_l, o_l, d_l, w_l) \in E_{det}$
\item \hspace{2em} do if $o_l = \epsilon$
\item \hspace{3em} then $A' \leftarrow O \leftarrow L \leftarrow \{\}$
\item \hspace{4em} $\hat{p} \leftarrow \emptyset$
\item \hspace{4em} for each $(q_g, l_g, o_g, d_g, w_g)$
\item \hspace{5em} do
\item \hspace{6em} if $l_g \in \text{DESTSET}(e_l)$
\item \hspace{7em} then $d \leftarrow (d_l, q_g)$
\item \hspace{7em} $A' \leftarrow A' \cup \{(q, l_l, \epsilon, d, w_l)\}$
\item \hspace{7em} $L \leftarrow L \cup \{l_g\}$
\item \hspace{7em} $O \leftarrow O \cup \{(o_g, d_g)\}$
\item \hspace{7em} $\hat{p} \leftarrow \hat{p} \oplus w_g$
\item \hspace{6em} $A \leftarrow A \cup \text{PUSHPREFIXEDGES}(A', L, O, \hat{p})$
\item \hspace{5em} else for each $(q_g, l_g, o_g, d_g, w_g)$
\item \hspace{6em} do
\item \hspace{7em} if $o_l = l_g$
\item \hspace{8em} then for each $d' \in \text{FINDINSUFFIXLEX}(d_l, o_l)$
\item \hspace{9em} do $\hat{w} \leftarrow w_l \otimes w_g \otimes \text{pushedCost}[q]^{-1}$
\item \hspace{9em} $A \leftarrow A \cup \{(q, l_l, o_l, (d', d_g), \hat{w})\}$
\item \hspace{5em} end for
\item \hspace{4em} end if
\item \hspace{3em} end for
\item \hspace{2em} end do
\item \hspace{1em} end if
\item \hspace{1em} end do
\item \hspace{1em} end if
\item \hspace{1em} end for
\item \hspace{1em} end for
\item return $A$
\end{enumerate}

Figure 7.18: Suffix sharing algorithm.
**Figure 7.19: Suffix sharing algorithm (cont.)**

**PUSHPREFIXEDGES**(A’, L, O, ̂p)
1. A ← {}
2. for each (q, l, o, (d_l, d_q), w) ∈ A’
3. do if ̂p ≠ 0
4. then ̂w ← w ⊗ ̂p ⊗ pushedCost[q]^{-1}
5. pushedCost[(d_l, d_q)] ← ̂p
6. else ̂w ← w
7. if L = {l}
8. then for each d’ ∈ FINDINSUFFIXLEX(d_l, ̂l)
9. do for each (ɔ, ̂d) ∈ O
10. do A ← A ∪ {(q, l, ɔ, (d’, ̂d), ̂w)}
11. else A ← A ∪ {(q, l, o, (d_l, d_q), ̂w)}
12. return A

**Figure 7.20: Function FINDINSUFFIXLEX.**

**FINDINSUFFIXLEX**(q, l)
1. S ← {}
2. R ← {}
3. for each (i_{comp}, l_l, ɔ_l, d, w_l) ∈ E_{comp}
4. do if ɔ_l = l
5. then ENQUEUE(S, d)
6. while S ≠ {}
7. do (q_{det}, q_{suf}) ← HEAD(S)
8. DEQUEUE(S)
9. if q_{det} = q
10. then if q_{suf} ∈ E_{suf}
11. then R ← R ∪ {i_{det}}
12. else R ← R ∪ {q_{suf}}
13. else for each ((q_{det}, q_{suf}), l_l, ɔ_l, d, w_l) ∈ E_{comp}
14. do if d ∉ S
15. then ENQUEUE(S, d)
16. return R
Lines 23 to 28 deal with non-$\epsilon$ output edges in the deterministic lexicon: when a matching edge is found in the language model (line 25), the algorithm searches for corresponding states in the suffix lexicon (line 26) in order to switch to using the suffix lexicon, producing multiple transitions if necessary.

The search for the set of suffix states equivalent to a state in the deterministic lexicon is performed in function \texttt{FINDIN\_SUFFIXLEX} (shown in Figure 7.20). It is implemented as a simple graph traversal algorithm over the composition lexicon transducer. To accelerate the search, the results of this function can be memoized in a hash table.

### 7.5.3 Tail Sharing

The algorithm presented in the previous section addresses the problem of minimizing the composition network. Yet, its ability to share suffixes for different words is useless when n-gram language models are used, for the edges entering an n-gram language model are either backoff edges labelled with $\epsilon$ or have all the same label. When n-gram language models are used, sharing the suffixes of the same word is enough. In this section, we present a technique for sharing suffixes which is particularly effective when n-gram language models are used, but does not require a suffix lexicon.

The method proposed here resembles Brugnara and Cettolo’s \textit{tail sharing} technique [23] (which we described in section 6.1) because it allows the sharing of pronunciation suffixes by instances of the same word. Yet, it is more general since it shares pronunciation suffixes regardless of their topology; in particular, it is not limited to sharing linear sequences of phones.

Instead of using a different lexicon \textit{WFST} to guess the structure of suffixes, our tail sharing algorithm simply reuses paths taken by other instances of suffixes of the same word going into the same anchor state.

Figure 7.21 illustrates a typical fragment of an n-gram language model; Figure
Figure 7.21: Sample language model $G_3$.

Figure 7.22: Composition $L^* \circ G_3$ with pushing.

Figure 7.23: Desired composition $L^* \circ G_3$, with complete sharing of suffixes.
7.22 shows the composition network obtained by the pushing algorithm; and Figure 7.23 shows the desired network.

The modification of the pushing algorithm to implement sharing of tails consists of using a hash table $H$ containing states from the composition transducer. These states are indexed by a tuple $(q_l, o_l, (i_l, d_g))$ where $q_l$ is a lexicon state, $o_l$ is the word whose suffix is being shared, and $(i_l, d_g)$ is the next anchor state. Whenever a state $(q_l, q_g)$ is generated such that $q_l$ is a prefix state but the non-$\epsilon$ output label $o_l$ was already produced, a lookup is made in the hash table $H[q_l, o_l, (i_l, d_g)]$ to find a state on the same position in the lexicon (meaning, on a path of the same word and going to the same anchor state). If such a state is found, it is used instead of $(q_l, q_g)$, otherwise $(q_l, q_g)$ is entered in the hash table. There is an apparent ambiguity in indexing the hash table with both the word and the destination state, but this is necessary in order to avoid incorrect behavior when using language models which allow different words to enter the same state.

In Figure 7.24, we see the effect of the tail-sharing composition algorithm. Assuming that state (1,1) is generated first and is thus present in the hash table, when state (0,2) is expanded and state (1,2) generated, the algorithm detects that (1,2) is equivalent to (1,1), meaning $H[1, \text{ATO}, (0,3)] = (1,1)$, and uses it as destination of the edge leaving (0,2). State (1,2) is generated, but ends up being ignored as it has no incoming or outgoing edges.
7.6 Optimizing the Algorithm

The algorithms presented so far are very inefficient as their running time is proportional to the product of the number of edges leaving the lexicon and the language model states. This quadratic complexity is the worst case of the composition algorithm, where each lexicon edge matches every language model edge. Nevertheless, in the instances of the problem we are interested in, a lexicon edge tends to match with very few, if any, of the language model edges. Previously this sparseness was not exploited. In this section, we will show how the algorithms can be improved to exhibit a running time proportional to the number of edges produced in the composition WFST.

The inefficiency of the previous algorithms lies in the embedded loops which match lexicon edges to language model edges. In a classical composition algorithm, we can easily improve this matching if the edges leaving the language model and the lexicon state are pre-sorted and kept in a suitable data structure, such as a sorted array. Figure 7.25 shows how the generation of edges for the $\epsilon$-free composition algorithm can thus be implemented.

Functions $\text{SEEKINPUT}(E, i, l)$ and $\text{SEEKOUTPUT}(E, i, l)$ return the first index $i$, such that $i > i$, and the input label (or output label, respectively) of the edge $E[i]$ is equal or greater than $l$. These functions can have a complexity of $O(\log |E|)$ if implemented using binary search or of $O(1)$ if an index table is used. Even if these functions are implemented using linear search, and assuming that only one language model edge matches a given lexicon edge (meaning that the while loop in lines 13 to 17 iterates at most once for each lexicon edge), the worst case complexity of the algorithm is only $O(|E_l| + |E_d|)$.

The problem we face when trying to port this idea to our algorithm is that, in our case, the lexicon edges should be sorted by their tagging set. Yet, a complete order for those sets cannot always be found, given the large class of lexicon WFSTs the algorithm should work with.
ARCS($T, q = (q_l, q_d)$)
  
  1  $A \leftarrow \{\}$
  
  2  $E_l \leftarrow$ ARCS($T_l, q_l$) /*Sorted by output-label*/
  
  3  $E_g \leftarrow$ ARCS($T_g, q_g$) /*Sorted by input-label*/
  
  4  $i \leftarrow j \leftarrow 0$
  
  5  while $i < |E_l| \text{ and } j < |E_g|$
  
  6  do $(q_l, l_l, o_l, d_l, w_l) \leftarrow E_l[i]$
  
  7  $(q_g, l_g, o_g, d_g, w_g) \leftarrow E_g[j]$
  
  8  if $o_l < l_g$
  
  9  then $j \leftarrow$ SEEKOUTPUT($E_l, j, l_g$)
  
  10  else if $o_l > l_g$
  
  11  then $i \leftarrow$ SEEKINPUT($E_g, i, o_l$)
  
  12  else $jj \leftarrow j$
  
  13  while $jj < |E_g| \text{ and } o_l = l_g$
  
  14  do $A \leftarrow A \cup \{(q_l, l_l, o_l, (d_l, d_g), w_l \otimes w_g)\}$
  
  15  $jj \leftarrow jj + 1$
  
  16  if $jj < |E_g|$
  
  17  then $(q_g, l_g, o_g, d_g, w_g) \leftarrow E_g[jj]$
  
  18  $i \leftarrow i + 1$
  
  return $A$

Figure 7.25: Efficient generation of edges for $\epsilon$-free composition.
Assuming a complete order over labels (e.g., the lexicographic order of their strings), two sets of labels $W_1$ and $W_2$ can be ordered if either $\forall w_1 \in W_1, w_2 \in W_2 \ w_1 < w_2$ or $\forall w_1 \in W_1, w_2 \in W_2 \ w_2 < w_1$. It two sets cannot be thus ordered, we say that they overlap.

If the prefix region of the lexicon $WFST$ is organized as a tree (meaning that the in-degree of prefix states is one), and each word has only one pronunciation, then an order can be found by traversing the lexicon tree in post-order, assigning consecutive identification codes (IDs) to non-$\epsilon$ output labels as they are found. With those IDs, the sets associated with edges leaving a prefix state will consist of non-overlapping ranges, which can now be sorted. To exemplify, Figure 7.26 shows an appropriate lexicon $WFST$ with edges tagged with ranges.

This constraint is less severe than the tree lexicon structure which prevails in traditional speech recognition systems, because this particular topology is only required in the prefix region of the lexicon $WFST$. From the point of view of speech recognition, it is almost irrelevant. It only gains importance when a large number of pronunciation variants are used. Nevertheless, we feel it to be an inadequate constraint, and that the optimized algorithm should have no more constraints than the previous one.

The solution we found consists of a preprocessing step over the lexicon $WFST$, which splits sets and replicates edges so that sets can be ordered. This process is exemplified in Figures 7.27 and 7.28, where $A, B, C$ and $D$ are arbitrary words and
the usual lexicographic order of words is assumed.

In the example, we observe that the set \{A, C, D\} is overlapping with set \{B\}. After preprocessing, the sets \{A\}, \{C, D\} and \{B\} do not overlap and can be sorted. The preprocessing consists of splitting sets to avoid overlaps and replicating the edges whose sets were split.

After this preprocessing step, lexicon edges can be sorted by tagging set.

Now this lexicon structure presents us with a problem. Our pushing algorithm assumes that all prefix edges arriving at the same state are tagged with the same set. This is no longer the case.

This assumption is used in the inner loop which uses auxiliary variables \(L, O\), and \(\hat{p}\) to push the weight or label of a given edge. Now, because each lexicon edge only provides partial information, those values are accumulated at the destination state \(d\), in variables \(L[d], O[d]\) and \(\hat{p}[d]\). Prefix edges are only corrected after the creation of all edges, in a slightly modified version of \textsc{pushprefixedges}.
An additional problem is that the accumulation of weights at destination states $\hat{\rho}[d]$ is only correct when the $\oplus$ operation of the semi-ring is idempotent.

When other semi-rings are used, the value $\hat{\rho}$ will be dependent on the number of edges which arrive at that state $d$ (in the original lexicon).

Figure 7.29 illustrates the problem with a fragment of a lexicon. Let us suppose that the lexicon states are matched against a language model state $p$ origin of an edge with input label $A$ and cost 0.3, in the probability semi-ring. Then, the pushed cost $\hat{\rho}[(3, p)]$ would be $0.6 = 0.3 + 0.3$ when state $(1, p)$ is expanded and 0.3 when $(2, p)$ is expanded.

The problem is solved with the creation of new states in the lexicon, so that each tagged edge which leaves the same lexicon state has a different destination. This operation is done in the lexicon preprocessing step. Figure 7.30 shows the effect of this processing over the lexicon fragment of Figure 7.29. Now the algorithm will produce the pushed cost $\hat{\rho}[(3, 5)]$ of 0.3 when expanding either $(1, 5)$ or $(2, 5)$.

125
<table>
<thead>
<tr>
<th>$q$</th>
<th>$X[q]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 7.2: State translation table $X$ for the example of Figure 7.30.

When new states are created, they are entered in a translation table $X$ which allows the algorithm to recover the original lexicon structure during composition. Table 7.2 shows the relevant portion of Table $X$ for this example.

Algorithm

In Figures 7.31, 7.32 and 7.33, we show the optimized version of the algorithm which performs pushing and suffix sharing.

It is assumed that the lexicon edges $E_{det}$ are sorted first by output label, and then by tagging set, in such a way that edges with $\epsilon$ output appear before edges with non-$\epsilon$ output. The language model edges $E_q$ are sorted by input label.

Edges after the production of the output label and before the next anchor state are generated in lines 3 to 7 by replication of the suffix lexicon.

Prefix edges are generated in lines 8 to 33. The algorithm uses the order of tagging sets and language model input labels to quickly avoid non matching edges (lines 14 to 20). In lines 21 to 31, the edges which do match are processed. The translation table $X$ is used when the destination state is generated (line 23), and, as an optimization, the auxiliary variables $L,O$ and $\hat{p}$ are only accumulated when the lexicon destination $d_j$ is not one of the states generated during lexicon preprocessing (line 25). The auxiliary pushing variables are indexed with the destination state of the edge.
\[
\text{ARCS}(T, q = (q_l, q_g))
\]
\begin{algorithm}
1. \forall d \in Q, O[d] \leftarrow \{\}, L[d] \leftarrow \{\}, \hat{p}[d] \leftarrow \emptyset
2. \quad A \leftarrow \{\}
3. \quad \textbf{if } q_l \in Q_{\text{sub}}
4. \quad \quad \textbf{then for each } e_l = (q_l, l_l, o_l, d_l, w_l) \in E_{\text{sub}}
5. \quad \quad \quad \textbf{do if } d_l \in E_{\text{sub}}
6. \quad \quad \quad \quad \textbf{then } A \leftarrow A \cup \{(q_l, l_l, e, (d_l, q_l), w_l)\}
7. \quad \quad \quad \quad \textbf{else } A \leftarrow A \cup \{(q_l, l_l, e, (d_l, q_g), w_l)\}
8. \quad \quad \textbf{else } E_{\text{det}} \leftarrow \text{ARCS}(T_{\text{det}}, q_{\text{det}}) \quad */\text{Sorted by tagging set}*/
9. \quad \quad E_g \leftarrow \text{ARCS}(T_g, q_g) \quad */\text{Sorted by input-label}*/
10. \quad \quad A' \leftarrow \{\}
11. \quad \quad i \leftarrow j \leftarrow 0
12. \quad \quad \hat{\mu} \leftarrow \text{SEEKOUTPUT}(E_{\text{det}}, 0, \epsilon + 1)
13. \quad \quad \textbf{while } i < |E_{\text{det}}| \text{ and } i < \hat{\mu} \text{ and } j < |E_g|
14. \quad \quad \quad \textbf{do } (q_l, l_l, o_l, d_l, w_l) \leftarrow E_{\text{det}}[i]
15. \quad \quad \quad (q_g, l_g, o_g, d_g, w_g) \leftarrow E_g[j]
16. \quad \quad \quad S \leftarrow \text{DESTSET}(E_{\text{det}}[i])
17. \quad \quad \quad \textbf{if } \max(S) \leq l_g
18. \quad \quad \quad \quad \textbf{then } j \leftarrow \text{SEEKTAGSET}(E_{\text{det}}, j, l_g)
19. \quad \quad \quad \quad \textbf{else if } \min(S) > l_g
20. \quad \quad \quad \quad \quad \textbf{then } i \leftarrow \text{SEEKINPUT}(E_g, i, \min(S))
21. \quad \quad \quad \quad \textbf{else } j j \leftarrow j
22. \quad \quad \quad \quad \textbf{while } j j < |E_g| \text{ and } \alpha \in S
23. \quad \quad \quad \quad \quad \textbf{do } d \leftarrow (X[d_l], q_g)
24. \quad \quad \quad \quad \quad \quad A' \leftarrow A' \cup \{(q, l_l, e, d, w_l)\}
25. \quad \quad \quad \quad \quad \quad \textbf{if } X[d_l] = d_l
26. \quad \quad \quad \quad \quad \quad \quad L[d] \leftarrow L[d] \cup \{l_q\}
27. \quad \quad \quad \quad \quad \quad \quad O[d] \leftarrow O[d] \cup \{\omega_y, d_g\}
28. \quad \quad \quad \quad \quad \quad \quad \hat{p}[(d_l, q_g)] \leftarrow \hat{p}[(d_l, q_g)] \oplus w_g
29. \quad \quad \quad \quad \quad \quad j j \leftarrow j j + 1
30. \quad \quad \quad \quad \textbf{if } j j < |E_g|
31. \quad \quad \quad \quad \quad \quad (q_g, l_g, o_g, d_g, w_g) \leftarrow E_g[j j]
32. \quad \quad \quad \quad \quad \quad i \leftarrow i + 1
33. \quad \quad \quad \quad A \leftarrow \text{PUSHPREFIXEDGES}(A', L, O, \hat{p})
34. \quad \quad \textbf{if } q_l = i_l
35. \quad \quad \quad \textbf{then } j \leftarrow \text{SEEKINPUT}(E_g, 0, \epsilon)
36. \quad \quad \quad \quad \textbf{if } j < |E_g|
37. \quad \quad \quad \quad \quad \quad (q_g, l_g, o_g, d_g, w_g) \leftarrow E_g[j]
38. \quad \quad \quad \quad \textbf{while } j < |E_g| \text{ and } l_g = \epsilon
39. \quad \quad \quad \quad \quad \quad \textbf{do } A \leftarrow A \cup \{(q, e, o_y, (q_l, d_g), w_g)\}
\end{algorithm}

Figure 7.31: Algorithm for creating edges with weight and output label pushing.
(continues from fig. 7.31)

\[ j \leftarrow \text{SEEKINPUT}(E_g, 0, \epsilon + 1) \]
\[ i \leftarrow \text{SEEKOUTPUT}(E_{det}, 0, \epsilon + 1) \]
\[ \text{while } i < |E_{det}| \text{ and } j < |E_g| \]
\[ \text{do } (q_l, l_l, o_l, d_l, w_l) \leftarrow E_{det}[i] \]
\[ (q_g, l_g, o_g, d_g, w_g) \leftarrow E_g[j] \]
\[ \text{if } o_l < l_g \]
\[ \quad \text{then } j \leftarrow \text{SEEKOUTPUT}(E_{det}, j, l_g) \]
\[ \quad \text{else } \text{if } o_l > l_g \]
\[ \quad \quad \text{then } i \leftarrow \text{SEEKINPUT}(E_g, i, o_l) \]
\[ \quad \text{else } jj \leftarrow j \]
\[ \quad \text{while } jj < |E_g| \text{ and } o_l = l_g \]
\[ \quad \text{do for each } d_{suf} \in \text{FINDIN\text{SUFFIXLEX}}(d_l, o_l) \]
\[ \quad \quad d \leftarrow (d_{suf}, d_g) \]
\[ \quad \quad A \leftarrow A \cup \{(q_l, l_l, o_l, d, w_l \odot w_g \odot \text{pushedCost}[q]^{-1})\} \]
\[ \quad \quad jj \leftarrow jj + 1 \]
\[ \quad \text{if } jj < |E_g| \]
\[ \quad \quad \text{then } (q_g, l_g, o_g, d_g, w_g) \leftarrow E_g[jj] \]
\[ \quad i \leftarrow i + 1 \]
\[ \text{return } A \]

Figure 7.32: Algorithm for creating edges with weight and output label pushing.
PUSHPREFIXEDGES($A', L, O, \hat{p}$)
1. $A \leftarrow \{\}$
2. for each $(q, l, o, d, w) \in A'$
3.   do
4.     $(d_l, d_o) \leftarrow d$
5.     if $\hat{p}[d] \neq \emptyset$
6.       then $\hat{w} \leftarrow w \otimes \hat{p}[d] \otimes \text{pushedCost}[q]^{-1}$
7.       pushedCost[d] $\leftarrow \hat{p}[d]$
8.     else $\hat{w} \leftarrow w$
9.     if $L[d] = \{\hat{l}\}$
10.    then for each $d' \in \text{FINDINSUFFIXLEX}(d_l, \hat{l})$
11.       do for each $(\hat{o}, \hat{d}) \in O[d]$
12.         do $A \leftarrow A \cup \{(q, \hat{l}, \hat{o}, (d', \hat{d}), \hat{w})\}$
13.     else $A \leftarrow A \cup \{(q, l, o, d, w)\}$
14. return $A$

Figure 7.33: Function for fixing edges.

After all lexicon prefix edges have been matched against language model edges, the composition edges collected in $A'$ are processed to implement pushing (line 33).

Language model edges with $\epsilon$ input are processed in lines 34 to 39.

Non-$\epsilon$ edges in the lexicon are processing in lines 41 to 58 (Figure 7.32). This portion of the algorithm is similar to the $\epsilon$-free composition algorithm shown before.

In Figure 7.33, the new version of the function PUSHPREFIXEDGES is shown. The main difference from previous versions is that auxiliary variables $L$, $O$ and $\hat{p}$ are indexed with destination states.

7.7 Network Factorization

We denote by factorization operations or algorithms which, given a transducer $T$, produce transducers $T_1$, $T_2$ such that $T = T_1 \circ T_2$.

Factorization is used to obtain better time or space efficiency. Among the factorization methods used to improve the time efficiency are bi-machines, which were mentioned in Section 3.6, and that factorize a transducer $T$ into the composition
of a left-sequential and a right-sequential transducers. Another example is the method proposed by Kempe [78], which factorizes an ambiguous transducer into the composition of a sequential transducer with a transducer containing some left-over ambiguity. An interesting property of the decomposed transducers is that their composition can be performed without generating non-coaccessible states, hence avoiding the need for backtracking.

In speech recognition, factorization has been used mainly to reduce the size of the integrated search network \( N \).

This network, which results from the composition of the acoustic model \( WFST \) \( H \) with the composition of the lexicon with the language model (and eventually other components), has very long sequences of states forming “linear paths”, where each state has only one successor and eventually a self-loop edge.

Mohri and Riley [103] proposed the factorization of the network such that \( N = F \circ N' \), where \( N' \) results from replacing each linear path in \( N \) with an identifying edge, and \( F \) converts linear paths into identifying edges. They were able to achieve a 4 times reduction of the network size. The identifying edges were expanded in run time by the decoder, which conceptually performed the composition \( F \circ N' \).

A different approach was taken by Dolfing and Hetherington [46] to reduce the size of the integrated network in runtime. They proposed the factorization of the language model \( G \) into two language model \( WFSTs \): a small language model \( G_s \), and \( G_{\text{diff}} \) which has the same structure as \( G \), but whose weights are the difference from similar weights in \( G_s \). \( G_s \) was used to build an optimized static search state which has composed with \( G_{\text{diff}} \) in runtime.

### 7.7.1 “On-the-Fly” Factorization

In order to obtain a smaller network, even when generating it “on-the-fly”, the composition cascade \( H^* \circ L^* \circ G \) can be associated as \( (H^* \circ L^*) \circ G \) instead of \( (H^* \circ (L^* \circ G)) \). Since in most cases \( H^* \circ L^* \) satisfies the restrictions imposed by
the specialized composition algorithm on the structure of the lexicon, it can be used directly by the specialized composition algorithm. Linear paths in $H^* \circ L^*$ can be factorized such that $H^* \circ L^* = F^* \circ H L^*$, where $F^*$ contains all linear paths. The specialized composition algorithm can then be used to compose $H L^* \circ G$ in runtime. Besides space, one additional advantage of this approach is that if $H^* \circ L^*$ is determinized and minimized before being factorized, the final network will be optimized at the state level, which, according to [103] and [45], provides a significant performance boost when tied-state context-dependent acoustic units are used. We have not explored this aspect of factorization yet because our recognition system is based on context independent units.

A simpler but effective factorization alternative consists of factorizing $H$, in which case each factorized linear path will resemble an acoustic unit $H M M$ in a traditional decoder. In [28] we opted for this last approach and still managed a reduction of about 2.5 times in the number of states.

### 7.8 Experimental Results

We performed various experiments to compare the search networks produced by the proposed algorithms with the network generated with Mohri et al.’s explicit weighted determinization and pushing algorithms.

We also performed other experiments to evaluate the effect of various optimizations of the component transducers on the performance of our algorithms.

#### 7.8.1 Comparison with the explicit determinization approach

In order to evaluate the effectiveness of the approximations introduced by our algorithms, we decided to compare them with the networks generated by Mohri et al.’s explicit weighted determinization and pushing algorithms.
One problem we faced when trying to apply Mohri and Riley’s pushing algorithm was that it is cannot be applied to some WFSTs. In particular, the WFSTs which approximate n-gram language models can have loops with negative weights which arise from backoff transitions. The existence of a negative weighted loop prevents the pushing algorithm from finding the best cost to final states since a better cost can always be found by iterating the loop once more; consequently, the cost to a final state of some nodes along paths with negative weighted loops will be $-\infty$.

In [104] the manipulation of the word insertion penalty is proposed to control similar cases. This can be done by adding a penalty weight to every language model edge, pushing the model, and then removing the penalty weight. While this manipulation is sometimes effective, and preserves the total cost of each path, its effect on performance is not clear.

Alternatively, we can replace the weight of negative edges by 0, perform pushing and then add back the previous value to each changed edge. This is implemented as factorizing the language model $G$ into two transducers $\hat{G} \circ G'$, where $\hat{G}$ is the language model without negative weights, and $G'$ restores the respective negative weights. The pushing operation becomes $\text{push}(\hat{G}) \circ G'$. This manipulation also preserves the total cost of each path, and its impact on performance is also not clear.

To avoid the stated problems, and to fairly compare the approaches, we used a pruned language model which did not contain negative weighted edges.

The experiments were done in the Alert Broadcast News domain with a 57k lexicon, and a pruned 3-gram language model with 118,629 n-grams.

In Table 7.3, we show the size of the composition network obtained with weighted determinization, minimization and pushing using the log probability semi-ring. The table also shows the size of the network obtained using our suffix-sharing weight-pushing algorithm. The deterministic lexicon WFST was minimized and
<table>
<thead>
<tr>
<th>Network</th>
<th>States</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min \text{ push det}(L \circ G)$</td>
<td>150,930</td>
<td>237,316</td>
</tr>
<tr>
<td>share suffixes log prob</td>
<td>150,980</td>
<td>237,401</td>
</tr>
</tbody>
</table>

Table 7.3: Dimension of search networks.

Figure 7.34: Explicit determinization vs suffix sharing algorithm.

the language model was pushed and minimized. Both the composition algorithm and the language model pushing employed the $\log \text{ prob}$ semi-ring.

We can see that the size of the networks is almost the same; our approximation has less than 0.1% more states or edges than the explicit determinization approach.

Figure 7.34 shows that the performance of both algorithms is very, very similar, and that our approach is a very good approximation to explicit determinization and offline optimization of the search network. The inflexion observed in the asymptotic evolution of the performance curve is a consequence of the weight pushing algorithm. We have observed similar behavior in most experiments, and it can also
be seen in the results of other researchers, for example in [104].

7.8.2 Evaluating the relative effect of optimizations

In this section we describe experiments done to ascertain the impact of various optimizations of both the component transducers and the specialized composition algorithm on the performance of the resulting search spaces.

Suffix and Tail Sharing

Some experiments were performed on the Alert domain using a large 4-gram language model WFST with 38M edges and 16M states. Their purpose was to ascertain the impact on performance of the various approximations to minimization when a large language model is used to build the search space “on-the-fly”. In this experiment, the language model was not pushed and the algorithms used the tropical semi-ring.

In Figure 7.35, we see the performance of three approximations: the first one merely uses a deterministic minimal lexicon with the pushing version of composition, the second one uses also tail sharing, and the third uses suffix sharing instead.

Surprisingly, the performance of the suffix sharing version is significantly worse than the others, while the tail sharing version is only slightly better than the simpler pushing composition algorithm. The worse behavior of the suffix version is justified by the computational overhead of the algorithm since, as can be seen in Figure 7.36, it is the version which generates less active states per frame. This figure also shows that the impact of the approximations to minimization are very small relative to the pushing composition algorithm when using a minimal lexicon.

Pushing

Our algorithm approximates pushing locally in the search network using language model lookahead when spreading weights through pronunciation paths. It relies on
Figure 7.35: Performance of various minimization approximations.

Figure 7.36: Average number of active states per frame for various minimization approximations.
pre-pushing the language model to approximate global pushing. Since the choice of the semi-ring has a very strong impact on performance [104], we performed some experiments to ascertain that impact. Furthermore, since we can independently control the choice of semi-ring at the local level and at the language model level, 6 different recognition experiments were conducted where on one hand, the language model was not pushed, pushed in the tropical semi-ring, or pushed in the log probability semi-ring, and on the other hand the composition algorithm used either log probability or tropical pushing.

The recognition experiments using the same small language model used in Section 7.8.1 are shown in Figure 7.37. Figure 7.38 shows the same experiments performed in a medium size language model with 3.7 million n-grams.

Observing the figures, it is clear that the worse behavior is obtained when the language model is pushed with either semi-ring and the composition algorithm used the tropical semi-ring. Classical language model lookahead (corresponding to not pushing the language model and composing with the tropical semi-ring) has a relatively bad behavior at lower beams, but it has the best behavior when higher beams are used. Composing with the log probability semi-ring provides the best behavior at low beams, while at higher beams the choice of language model semi-ring is important; pushing with the log probability semi-ring is the the best choice, and not pushing has a negative impact.

7.8.3 Overhead of “On-the-fly” Generation of the Search Space

In this section we describe some experiments done to ascertain the impact of generating the search space “on-the-fly” versus the use of a pre-compiled static search network.

The experiments were done in the Alert domain using the 57k lexicon and a trigram language model with 3.7M n-grams.
Figure 7.37: Impact of pushing approximations in a small language model.

Figure 7.38: Impact of pushing approximations in a medium language model.

137
Figure 7.39: “On-the-Fly” vs. Static network.

Figure 7.39 shows the performance of the pushing algorithm with or without tail-sharing, and using a static network or creating the network “on-the-fly”.

As expected we observe an overhead when the network is created “on-the-fly”. In order to measure this overhead, we computed the ratio between the time required by the “on-the-fly” and the time required by the static versions, and plotted the evolution of this ratio against the WER. As seen in Figure 7.40, the overhead starts at over 30% at low beams, but quickly becomes lower than 10%. We also observe that the tail-sharing version of the algorithm has a higher overhead.

7.8.4 Impact of the Size of Language Models

In order to investigate the scalability of the algorithm to larger language models, some experiments were performed were performed on the BD-Público domain with a 27k lexicon and various language models.

Language models of various sizes were created using cutoffs of 10, 5 and 2. As
can be seen in Figure 7.41, although there is a negative impact of the larger language model at lower beams, the better information available with larger models is used to easily outperform the smaller ones. The worse performance at lower beams is due to the existence of more paths in the larger language models which compete with backoff paths. The figure shows that, if allowed by the available memory, there is much to be gained by using a detailed language model in a single pass decoder.

7.8.5  Application in a Very Large Task

The "on-the-fly" algorithm described in this chapter was integrated in the large broadcast news transcription system Alertmedia, which uses a 4-gram language model with more than 20 million n-grams. The resulting new WFST system achieved a 6 times reduction of the decoding time relative to a previous decoder not based on WFSTs. The construction of the search space in runtime using the "on-
Figure 7.41: Performance variation with the increase of the language model.

"on-the-fly" composition algorithm was crucial in this system since almost all memory of the development and runtime systems (1GB) was used by the language model, and offline compilation of the search network was not feasible, not even using our specialized algorithm [29, 91].

7.9 Summary

In this chapter, various specialized optimizing algorithms for composing the lexicon with the language model were presented. The algorithms provide an exact simultaneous composition and determinization of the lexicon and language model WFSTs. Furthermore, versions which approximate minimization and pushing of weights and output labels were also presented. The algorithms were designed to permit an efficient "on-the-fly" generation of the composition network for use in a dynamic speech recognizer.

The specialized algorithms impose some limitations on the structure of the
lexicon. We feel that the most restrictive one is imposing that the lexicon loops through the initial state, since it limits cross word lexicon modelling, yet we believe that this restriction can be easily overcome if word transitions are somehow marked (for example, by using special “end-of-word” transitions) so as to be identified by the composition algorithm.

The approximation to log probability pushing is very good, yet other language model spreading techniques have been proposed in the literature, for example in [36], which are promising but not easily implementable “on-the-fly”. The implementation of those techniques is an interesting topic for future work.
In this chapter we present WFST modelling approaches to address two research problems related with modelling the European Portuguese language.

The first problem is the use of a WFST-based aligner to perform both word and phone time alignment of large speech corpora. To address this problem, a time tracking mechanism is presented involving the use of special time marker labels on the search space of the decoder.

A WFST-based phonologic rule module for better phone level segmentation is also presented. This phonologic rule module is a very important tool for the study of intra and cross-word pronunciation phenomena in spontaneous speech.

The second problem addressed is the problem of grapheme-to-phone conversion in European Portuguese. This conversion is usually associated with the speech synthesis problem. However, a grapheme-to-phone module is also a useful tool in speech recognition systems, either for initial lexicon development, or for lexicon expansion. We present various approaches to grapheme-to-phone conversion, from knowledge based approaches to data driven and hybrid approaches. The knowledge based approach was also ported to Mirandese, an oral transmission language closely related with Portuguese.

In the next section, we present WFST approaches to time alignment of large
speech corpora, and in Section 8.2 WFST approaches to grapheme-to-phone conversion are presented.

8.1 Word and Phone Level Alignment

8.1.1 Introduction

The work presented in this section was initially motivated by the needs of the national project known as IPSOM [145], whose main goal is to improve the access to digitally stored spoken books, used primarily by the visually impaired community, by providing tools for easily detecting and indexing units (words, sentences, topics). Simultaneously, the project also aims to broaden the usage of multimedia spoken books (for instance in didactic applications, etc.), by providing multimedia interfaces for access and retrieval. Hence, spoken book alignment is a major task.

From the point of view of research, one of the most interesting aspects of the IPSOM project is the fact that indexed spoken books provide an invaluable resource for data-driven prosodic modelling and unit selection in the context of text-to-speech synthesis. This motivated doing the alignment not only on the basis of words, but rather sub-word units.

One other aspect, is that it motivated us to incorporate mechanisms for tracking time frontiers in the WFST-based speech decoder described in Chapter 6.

Phonological rules based in our WFST framework were introduced to automatically generate multiple pronunciations with the goal of improving phone alignment. However, the lack of a hand segmented testing corpus prevented us from properly evaluating the quality of rules in the spoken books.

In [26], and later in [156], we shifted the focus of our pronunciation modelling effort to spontaneous speech, in the context of the Coral dialog corpus [158].

In the next sections we present a description of the aligner, including its phonological rule system. In Section 8.1.3, we describe how pronunciation variation in
European Portuguese was modelled using the rule system. In Section 8.1.4 we describe the word level alignment experiments done in the IPSOM project, and in Section 8.1.5 we describe the phone level experiments using explicit pronunciation modelling in the Coral corpus.

8.1.2 Alignment

An aligner can be build using a decoder which keeps track of the time boundaries between words or phones. Our WFSTs based decoder uses a search space defined by a distribution-to-word transducer that is built outside the decoder. As we saw in previous chapters, that search space is usually build using WFST composition and incorporates at least the composition cascade $H \circ L \circ G$, where $H$ is the HMM or phone topology, $L$ is the lexicon and $G$ is the language model.

For alignment, $G$ should be the sequence of words that constitute the orthographic transcription of the utterance, possibly allowing optional silences between words. The main advantage of the WFST approach regarding recognition, is that since no restrictions are placed on the construction of the search space it can easily integrate other sources of knowledge, furthermore, the network can be optimized and replaced by an optimal equivalent one. This last advantage is a disadvantage from the perspective of alignment because the output labels cannot be used as time markers since there are no warranties that the synchronization between output and input labels is kept after optimizations. As described in Section 6.3.2, our solution to this problem involved the extension of the decoder to deal with special time-marking labels, on the input side, that are internally treated as $\varepsilon$ labels, but are used to mark time transitions or boundaries. The time instants when such time-marking labels are crossed are stored in the recognition hypotheses.

Phone or word level alignment is implemented by placing those labels at the end either of phone WFSTs or of word WFSTs, when the search network is built.
Phonological Rules

Instead of building a lexicon with multiple pronunciations per word, our goal is
to develop phonological rules that can be used with a lexicon of canonical forms,
in order to systematically account for alternative pronunciations. These rules are
specified using a finite-state grammar whose syntax is similar to the Backus-Naur-
form augmented with regular expressions. Each rule is represented by a regular
expression, and to the usual set of operators we added the operator $\rightarrow$, *simple
transduction*, such that $(a \rightarrow b)$ means that the terminal symbol $a$ is transformed
into the terminal symbol $b$. The language allows the definition of non-terminal
symbols (e.g., $\$vowel$). All rules are compiled into *WFSTs*.

Initially [26] all rules were optional. However, in later work [156], *forbidden
rules*, specifying negative constraints, were also incorporated.

Figure 8.1.2 presents an example of the specification of an optional rule; that
specification is first transformed into a transducer $T$, and then compiled into $R_T = \Sigma^*(T \Sigma^*)^*$. That transducer, when composed with the canonical phone transducer
$S$ will produce $S_T = \pi_2(S \circ R_T)$ that allows new pronunciation alternatives.

\[
\$V = (\$Vowel|\$NasalVow|\$Glide|\$NasalGli)
\]

\[
\text{DEF.RULE S.z,} (\$V \ (S \rightarrow z) \$WORD\_BREAK \ $V))
\]

Figure 8.1: Example of an optional rule specification.

Figure 8.2 shows how forbidden rules were later used to improve modelling.
This rule set allows for /S/ in word final position not to be changed into /z/ when
the next word starts by a vowel, only when there is a silence before the next word.
The difference from Figure is that if there is no silence, then the path /S/ end-of-
word vowel is forbidden. A forbidden rule is first transformed into a transducer $T$,
and then compiled into $\Sigma^* \cap \Sigma^* T \Sigma^*$.

We do not apply the optional rules one by one on a cascade of compositions,
$V = (\$Vowel|\$NasalVow|\$Glide|\$NasalGli)$
DEF.RULE S.z, ($V (S \rightarrow z) \$WORD\_BREAK \$V)\)
FORBIDDEN.RULE No.S.z, ($S \ EOW \ V$)

Figure 8.2: Example of forbidden rule specification.

but rather build their union $R = R_{T_1} \cup R_{T_2} \cup \cdots \cup R_{T_n}$. By performing the union of the rules we avoid the exaggerated growth of the resulting transducer, which can be exponential with the length of the composition cascade.

A different type of rules, involving contractions and multi-word reductions can also be implemented through a reduction transducer $Rd$, that encodes rules that map such reductions to their canonical form [57] (e.g. gonna → going to). The phone aligner search space becomes then $H \circ \pi_1(R^{-1} \circ L \circ Rd \circ W)$.

### 8.1.3 Modelling Pronunciation Variation

The way we dealt with different sources of pronunciation variation has some similarities with the one described in [57]. The variations that depend on word-level features of lexical items (such as part-of-speech) and those that are particular to specific lexical entries (such as many acronyms in Portuguese, for instance) are just included in the lexicon that only includes multiple pronunciations for heterophonemic homographs.

The remaining variants that depend on the local phonemic or phonetic context are modelled through pronunciation rules. Rather than specifying rules which would mainly affect function words and forms of the verb to be (estar), one can build a different lexicon which only includes multiple pronunciations for such words.

Some of the rules concern variations that depend on the stress and syllable position of phonemes. The lexicon uses different labels for representing phonemes in particular positions. For instance, label $I$ denotes a frequent alternation between /i/ and /e/ in the beginning of some words starting by “e”. When no rules are
applied, the default pronunciation is /i/.

The main phonological aspects that these rules are intended to cover are: (1) intra-word vowel devoicing; (2) voicing assimilation; and (3) vowel and consonant deletion and coalescence. Both (2) and (3) may occur within and across word boundaries. Some common contractions are also accounted for, with both partial or full syllable truncation and vowel coalescence. Vowel reduction, including quality change, devoicing and deletion, is especially important for European Portuguese, being one of the features that distinguishes it from Brazilian Portuguese and that makes it more difficult to learn for a foreign speaker. As a result of vowel deletion, rather complex consonant clusters can be formed across word boundaries.

8.1.4 Spoken Book Alignment Experimental Results

Pilot Corpus

A small pilot corpus (O Senhor Ventura, by Miguel Torga) was chosen as a test bed for spoken book alignment. The high-quality DAT recordings were manually edited to remove reading errors and extraneous noises, amounting to a total of 2h15m (around 138k words, corresponding to 5k different forms). Although very intelligible, as expected from a professional speaker, the speaking rate was relatively high - 174 words per minute.

Spoken Book Experiments

The experimental results described in this section were obtained with acoustic models trained for a dictation task [112], and are described in Section 5.1.1.

Alignment experiments with spoken books The decoder/aligner allowed us to align the full audio version of the book in a single step. This is especially important if we take into account that the memory limitations of our previous alignment tool imposed a maximum of 3-minute audio segments. We thus avoid
the tedious task of manually breaking-up the audio into smaller segments with their associated text.

The word segmentation of the book, excluding parameter extraction and posterior probability computation, took 197.5 seconds in a 600MHz Pentium III computer (0.024 xRT), and required 200MB of RAM. The phone level alignment of the book ran at 0.027 xRT when using the canonical pronunciations of the lexicon, and 0.03 xRT when using also the pronunciation rules.

**Recognition experiments with spoken books** The edition of recordings to remove reading errors and extraneous noises produced by the speaker is a very labor intensive task. As a first step to automate this procedure, we tried to match text recognized using a dedicated recognizer with the original text in order to detect incorrect audio portions. The dedicated recognizer uses a lexicon and an \( n \)-gram language model estimated from all the book’s text and achieved a word error rate of 17.2%.

Significant improvements were obtained by using the automatic alignment labels for speaker adaptation of the acoustic model, using only 80% of the available audio. The adapted model created using the alignment made with the canonical pronunciation lexicon achieved 7.8% WER, and the one using the phonological rules obtained 7.1%.

### 8.1.5 Spontaneous Dialogs Alignment Experimental Results

**Coral Corpus**

Coral is a map task dialog corpus, involving spontaneous conversations between pairs of speakers about map directions. In the 16 different maps, the names of the landmarks were chosen to allow the study of some connected speech phenomena: sequences with /l/ favoring or not its velarization (e.g. *sala malva*, *sal amargo*);
sequences with /s/ in word final position followed by another coronal fricative (e.g. poos secos); sequences of plosives formed across word boundaries (e.g. clube de tiro); and sequences of obstruents formed within and across word boundaries (e.g. bairros degradados).

The recordings involved 32 speakers (students from the Lisbon area), and took place in a small sound proof room at INESC. The two speakers were separated by a distance of about one meter with a small screen wall in between them, whose goal was to avoid direct visual contact between the participants, but did not provide acoustic isolation. The speakers wore close-talking microphones and the recordings were made in stereo directly to DAT and later down-sampled to 16 kHz per channel.

Given the recording conditions, a reasonable amount of cross-talk from the other channel was clearly audible. Our first alignment experiments, in fact, were made with the original signals and produced too bad results - the end of the turns was not properly detected, which caused words from one of the speakers to be frequently aligned during the other speaker's turn. The problem was much aggravated when overlap occurred, which was fairly frequent. In order to reduce this cross-talk, we adopted an adaptive noise cancelling scheme [170, 26].

All dialogs were orthographically transcribed following the same transliteration conventions using SGML format of other map task corpora\footnote{http://www.herc.ed.ac.uk/dialogue/maptask.html}. Only a small pilot dialog was annotated at all levels (including, fairly recently, the phone level, using the SAMPA phonetic alphabet, but only for the left channel).

Alignment Evaluation

The experimental results described in this section were obtained with acoustic models trained for a broadcast news recognition task [92]. Those models are described in Section 5.1.2.

This section describes our alignment results with the small subset of the Coral
corpus that has been manually annotated at the phone level. In order to evaluate the distinct versions of our aligner, we used as a metric the phone level error rate and two additional measures: the percentage of matching phone labels for which the absolute error is less than 10 ms and the average absolute error in 90% of the cases. The results are shown in table 8.1, where Lex0 is a lexicon which only contains multiple pronunciations for heterophonic homographs, and Lex1 also includes multiple pronunciation for function words and forms of the verb to be estar; as described in Section 8.1.3.

A dynamic programming algorithm was developed to match the manually annotated labels with the automatically derived ones, minimizing their string-edit distance. The algorithm penalizes substitutions, insertions and deletions (costs 10, 7 and 7 respectively), but favors very common ones (cost 3). The algorithm was based on the WFST approach to string-edit distance computation referred in Section 3.4.

Our first experiment was made with Lex0 and no alternative pronunciation rules. An analysis of the largest errors shows they are due to the fact that we did not try to align laughs, annoying grunts, and filled pauses, which causes severe misalignments in the neighboring words (up to 5 ms). In fact, our acoustic models could not yet cope with such phenomena. The performance in overlapping turns is on the same level as the one in non overlapping turns. The second experiment was made with Lex1 and still no rules. The values obtained with this new lexicon were good enough to make all further tests using this lexicon.

We then followed an exhaustive process of testing the efficiency of several types of alternative pronunciation rules. The results obtained with 32 rules are shown in the last line of the table. We were expecting greater improvements, but we cannot dismiss the generalization capabilities of our acoustic models and also the fact that they cannot adequately model laughs and other voice quality changes that seriously affect some portions of the dialogues.

151
A last experiment was made to test the efficiency of reduction rules developed to handle a couple of examples (e.g. \textit{para a} $\rightarrow$ \textit{pad}) whose reduced forms are rarely lexicalized in Portuguese. These examples, however, were not so frequent in our test corpus, which justifies that the results were practically the same with and without the reduction transducer.

<table>
<thead>
<tr>
<th>Lex</th>
<th>Rules</th>
<th>%ACC</th>
<th>\leq 10ms</th>
<th>Percentil 90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lex0</td>
<td>no</td>
<td>70.04</td>
<td>46.54</td>
<td>0.0122</td>
</tr>
<tr>
<td>Lex1</td>
<td>no</td>
<td>71.50</td>
<td>47.29</td>
<td>0.0118</td>
</tr>
<tr>
<td>Lex1</td>
<td>yes</td>
<td>78.19</td>
<td>48.57</td>
<td>0.0115</td>
</tr>
</tbody>
</table>

Table 8.1: Alignment results.

8.1.6 Summary

Various applications of WFST techniques to the problem of automatic alignment of speech corpora were described.

The initial spoken book alignment task motivated the development of an aligner, which was based on the decoder described in chapter 6. This aligner managed to fully automate the spoken book alignment process in a very fast single-step procedure, even for a 2-hour long recording.

A phonological rule system based on WFST's was built and used to study the use of explicit pronunciation modelling in a spontaneous speech phone alignment task. A significant reduction in the percentage of miss aligned phones was observed.

8.2 Grapheme-to-Phone

Grapheme-to-phone conversion is a task usually associated with speech synthesis, rather than speech recognition which is the topic of this thesis. Nevertheless, a grapheme-to-phone module can be an important component of a speech system,
specially if the vocabulary of the system is to be adapted or extended by the end user. In such a system, of which a dictation system is an example, a grapheme-to-phone module could provide initial pronunciations for unknown words. The inversion of a grapheme-to-phone transducer, even considering the overanalysis problem, could also be helpful to deal with out-of-vocabulary words.

This section describes different approaches to grapheme-to-phone conversion implemented as Weighted Finite State Transducers (WFSTs). The objective of a grapheme-to-phone module implemented as WFSTs is justified by their flexibility in the efficient and elegant integration of multiple sources of information, such as the information provided by other “text-analysis” modules. The flexibility of WFSTs also allows the easy integration of knowledge-based with data-driven methods.

Our first approach to grapheme-to-phone (GtoP) conversion for European Portuguese was a rule-based system (DIXI), with about 200 rules [119]. All the code was programed in C, directly in the case of the stress assignment rules, and using the SCYLA (“Speech Compiler for Your Language”) [88] rule compiler, developed by CSELT, for the remaining rules. The multi-level structure of this compiler allowed each procedure to simultaneously access the data resulting from all the previous procedures, so the rules could simultaneously refer to several levels (such as the grapheme level, phone level, sandhi level, etc.)

Later, this rule-based approach was compared with a neural net approach [159] that, in spite of the fairly good results, was never integrated in our synthesizer. Instead, we integrated an approach based on CARTs (Classification and Regression Trees) [120] combined with a large coverage lexicon, as part of the porting of our TTS system (now designated as DIXI+) to the Festival framework [15].

Some of the most common approaches to grapheme-to-phone conversion can be compiled to WFSTs, among which are CARTs [150], and most rule systems, such as two-level [75] and rewriting rules [67].

In this section, we first show how we compiled the rules of the DIXI system
to WFSTs (Section 8.2.1), we then present data-driven approaches to the problem (Section 8.2.3), and finally we combine the knowledge-based with the data-driven approaches (Section 8.2.4).

In order to assess the performance of the different methods, we used a pronunciation lexicon built on the PF (“Português Fundamental”) corpus. The lexicon contains around 26000 forms. 25% of the corpus was randomly selected for evaluation. The remaining portion of the corpus was used for training or debugging.

The size of the training material for the data-driven approaches was increased with a subset of the BD-Público [113] text corpus. This corpus includes a collection of texts from the on-line edition of the Público newspaper. We used all the words occurring in the first 1,000,000 paragraphs of this corpus, and obtained their transcription by rule using DIXI. The 205k words not in PF were added to the training set.

8.2.1 Knowledge-Based System

Our first goal was to convert DIXI’s rules to a set of WFSTs. SCYLA rules are of the usual form $\phi \rightarrow \psi/\lambda\ldots\rho$ where $\phi$, $\psi$, $\lambda$ and $\rho$ can be regular expressions that refer to one or multiple levels. The meaning of the rules is that when $\phi$ is found in the context with $\lambda$ on the left and $\rho$ on the right, $\psi$ will be applied, replacing it or filling a different level of $\psi$.

The application of standard generative rewriting rules [32] to a sequence of graphemes poses some well known problems, as it leads to unnecessary rule dependencies due to the replacement of graphemes by phones: the first rule has only graphemes on its context, while the last ones have mainly phones. That happens to a small-extend in [18], for example.

In DIXI’s case, some of these problems may be avoided, as most of the grapheme-to-phone rules were written such that $\phi$, $\lambda$ and $\rho$ only refer to the grapheme level (with stress marks already placed on it) and $\psi$ only to the phone
level, represented in a different tier of the multi-level data-structure. There are no intermediate stages of representation and no rule creates or destroys the necessary context for the application of another rule. In order to prevent some common errors, a small set of 6 rules was nevertheless added which refers grapheme-phone correspondences on either context $\lambda$ or $\rho$. Note, however, that although some similarities may be found between DIXI's and a Two-Level Phonology approach ([85], [6]), DIXI's rules are not two-level rules: contexts are not fully specified as strings of two-level correspondences and within the set of rules for each grapheme, a specific order of application is required. Default rules need to be the last and in some cases in which the contexts of different rules overlap partially, the most specific rule needs to be applied first.

In order to preserve the semantic of DIXI's rules, we opted to use rewriting rules, but in the following way:

First, the grapheme sequence $g_1, g_2, \ldots, g_n$, is transduced into $g_1, \ldots, g_2, \ldots, \ldots, g_n$, where $-$ is an empty symbol, used as a placeholder for phones. Each rule will replace the phone corresponding to the previous grapheme, keeping it. The context of the rules can now freely refer to the graphemes. The few DIXI rules whose context referred to phones can also be straightforwardly implemented. The very last rule removes all graphemes, leaving a sequence of phones. The input and output language of the rule transducers is thus a subset of (grapheme phone)*. The set of graphemes and the set of phones do not overlap.

**Rule specification language**

The rules are specified using the same rule specification language introduced originally to deal with alternative pronunciation rules (see section 8.1.2). This work motivated us to extend the language with two commands.

The first one is:

```
OB_RULE n, \phi \rightarrow \psi/\lambda\ldots\rho
```
where \( n \) is the rule name and \( \phi, \psi, \lambda, \rho \) are regular expressions. \texttt{ObRule} specifies a context dependent \textit{obligatory rule}, and is compiled using Mohri and Sproat’s algorithm [107].

The second command is:

\texttt{CD.Trans} \( n, \tau \Rightarrow \lambda \ldots \rho \)

where \( \tau \) is a transducer (an expression that might include the \( \rightarrow \) operator). \texttt{CD.Trans} (Context-Dependent Transduction) is a generalization where the replacing expression depends on what was matched. It is compiled using a variation of Mohri and Sproat’s algorithm, that uses \( \pi_1(\tau) \) instead of \( \phi \), and \( \tau \) instead of the cross product \( \phi \times \psi \). Its main advantage is that it can succinctly represent a set of rules that apply to the same context. We use it mainly in the stress-marking phase of the grapheme-to-phone conversion.

Grapheme-to-Phone Phases

The rules of the grapheme-to-phone system are organized in various phases, each represented by transducers that can be composed to build the full system for EP. Figure 8.3 shows how the various phases are composed.

Each phase has the following function:

- the \texttt{stress} phase consists of 27 rules that mark the stressed vowel of the word.

- \texttt{introduce-phones} is the simple rule that inserts the \texttt{empty phone} placeholder after each grapheme. (\texttt{\$Letter (NULL \rightarrow \texttt{EMPTY}) \Rightarrow \ldots}).

- \texttt{prefix-lexicon} consists of pronunciation rules for compound words, namely with roots of Greek or Latin origin such as “tele” or “aero”. It includes 92 rules.

156
gr2ph is the bulk of the system, and consists of 340 rules, that convert the 45 graphemes (including graphically stressed versions of vowels) to phones.

sandhi implements word co-articulation rules across word boundaries. (This rule set was not tested here, given the fact that the test set consists of isolated words.)

remove-graphemes removes the graphemes in order to produce a sequence of phones. ($Letter \rightarrow \text{NULL} / \_\_\_\_\_\_\_\_\_$).

stress o
introduce-phones o
prefix-lexicon o
gr2ph o
sandhi o
remove-graphemes

Figure 8.3: Phases of the knowledge based system.

The following example illustrates the specification of 2 gr2ph rules for deriving the pronunciation of grapheme g: either as /Z/ (e.g. agenda, gisela) when followed either by e or i, or as /g/ otherwise (SAMPA symbols used).

OB_RULE 0200, g EMPTY -> g _Z \ 
/ NULL ___ ($A11E | $A11I)

OB_RULE 0201, g EMPTY -> g _g \ 
/ NULL ___ NULL

8.2.2 From Rules to Transducers

The compilation of the rules results in a very large number of WFSTs (almost 500) that need to be composed in order to build a single grapheme-to-phone transducer.
We did not build a single WFST but selectively composed the WFSTs and obtained a small set of 10 WFSTs that are composed with the grapheme WFST in runtime to obtain the phone WFST.

The most problematic phase was gr2ph. We started by composing each of the other phases into a single WFST. gr2ph was first converted to a WFST for each grapheme. Some graphemes, such as e, lead to large transducers, while others lead to very small ones. Due to the way we specified the rules, the order of composition of these WFSTs was irrelevant. Thus we had much flexibility in grouping them and managed to obtain 8 transducers with an average size of 410k. Finally, introduce-phones and remove-graphemes were composed with other WFSTs and we obtained the final set of 10 WFSTs.

In runtime, we can either compose the grapheme WFST in sequence with each WFST, removing dead-end paths at each step, or we can perform a lazy simultaneous composition of all WFSTs. This last method is slightly faster than the DIXI system.

Evaluation

We evaluated the WFST-based rule approach, and compared its performance with the one of our previous rule-based DIXI system. As can be seen in table 8.2, the WFST achieved almost the error rate of the DIXI system it is emulating, both at a word level and at a grapheme level. The two rightmost columns show the error rates obtained without taking stress mark errors into account. The difference between the performance of the current and previous approaches is due to the exception lexicon included in DIXI that we did not yet implement. In [157] we added this lexicon as an additional state in the composition cascade and obtained the same results as DIXI.
<table>
<thead>
<tr>
<th>System</th>
<th>% Error word</th>
<th>% Error graph.</th>
<th>% Error w/o stress word</th>
<th>% Error w/o stress graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFST</td>
<td>3.56</td>
<td>0.54</td>
<td>3.13</td>
<td>0.47</td>
</tr>
<tr>
<td>DIXI</td>
<td>3.25</td>
<td>0.50</td>
<td>2.99</td>
<td>0.45</td>
</tr>
</tbody>
</table>

Table 8.2: Comparison of the current and previous rule-based approaches.

### 8.2.3 Data-Driven Approaches

**Grapheme-Phone Alignment**

The first step in preparing the corpus for the data driven techniques consisted of aligning each grapheme with the corresponding phone.

We performed the alignment by minimizing the string-edit distance between corresponding grapheme and phone strings, using the technique described in section 3.4. We encoded the distance between any grapheme and any phone, as well as the insertion costs of phones and the deletion costs of graphemes as a transducer s. The alignment between a grapheme sequence g and a phone sequence p was obtained as bestpath(g ⪋ s ⪋ p).

When creating s we opted to capitalize on the knowledge obtained from the rule system, although automatic techniques exist that can learn such a transducer automatically [40].

Besides the usual matching of 1 grapheme to 1 phone, we also allowed the direct matching of some sequences. The cost of matching a grapheme sequence with a phone sequence was set to zero if there is a rule that assigns the phone sequence to the grapheme sequence (completely ignoring the context of the rule). In most cases, the matching was of 1 grapheme to 1 phone, but we modelled some cases of 2 graphemes to 1 phone (such as nh → /J/, lh → /L/, rr → /R/) and some cases of 1 grapheme to 2 or 4 phones (such as ê → /6̃j̃6̃j̃/ in têm). In order to score all possible alignments, we allowed the alignment of a grapheme with any phone.
the deletion of graphemes and the insertion of phones, also at a cost. The costs were set to the costs commonly used to determine the word error rate in speech recognition (3 for insertion and deletion, 4 for substitution and 0 for matching).

N-gram approach

The alignment obtained in section 8.2.3 is a sequence of pairs (grapheme, phone), where the grapheme or the phone can also be \( \epsilon \). Our first data-driven approach consisted of modelling that sequence using an n-gram model, as proposed by [149].

This model is based on the probability of a grapheme matching a particular phone given the history up to the previous \( n-1 \) pairs \( P((g_2,p_2)(g_{i-1},p_{i-1})\ldots(g_{i-1},p_{i-1})) \).

The language model is first converted to a finite-state acceptor (WFSA) over pairs of symbols, and then to a finite-state transducer \( t \), by transforming each pair of symbols into an input and an output label. \( t \) is ambiguous because \( \epsilon \) labels are used to model backoff transitions during the conversion from n-gram to WFSA, and hence, even if there is an explicit n-gram in the model, the WFSA will still allow alternative paths that use the backoff.

Due to this ambiguity, in order to use the WFST to convert a grapheme sequence WFST \( g \) to phones, we need to compute \( \text{bestpath}(\pi_2(g \circ t)) \).

Evaluation

We trained various n-gram backoff language models using history lengths \( n-1 \) ranging from 2 to 7. Table 8.3 shows the size of the various models, and table 8.4 shows the error rate on the test set (second and third columns).
<table>
<thead>
<tr>
<th>n</th>
<th>n-grams</th>
<th>states</th>
<th>edges</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1,392,426</td>
<td>820,778</td>
<td>1,983,113</td>
<td>42M</td>
</tr>
<tr>
<td>7</td>
<td>981,565</td>
<td>592,184</td>
<td>1,459,738</td>
<td>30M</td>
</tr>
<tr>
<td>6</td>
<td>657,107</td>
<td>361,944</td>
<td>980,123</td>
<td>20M</td>
</tr>
<tr>
<td>5</td>
<td>401,855</td>
<td>159,425</td>
<td>549,398</td>
<td>11M</td>
</tr>
<tr>
<td>4</td>
<td>173,307</td>
<td>37,869</td>
<td>208,668</td>
<td>4M</td>
</tr>
<tr>
<td>3</td>
<td>42,451</td>
<td>3,618</td>
<td>46,018</td>
<td>0.8M</td>
</tr>
</tbody>
</table>

Table 8.3: Pair n-gram WFSTs.

<table>
<thead>
<tr>
<th>n</th>
<th>% Error</th>
<th>% Error w/o stress</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>word</td>
<td>graph.</td>
</tr>
<tr>
<td>8</td>
<td>9.04</td>
<td>1.37</td>
</tr>
<tr>
<td>7</td>
<td>9.02</td>
<td>1.37</td>
</tr>
<tr>
<td>6</td>
<td>9.16</td>
<td>1.37</td>
</tr>
<tr>
<td>5</td>
<td>9.86</td>
<td>1.46</td>
</tr>
<tr>
<td>4</td>
<td>15.34</td>
<td>2.25</td>
</tr>
<tr>
<td>3</td>
<td>31.62</td>
<td>4.62</td>
</tr>
</tbody>
</table>

Table 8.4: Performance of the n-gram approach.

8.2.4 Combining Multiple Approaches

Combining Data-Driven and Knowledge-Based Approaches

One of the greatest advantages of the WFST representation is the flexible way in which different methods may be combined. In this section we show some examples of the combination of data-driven with knowledge-based methods.

In [149], as an example of the integration of knowledge-based with data-driven methods, some improvements were obtained by composing the n-gram WFST with a WFST that restricts the primary stress to exactly one per word. This type of restriction had also been implemented in our neural network method as a post-processing filter.

We opted for a different approach: as we have the stress marking WFST stress,
we decided to perform the grapheme-phone alignment of the training data not with the original words, but with the output of the stress WFST. The alignments thus obtained were used to build n-gram WFSTs, as described in section 8.2.3. To convert a sequence of graphemes \( g \) to phones, we now use \( \text{bestpath}(\pi_2(g \circ \text{stress} \circ t)) \).

**Evaluation**

Table 8.5 shows the results obtained with this variation with several n-gram models. We observe a reduction of the word error rate to less than half. The result is even more impressive when we remember that around 90% of the training set was converted by rule with a system that has around 3% errors. The size of the n-gram WFSTs was similar.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( % \text{ Error} )</th>
<th>( % \text{ Error w/o stress} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>word</td>
<td>graph.</td>
</tr>
<tr>
<td>8</td>
<td>4.01</td>
<td>0.61</td>
</tr>
<tr>
<td>7</td>
<td>3.94</td>
<td>0.59</td>
</tr>
<tr>
<td>6</td>
<td>4.02</td>
<td>0.61</td>
</tr>
<tr>
<td>5</td>
<td>4.04</td>
<td>0.61</td>
</tr>
<tr>
<td>4</td>
<td>4.48</td>
<td>0.67</td>
</tr>
<tr>
<td>3</td>
<td>6.40</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table 8.5: Performance of the n-gram approach, when trained and used after the stress marking WFST.

**Combining Multiple Data-Driven Approaches**

Some experiments combining multiple data-driven approaches were also performed. Since the stressed vowel in Portuguese is set in most cases relative to the end of the word (the default rule is that the penultimate syllable is stressed), one can expect a better behavior from backward n-grams (trained from reversed strings) than
from standard forward ones. Backward n-grams are obtained by first reversing all strings in the training corpus and then training n-gram models from the resulting corpus. We observe improvements relative to forward n-grams, as shown in Table 8.6. Furthermore, a slight additional performance improvement can be obtained by combining both forward and backward n-grams as seen in Table 8.7. In this case, the solution is obtained as $\text{bestpath}(\pi_2(g \circ t_f) \cap \text{rev}(\pi_2(\text{rev}(g) \circ t_b)))$, where $t_f$ and $t_b$ are respectively the forward and backward n-gram WFSTs, and $\text{rev}$ is the reverse operation.

<table>
<thead>
<tr>
<th>$n$</th>
<th>% Error word</th>
<th>% Error graph.</th>
<th>% Error w/o stress word</th>
<th>% Error w/o stress graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8.92</td>
<td>1.34</td>
<td>6.09</td>
<td>0.91</td>
</tr>
<tr>
<td>7</td>
<td>8.59</td>
<td>1.29</td>
<td>5.97</td>
<td>0.89</td>
</tr>
<tr>
<td>6</td>
<td>8.61</td>
<td>1.29</td>
<td>5.95</td>
<td>0.89</td>
</tr>
<tr>
<td>5</td>
<td>8.87</td>
<td>1.32</td>
<td>6.06</td>
<td>0.89</td>
</tr>
<tr>
<td>4</td>
<td>14.22</td>
<td>2.02</td>
<td>8.85</td>
<td>1.25</td>
</tr>
<tr>
<td>3</td>
<td>31.29</td>
<td>4.49</td>
<td>18.25</td>
<td>2.67</td>
</tr>
</tbody>
</table>

Table 8.6: Performance of the backward n-grams.

<table>
<thead>
<tr>
<th>$n$</th>
<th>% Error word</th>
<th>% Error graph.</th>
<th>% Error w/o stress word</th>
<th>% Error w/o stress graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8.10</td>
<td>1.21</td>
<td>5.68</td>
<td>0.84</td>
</tr>
<tr>
<td>7</td>
<td>7.93</td>
<td>1.18</td>
<td>5.56</td>
<td>0.82</td>
</tr>
<tr>
<td>6</td>
<td>8.13</td>
<td>1.20</td>
<td>5.63</td>
<td>0.83</td>
</tr>
<tr>
<td>5</td>
<td>8.83</td>
<td>1.29</td>
<td>5.96</td>
<td>0.86</td>
</tr>
<tr>
<td>4</td>
<td>14.16</td>
<td>2.03</td>
<td>8.71</td>
<td>1.23</td>
</tr>
<tr>
<td>3</td>
<td>31.39</td>
<td>4.59</td>
<td>31.39</td>
<td>2.68</td>
</tr>
</tbody>
</table>

Table 8.7: Performance of combining the forward and reverse n-grams.
8.2.5 Porting to Other Languages

The flexibility obtained with the WFST approaches to grapheme-to-phone led us to consider the adaptation of the knowledge-based technique to other varieties of Portuguese or oral transmission languages spoken in the CPLP (Community of the Portuguese Speaking Countries).

The first effort was to port the system to the Mirandese (Mirandés) language.

On the northeastern border of Portugal, there is a small set of villages where people use two languages in their daily life: Portuguese and Mirandese. The use of the latter is typically confined to conversations among relatives and neighbors. Altogether, the population that speaks Mirandese does not exceed 12,000. Its recognition as official language is fairly recent (1999). Nowadays, it is also taught as an optional course in high school.

Mirandese is a romance language, related to Asturian-Leonese, and for several centuries it was preserved only as an oral transmission language. Despite belonging to this geolinguistic domain, Mirandese is different from all the languages spoken in the contiguous territories. The first attempt at creating a spelling norm for Mirandese was done by a nineteenth century linguist José Leite de Vasconcelos [162], who based the norm on the pronunciation of the village Duas Igrejas and used diacritics in order to faithfully encode this pronunciation. These diacritics created so many doubts in whoever tried to follow him, that the end result was the appearance of many spelling modes. This provided a strong motivation for the recent efforts to create an orthographic convention [12] in order to establish unifying criteria for writing in this language. In the design of the convention, it was not attempted to annotate all local pronunciation variants, but only those that either show up regularly or represent a phenomenon that was once regular in the language and nowadays is realized differently in the set of villages.

The motivation for deriving grapheme-to-phone rules for Mirandese was to build

http://mirandes.no.sapo.pt

164
a tool that may help native speakers to learn how to read and write, as well as students interested in that language.

As a starting point, we used the knowledge-based approach for the European Portuguese. In fact, none of the data-driven tools that we had developed since were suited for Mirandese, given the small amount of training material.

Continuities and Differences Between the Two Languages

A very detailed description of the main continuities and differences between Portuguese and Mirandese can be found in [11]. Here, for the sake of space, we only try to emphasize the most relevant ones.

Mirandese exhibits several continuity features relative to either what is commonly designated as standard European Portuguese or the one spoken in the Northern part of the country: the existence of initial $f$; unvoiced affricate $ch$ derived from the Latin $cl$, $pl$ and $fl$; voiced palatal consonant derived from the Latin $ly$ and $cl$; vocalization of the $-ct-$ group as $-it-$; existence of oral diphthongs $ei$ and $ou$; voiced and unvoiced post-alveolars; four voiced and unvoiced sibilants; absence of $v$ ($b$ used instead); existence of nasal vowels and diphthongs; etc.

Some of the main differences are: the preservation of the Latin intervocalic $-n-$ and $-l-$ (originating differences in syllabic structure); palatalization of the Latin $-mn-$, $-mn$ and $-ll$; existence of initial $lh$- originating from the Latin $l$; the tendency for the non-existence of high unstressed vowels in initial position; the reduction of the initial $des$- to a sibilant consonant; male form of the definite article reduce to $l$, with two phonetic values depending on the beginning of the next word and the ending of the previous one; existence of raising diphthongs $ie$ and $uo$ (although with several differences in realization from village to village); etc.

We tried to limit to the minimum the creation of new SAMPA symbols to cover Mirandese, which means that the same symbol may be used with slightly different realizations in EP and Mirandese (e.g., while the symbols $[E]$ and $[O]$ account for
low vowels in EP, they correspond to mid-low ones in Mirandese). New symbols were only introduced to account for either phonological contrasts involving 3 pairs of coronal fricatives, or important differences in contextual variation (e.g. while in standard EP e is always realized as [eʷ] when followed by a tautosyllabic nasal consonant, in Mirandese it corresponds to [Eʷ] and [@ʷ] in stressed and unstressed positions, respectively). Table 8.8 lists the additional symbols, together with some examples.

<table>
<thead>
<tr>
<th>SAMPA</th>
<th>Orthography</th>
<th>Transcription</th>
</tr>
</thead>
<tbody>
<tr>
<td>@~</td>
<td>centelha</td>
<td>s@~tʲejL6</td>
</tr>
<tr>
<td>E~</td>
<td>benga</td>
<td>bʲE~g6</td>
</tr>
<tr>
<td>B</td>
<td>chuba</td>
<td>tSⁿuB6</td>
</tr>
<tr>
<td>D</td>
<td>roda</td>
<td>RⁿOD6</td>
</tr>
<tr>
<td>G</td>
<td>pega</td>
<td>pʲEG6</td>
</tr>
<tr>
<td>s_</td>
<td>sol</td>
<td>sₜ”Ol~</td>
</tr>
<tr>
<td>z_</td>
<td>rosa</td>
<td>RⁿOzₜ6</td>
</tr>
</tbody>
</table>

Table 8.8: Additional SAMPA symbols for Mirandese.

Porting Results

The porting of the FST-based approach from EP to Mirandese involved changing the stress and gr2ph transducers. The stress rules showed only small differences compared to the ones for EP (e.g. stress of the words ending in ç, n, and ie). The gr2ph transducer was significantly smaller than the one developed for EP (around 120 rules), reflecting the much closer grapheme-phone relationship.

The porting effort was done in a bottom up way i.e., we took only the most generic rules for EP and made the necessary modifications to adapt these rules to Mirandese, namely to the ones involving the new phonetic symbols. We then consecutively added more rules to this set of default ones. The rules for consonant conversion are very simple. The ones for the 5 vowels take up around 60% of the rules. As an example of differences between the two rule sets, let us take the ones that involve the conversion of graphemes p and x. Both graphemes have a single
rule in Mirandese whereas in EP we needed 8 rules to account for the cases where
\( p \) was not pronounced, and 16 to account for the 4 possible pronunciations of \( z \).

The hardest step in this porting effort involved the definition of development
and test corpora for Mirandese. Whereas for EP the choice of the reference pronun-
ciation (the one spoken in the Lisbon area and most often observed in the media),
was fairly easy, for Mirandese it was a very hard task, given the differences between
the pronunciations observed in the different villages of the region. This called for
a thorough review of the lexicon, and checking with native speakers. For develop-
ment, we used a small lexicon of about 300 words extracted from the examples in
[12]. For testing, we used a manually transcribed lexicon of around 1,100 words,
built from a corpus of oral interviews conducted by CLUL in the framework of
the ALEPG project (Atlas Linguístico-Etnográfico de Portugal e da Galiza). As a
starting point, we selected the interviews collected in the village of Duas Igrejas,
which was also the object of the pioneering studies of Mirandese by Vasconcelos.

Our first tests were done without an exceptions lexicon. In our small develop-
ment set, we obtained only 11 errors (3.68\% error rate at a word level), all of which
are exceptions (foreign words, function words, etc.). For the test set, a similar error
rate was obtained (3.09\%). Roughly half of the errors will have to be treated as
exceptions, and half correspond to stress errors.

This reduced error rate was expected, given the coherence of the recent ortho-
graphic convention. In fact, the number of errors obtained when we first run the
test set through the grapheme-to-phone module was very high. However, after
checking with native speakers, it became clear that the majority of the mispronun-
ciations were due to incorrectly spelled entries in this corpus. Most of them were
observed for words that according to the convention should be written with \( ie \), but
were written instead with \( e \) or \( i \), because, alongside with the pronunciation [je], we
can also find [e] or [i], depending on the village.
8.2.6 Summary

In this section we compared several WFST-based approaches to GtoP conversion for European Portuguese: knowledge-based, data-driven and hybrid. Best results were obtained with the knowledge-based approach, but one should take into account the fact that the data-driven one was trained with automatically transcribed material. The comparison between the different approaches should consider as well the size of the resulting transducers and other properties which may also be quite relevant, such as the fact that the rule-based approach generates dead ends, whereas the n-gram approach does not, but requires a best-path search.

We also took advantage of the flexibility of WFSTs to port the knowledge-based approach to a close language, Mirandese. The hardest part of the porting task turned out to be the establishment of a reference pronunciation lexicon that could be used as the development corpus, given the observed differences in pronunciation between the inhabitants of the small villages in that region.

The use of finite state transducers allows a very flexible and modular framework for deriving new rule sets, testing the consistency of the orthographic conventions and helping in the establishment of coherent orthographic forms for dictionary entries. Based on this experience, we think that grapheme-to-phone systems could be useful tools for researchers involved in the establishment of orthographic conventions. Moreover, such tools could be helpful in the design of such conventions for other oral transmission languages in the CPLP.

In the future, we plan to convert our CART-based approach to the WFST framework. This will give us much flexibility in combining the various methods, for example, a WFST resulting from the conversion of the tree of a particular grapheme could replace the respective grapheme rules in the WFST rule-based system. Another type of approach we plan to explore is transformation-based learning. In [18] it was successfully used to improve a Dutch rule-based grapheme-to-phoneme system. It is an attractive technique due to the readability of the
inferred rules and of the possibility of conversion to FSTs [137], as we saw in section 3.5.2.

The inversion property of transducers opens the possibility of using GtoP techniques in tasks such as reconstructing out of vocabulary words [42] in large vocabulary speech recognition systems.
CHAPTER 9

Summary and Future Directions

9.1 Summary

The main topics of this thesis were the problems of scalability and adaptability of WFST approaches when applied to large vocabulary speech recognition.

Our first proposal to reduce the memory required to build WFST systems consisted of decoupling the language model from the other knowledge sources of the system. The decoupling was done by using not one, but two WFSTs, to represent the search space. One WFST representing the language model, and the other being an optimized decoupled search network representing the remaining knowledge sources. A decoder was built that allowed multiple hypotheses to share the same state in the decoupled search network, as long as each hypothesis referred to a different language model state. This approach was first tested in a large vocabulary continuous speech task with 5k words and achieved good results. When we extended the approach to a more demanding task, with a vocabulary of 27k words, it became apparent that the technique had problems achieving zero search errors with good performance. These problems were related to the fact that the decoder could not exploit full language model information as early as possible by tightly coupled approaches.

171
A different approach was then followed where we replaced the general \textit{WFST} optimization algorithms (determinization, pushing and minimization) with a specialized composition algorithm. This algorithm was designed to perform the composition of the lexicon with the language model in a tightly integrated network.

The algorithm relies on various assumptions about the structure of the lexicon \textit{WFST}; these are very reasonable assumptions which apply to most lexicon \textit{WFST} representations, however they do not apply to general \textit{WFST}s.

The algorithm consists of tagging each \(\epsilon\)-output edge in the lexicon with the set of reachable non-\(\epsilon\) output labels. These sets are then used to avoid generating non-\(\epsilon\)-accessible states, when composing a deterministic lexicon with a language model. If the language model is itself deterministic, then the specialized algorithm generates a deterministic composition \textit{WFST}. The algorithm was later extended to approximate other optimizing operations such as pushing and minimization. Pushing is approximated with the use of language model lookahead. Minimization is approximated with both suffix and tail sharing versions of the composition algorithm.

A very important property of the algorithm is that it allows the “on-the-fly” generation of the composition \textit{WFST}, state by state. This property allows the algorithm to be embedded in a dynamic speech recognition system.

Various speech recognition systems were built based on the use of this algorithm to optimize a tightly integrated search space. Both static and dynamic systems were tested. The optimized search networks built with our algorithms performed on par with networks built using the general \textit{WFST} optimization algorithms. However, our optimization algorithm requires much less resources to compile the search space. The use of our algorithm in runtime in dynamic systems revealed a small overhead (of 5 to 30\%) relative to the use of a precompiled static network. To test the scalability of the approach, a dynamic system was build for a large broadcast news transcription task. This \textit{WFST} system achieved a 6 times reduction of the decoding
time relative to a previous decoder not based on WFSTs.

The thesis also addressed some modelling problems related with the European Portuguese language:

The first problem was the use of a WFST-based aligner to perform both word and phone time alignment of large speech corpora. To address this problem we developed a time tracking mechanism involving the use of special time marker labels on the search space of the decoder. The technique allowed the efficient segmentation of large speech signals; spoken books with over 2 hours of speech were word-segmented at 0.024 xRT.

This initial alignment work motivated us to develop a WFST-based phonologic rule module for better phone level segmentation. This phonologic rule module revealed itself a very important tool for the study of intra and cross-word pronunciation phenomena in spontaneous speech.

The experience gained during the development of these rules motivated us to address the problem grapheme-to-phone conversion in European Portuguese. This conversion is usually associated with the speech synthesis problem. However, a grapheme-to-phone module is also a useful tool in speech recognition systems, either for initial lexicon development, or for lexicon expansion. We implemented various approaches to grapheme-to-phone conversion, from knowledge based approaches to data driven and hybrid approaches. The knowledge based approach was later ported to Mirandese, an oral transmission language closely related with Portuguese.

9.2 Future Directions

Regarding the search related problems, we believe that future developments should focus on reducing the constraints imposed on the lexicon by our specialized composition algorithm.

However, the more interesting prospects for future work are related with the
use of the flexibility obtained with our specialized composition algorithm, to pursue the integration of higher level knowledge sources. Many of the written language techniques described in Chapter 3 can be seamlessly combined with WFST-based speech recognition, allowing the construction of powerful speech understanding systems.

A very interesting application area is the area of speech-to-speech translation. Currently, an important research problem in the field is how to combine statistical approaches with explicit linguistic knowledge. We believe that WFST techniques present a very promising framework in which multiple approaches can be integrated.

We plan to return to what was the original theme of the thesis, and start integrating machine translation knowledge sources in our speech recognition system, not only with the original goal of improving recognition accuracy, but with the more ambitious goal of performing full speech-to-speech translation.
APPENDIX A

The $FSTk$ Library and Tools

In this appendix, we describe the Finite-State Toolkit ($FSTk$), which consists of both a library and a set of tools to manipulate $WFST$s in speech processing applications.

In the following section, we describe the library, including its design and functionality. In section A.2 the command line tools are described.

A.1 Finite-State Library

The development of the library was initially motivated by our need to efficiently manipulate $WFST$s in our speech decoder (Chapter 6). As such, it includes not only $WFST$ operations but also interface operations with knowledge sources used in speech recognition. Apart from the decoder itself, the library includes the algorithms developed in the context of this thesis, including the specialized composition algorithms presented in Chapter 7.

A.1.1 Design

Having been designed from the start to be used in speech processing, the library allows the efficient use of very large automata with up to tens of millions of states, and the use of large vocabularies.
The design of the library follows a blend of object-oriented and abstract data-
type approaches. The two approaches operate at different abstraction layers. In
the higher layer, the WFSTs are an abstract data type which is manipulated using
non-destructive compositional operations. This means that the operations do not
change the state of its arguments and have no internal state or side effects, so that
the result of the operations depends only on their arguments. In the inner layer, the
entities manipulated by this compositional operations are themselves represented
using an object oriented approach, in which their ultimate internal representation
is still hidden, but the methods which manipulate the entities can have side effects
or change the internal state of objects.

The object-oriented layer allows the efficient use of multiple internal implemen-
tations for WFSTs. This allows not only various space/time tradeoffs but also the
use of specialized objects which behave like WFSTs, but are implemented with the
full Turing-machine power of C++.

Transducer Representation

Following the approach of the AT&T FSM Libray [99], at the highest levels of the
library, a transducer is an entity which responds to three fundamental operations:

Start which returns the initial state,

Final(q) which returns the final weight of the state q, or ∞ if the state is not final,

Acs(q) which returns the set of edges which follow state q.

These three functions are the basis of the “on-the-fly” representation of trans-
ducers, since they can build the WFST as needed by client programs.

The edges returned by Acrs are accessed using an iterator object. Each edge
contains its initial and destination states, one input and one output labels and
one weight. No distinct representation is provided for automata or non-weighted
transducers. An automata is represented and treated as an identity transducer.
The states and labels are represented using 32 bit integers, but the library provides for dictionaries which convert those integers to and from strings or arbitrary C++ objects.

Weights are represented using floating point numbers. Only numeric semi-rings are currently supported.

A.1.2 Functionality

The functionality of the library can be divided into various groups of operations: access operations, finite-state operations, representation operations, graph and search operations, conversion operations and specialized operations.

In the following sections we describe the functionality in each of these groups.

Access Operations

The transducer graph is accessed using the Start, Final, and Arcs operations described in Section A.1.1.

Finite-State Operations

Union Computes the union of two transducers,

Concat Computes the concatenation of two transducers,

Compose Composes two transducers,

Closure Returns the concatenative closure of a transducer,

Reverse Reverses the transducer,

Invert Returns the inverse transducer,

Project Returns either the input or the output projection automaton of a transducer,
RmEpsilons  Removes $\epsilon$ transitions,

Determinize  Returns a sequential equivalent transducer, if it exists,

Minimize  Minimizes a sequential transducer,

Compact  Determines and minimizes the underlying automaton of a transducer,

PushWeights  Pushes the weights of the transducer towards the initial (or final) states,

PushOutputLabels  Pushes the output labels of the transducer towards the initial (or final) states.

Finite-State Representation

Copy  Copies the transducer, eventually to a transducer with a different internal representation,

Cache  Caches the access to another transducer by storing the results of the Arcs operation, storing only a limited number of results,

Memoize  Like Cache, but stores all accesses to the Arcs operation.

Graph and Search Operations

BestPath  Returns the best path in a transducer,

BestNPaths  Returns the $n$ best paths is a transducer,

Viterbi  Returns the best path in a transducer using the breath-first Viterbi algorithm with beam pruning,

Connect  Removes non-useful states from the transducer,

Prune  Uses a forward-backward algorithm, to remove from the transducer all paths whose cost it over a beam of the best one.
Conversion Operations

The library supports multiple file formats which are standard in the speech recognition community. Those files represent various components of knowledge sources of a speech recognition system:

**BuildFromSentence** Given a string and a dictionary, builds a linear identity transducer representing the string.

**LoadFromATTFSM** Reads a WFST in the textual file format of the AT&T FSM tools,

**LoadFromARPAFile** Converts an $n$-gram language model specified in the ARPA format to a transducer,

**LoadFromWNetFile** Converts a lattice in the WNet format to a transducer,

**LoadFromLexFile** Converts a lexicon file to a linear transducer representation,

**LoadFromNOWAYPhoneModels** Converts HMM models in the format of the NOWAY decoder to a transducer,

**LoadFromHTKPhoneModels** Converts HMM models in the format of the HTK system to a transducer.

Specialized Operations

**ComposeLexLang** Implements various versions of the specialized composition algorithm presented in Chapter 7,

**ShareLMLContexts** Implements the language model heuristic minimization algorithm presented in [29],

**Bigram** Returns an "on-the-fly" transducer representing the composition of another transducer with all possible bigrams, thus allowing the "on-the-fly"
generation of context dependency transducers independently of the transducer alphabet size,

BuildStringEditDistanceFilter Builds a string-edit distance filter transducer using standard weights or from a distance matrix,

StringEditDistance Computes a transducer containing all the string-edit alternatives between two transducers.

A.1.3 Implementation

All “on-the-fly” operations are implemented at the object-oriented level as WFST subclasses of a single abstract WFST class.

Non “on-the-fly” operations are implemented as methods of classes which explicitly represent the transducer in memory.

Various explicit representations are available, each with a different tradeoff between speed and memory consumption.

Our basic representation is based on an adjacency-list representation of the WFST graph. The main data structure of that representation is a vector, containing all edges, primarily sorted by origin state. Each edge in the vector is a 5-tuple containing: the identification of the origin state; the destination state; the input label; the output label; and the weight. This tuple occupies 20 bytes of memory. Another vector contains the 4-byte offset to the first edge that leaves a particular state. Thus the memory required by the basic graph representation is \(4|Q| + 20|E|\) bytes, where \(|Q|\) is the number of states and \(|E|\) the number of edges.

Besides this explicit representation, other representations are implemented, some of which are not mutable and only provide the three access operations, but consume very little memory. One such representation used to represent very large scale transducers is described in the following section.
Compact Representation of WFSTs for ASR

Even when fully optimized, the WFSTs used in large vocabulary tasks can be very large. In order to reduce the runtime memory requirements, when using a large static network, we developed a memory efficient representation for WFSTs [29].

This compact representation uses a variable length representation for edges that takes into account the properties of the typical integrated network used in speech recognition. The main problem with a variable length representation is that it more difficult to directly access the edges that leave a given state, as direct indexing would still require 4 bytes per state (for a pointer or an offset). We reduced the memory required per state to 16.5 bits, by grouping the states in groups of 64 states. A master index \( M \) contains a 32-bit pointer to a chunk of memory containing the edges of 64 states. A separate offset index \( O \) contains the 16-bit offset of the first edge of the states in the chunk. Thus, the first edge of state \( q \) is the \( O[q] \) edge stored in the chunk \( M[q/64] \). The indexation scheme is illustrated in figure A.1.

![Compact WFST Indexation Scheme](image)

**Figure A.1:** Compact WFST Indexation Scheme.

Each edge contained in a chunk occupies an integral number of bytes that can vary from 1 to 10. A variable-length encoded edge is a sequence \( FDLW \), where:

- \( F \) is a record of 5 bits that specifies how the edge is encoded: 2 bits specify
the encoding of the destination state; 2 bits the encoding of the input and output labels; 1 bit specifies the encoding of the weight.

- \( D \) encodes the destination. The destination is encoded as the difference between the origin and destination states of the arc. It can be encoded in 3, 11, 19 or 27 bits. 3 of the bits are stored in the same byte as \( F \).

- \( L \) encodes the input and output labels. If both the input and output labels are \( \epsilon \), it is encoded in 0 bytes; if the output is \( \epsilon \) and the input is less than 256 it is encoded in one byte; if the output is less than 65536 and the input is \( \epsilon \), it is encoded in 2 bytes; otherwise the input and the output are stored in 4 bytes (the number of bits assigned to each label is determined from the size of the corresponding alphabets).

  When the automaton is an acceptor, only the input label is stored. It is stored in 0 bytes if equal to \( \epsilon \); 1 byte, if less than 256; 2 bytes, if between 256 and 65535; 3 bytes, if between 65536 and \( 2^{24} - 1 \).

- \( W \) encodes the weight. If zero, it is encoded in 0 bytes; otherwise it is stored in 2 bytes, as an integer obtained by linearly scaling between the minimum and maximum weights of the WFST.

To access an edge in the chunk, given its offset, all the previous edges in the chunk need to be traversed. In order to alleviate the overhead associated with the expansion and access to the edges, we use a cache that stores the expanded set of edges that leaves the most recently accessed states.

We performed several experiments using different European Portuguese, and North American English, large-vocabulary search networks and language models. We observed that using this representation, we achieved an average of 5.2 bytes per edge when encoding (context-independent) static search networks. When encoding language models, we achieved an average of 7.2 bytes per edge.
A.2 Command Line Tools

FSTk contains a set of command line tools, which can be used to manipulate WFST files. These commands are useful for rapid prototyping directly in the command line or through scripts. The commands available correspond almost directly to the operations described above in Section A.1.2. The main exception is that “on-the-fly” transducers do not yet have a file representation, and while some operations are performed “on-the-fly”, their output is always explicitly generated.
BIBLIOGRAPHY


186


188


