Novel Speech Recognition Models for Arabic

Johns-Hopkins University Summer Research Workshop 2002

Final Report

Contact:

Katrin Kirchhoff
University of Washington
Dept. of Electrical Engineering
Box 352500
Seattle, WA, 98115
katrin@ee.washington.edu
(206) 221-5476

Team website: http://ssli.ee.washington.edu/people/katrin/arabic_resources.html
Workshop website: http://www.clsp.jhu.edu/ws02

Workshop Team Members and Affiliates

Katrin Kirchhoff, University of Washington

Jeff Bilmes, University of Washington

John Henderson, MITRE

Richard Schwartz, BBN

Mohamed Noamany, BBN

Pat Schone, Department of Defense

Gang Ji, University of Washington

Sourin Das, Johns Hopkins University

Melissa Egan, Pomona College

Feng He, Swarthmore College

Dimitra Vergyri, SRI

Daben Liu, BBN

Nicolae Duta, BBN

Contents

1 Introduction: Problems in Arabic Speech Recognition

This is the final report of the 2002 Johns-Hopkins Summer Workshop effort on developing novel speech recognition models for Arabic. Our efforts concentrated on conversational Arabic, for which few standardized speech resources are available. Therefore, our two major goals were (a) to develop techniques for making formal Arabic speech data usable for conversational speech recognition, and (b) to design new statistical models that exploit the available data as well as possible. Our work included techniques for the automatic romanization of Arabic script, a new type of language model for highly inflected languages called *factored language model* and experiments on using formal Arabic text data for conversational Arabic language modeling.

In this chapter we will first give an overview of the Arabic language and Arabic-specific problems for automatic speech recognition. Sections 1.3 and 1.4 then present previous work on Arabic ASR and an overview of the topics addressed during the workshop.

1.1 The Arabic Language

Arabic is currently the sixth most widely spoken language in the world. The estimated number of Arabic speakers is 250 million, of which roughly 195 million are first language speakers and 55 million are second language speakers. Arabic is an official language in more than 22 countries. Since it is also the language of religious instruction in Islam, many more speakers have at least a passive knowledge of the language.
Linguistic Varieties

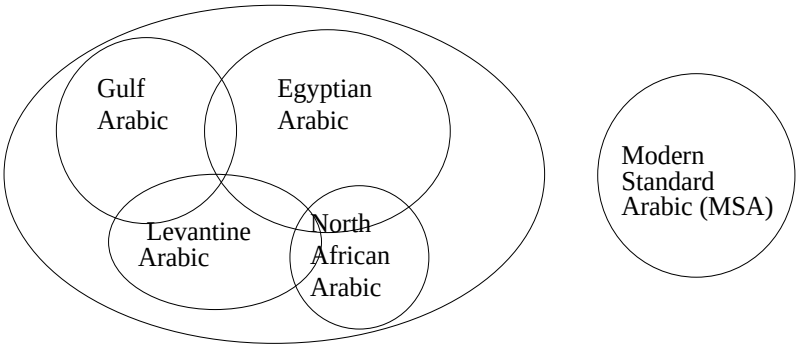

Figure 1: Arabic linguistic varieties.

It is important to realize that what we typically refer to as “Arabic” is not single linguistic variety; rather, it is a collection of different dialects and sociolects, as shown in Figure 1. Classical Arabic is an older, literary form of the language, exemplified by the type of Arabic used in the Quran. Modern Standard Arabic (MSA) is a version of Classical Arabic with a modernized vocabulary. MSA is a formal standard common to all Arabic-speaking countries. It is the language used in the media (newspapers, radio, TV), in official

speeches, in courtrooms, and, generally speaking, in any kind of formal communication. However, it is *not* used for everyday, informal communication, which is typically carried out in one of the local dialects.

The dialects of Arabic can roughly be divided into two groups: Western Arabic, which includes the dialects spoken in Morocco, Algeria, Tunisia, and Libya, and Eastern Arabic, which can be further subdivided into Egyptian, Levantine, and Gulf Arabic. These various dialects differ considerably from each other and from Modern Standard Arabic. Differences affect all levels of language, i.e. pronunciation, phonology, vocabulary, morphology, and syntax. Table 1 lists examples of the differences between Egyptian Colloquial Arabic (ECA) and Modern Standard Arabic. ECA is that dialect which is most widely understood through-

Change	MSA	ECA	Gloss
/θ/ → /s/,/t/	/θala:θa/	/tala:ta/	ثلاث ’three’
/ð/ → /z/,/d/	/ðahab/	/dahab/	ذهب ’gold’
/ay/ → /e:/	/ʂaif/	/ʂe:f/	صيف ’summer’
inflections	yatakallam(u)	yitkallim	يتكلم ’he speaks’
vocabulary	Tawila	tarabeeza	’table’
word order	VSO	SVO	

Table 1: Some differences between Modern Standard Arabic and Egyptian Colloquial Arabic.

out the Arabic-speaking world, due to a large number of TV programmes which are produced in Egypt and exported to other Arabic countries. Native speakers from different dialect regions are for the most part capable of communicating with each other, especially if they have had some previous exposure to the other speaker’s dialect. However, widely differing dialects, such as Moroccan Arabic and the Iraqi dialect, may hinder communication to the extent that speakers adopt Modern Standard Arabic as a lingua franca.

Writing System

Arabic is written in script and from right to left. The alphabet consists of twenty-eight letters, twenty-five of which represent consonants. The remaining three letters represent the long vowels of Arabic (/i:/,/a:/,/u:/) and, where applicable, the corresponding semivowels (/y/ and /w/). Each letter can appear in up to four different shapes, depending on whether it occurs at the beginning, in the middle, or at the end of a word, or in isolation. Letters are mostly connected and there is no capitalization. Table 2 lists the alphabet letters in their various forms. Some additional symbols have been introduced for certain phones not present in the MSA alphabet, such as ڤ for /v/, which is primarily used in loanwords.

A distinguishing feature of the Arabic writing system (and of related systems, such as that used for Hebrew) is that short vowels are not represented by the letters of the alphabet. Instead, they are marked by so-called *diacritics*, short strokes placed either above or below the preceding consonant. Several other pro-

Isolated	Beginning	Middle	End	Name	Phoneme
ا	ا	ا	ا	’alif	/aː/
ب	بـ	ـبـ	ـب	baa’	/b/
ت	تـ	ـتـ	ـت	taa’	/t/
ث	ثـ	ـثـ	ـث	thaa’	/θ/
ج	جـ	ـجـ	ـج	gym	/ʤ/
ح	حـ	ـحـ	ـح	Haa’	/ħ/
خ	خـ	ـخـ	ـخ	khaa’	/x/
د	د	ـدـ	ـد	daal	/d/
ذ	ذ	ـذـ	ـذ	dhaal	/ð/
ز	ز	ـزـ	ـز	zayn	/z/
ر	ر	ـرـ	ـر	raa	/r/
س	سـ	ـسـ	ـس	syn	/s/
ش	شـ	ـشـ	ـش	shyn	/ʃ/
ص	صـ	ـصـ	ـص	Saad	/sˁ/
ض	ضـ	ـضـ	ـض	Daad	/dˁ/
ط	طـ	ـطـ	ـط	Taa’	/tˁ/
ظ	ظـ	ـظـ	ـظ	Zaa’	/zˁ/
ع	عـ	ـعـ	ـع	’ayn	/ʕ/
غ	غـ	ـغـ	ـغ	ghayn	/ʁ/
ك	كـ	ـكـ	ـك	kaaf	/k/
ق	قـ	ـقـ	ـق	qaaf	/q/
ف	فـ	ـفـ	ـف	faa’	/f/
ل	لـ	ـلـ	ـل	laam	/l/
ن	نـ	ـنـ	ـن	nuwn	/n/
م	مـ	ـمـ	ـم	mym	/m/
ه	هـ	ـهـ	ـه	haa’	/h/
و	و	ـو	ـو	waaw	/uː/
ي	يـ	ـيـ	ـي	yaa’	/iː/
ء	أ	ئ	ـئ	hamza	/ʔ/

Table 2: Letters of the Arabic alphabet. The Arabic letter names are given in Qalam romanization [44]; the corresponding phonemes are shown in IPA notation [3].

nunciation phenomena are marked by diacritics, such as consonant doubling (phonemic in Arabic), which is indicated by the “shadda” sign, and the “tanween”, word-final adverbial markers which add /n/ to the pronunciation. The entire set of diacritics is listed in Table 3, along with the small number of special graphemes for certain letter combinations. Apart from these special graphemes, the grapheme-phoneme correspondence is one-to-one. It should be noted that Arabic texts are almost never fully diacritized; normally, diacritics are

Example	Symbol Name	Meaning
اَ	fatHa	/a/
اِ	kasra	/i/
اُ	Damma	/u/
رّ	shadda	consonant doubling
درْس	sukuwn	absence of vowel after consonant
اً	tanwyn al-fatHa	/an/
اٍ	tanwyn al-kasr	/in/
اٌ	tanwyn aD-Damm	/un/
ى	’alif maqsuwra	/a:/ sound, historical
هٰذه	dagger ’alif	/a:/ sound, historical
آ	madda	double alif
في ٱلبيت	waSla	on ’alif in *al*
لا	laam ’alif	combination of laam and ’alif
ة	taa marbuwta	morphophonemic marker

Table 3: Arabic diacritics

used sparingly and only to prevent misunderstandings. Exceptions are important religious and/or political texts, such as the Quran, translations of the Bible and the Declaration of Human Rights, which are published with full diacritics. Furthermore, beginners’ texts, such as children’s schoolbooks, may be diacritized, although they tend to be omitted as soon as the learner has a basic grasp of the language. The lack of diacritics may lead to considerable lexical ambiguity. These ambiguities must be resolved by contextual information, which presupposes knowledge of the language. Without this knowledge, it is even impossible to determine how to pronounce a non-diacritized text. As an illustrative example, an English sentence from which all short vowels have been deleted is shown in Figure 2. In normal English this sentence reads “The fish stocks in the North Atlantic have been depleted”. Considered in isolation, the word “stcks” could be interpreted as “sticks”, “stacks”, or “stocks”. Contextual information is necessary in order to determine the appropriate vowelization and thereby the pronunciation and the meaning of the word. The situation in Arabic is very

The fsh stcks n th Nrth tlntc v bn dpletd

Figure 2: Example of an English sentence without short vowels.

similar; however, non-diacritized word forms may have even more diacritized counterparts; the form كتب (ktb), for instance, has 21 potential diacritizations (see further Section 3).

Since Arabic dialects are primarily oral varieties they do not have generally agreed upon writing standards. Whenever there is the need to render dialectal speech in writing, speakers will use a phonetic spelling for non-MSA words, using the letters of the standard alphabet.

The Arabic writing system is also used for a number of other languages (sometimes with modifications), such as Farsi, Urdu, Hausa, and Pashto.

Several efforts have been made at representing Arabic script using Western (Roman) character sets; this is usually referred to as *transliteration* or *romanization*. Table 4 shows different transliteration schemes.

Finally, three binary encoding standards for Arabic characters have emerged: ISO 8599-6 [34], the Windows-based CP-1256 [16], and the more recent Unicode standard [51].

Phonology Arabic has thirty-one phonemes, those represented in Table 2 plus the three short vowels (/i/,/a/,/u/), which are counterparts of the three long vowels (/i:/, /a:/, /u:/). Vowel length is phonemic. Some Arabic dialects may have additional or fewer phonemes. Egyptian Arabic, for instance, lacks /ð/ and /θ/ and replaces /ʤ/ with /g/. Like other languages, Arabic has a set of phonological alternations that depend on the surrounding context. Two of the most pervasive alternations are the spread of pharyngealization or emphasis, and the change of "taa marbuwta". A subset of the Arabic consonants (/ŧ/,/ɖ/, /ʂ/,/ʐ/), the so-called *emphatics*, are pharyngealized forms of the consonants (/t/,/d/,/s/,/z/). In their vicinity, neighbouring sounds take on a pharyngealized quality as well. According to [22], this effect may spread over entire syllables and beyond syllable boundaries. Taa marbuwta, represented by ة, is a feminine grammatical marker which is pronounced /a/ in isolation or before a following consonant, but as /at/ before a vowel (this phenomenon is similar to French liaison but is limited to taa marbuwta in Arabic). Other examples of phonological alternations are the assimilation of the /l/ in the definite article /al/ to following homorganic consonants (/s/,/z/,/t/,/d/,/n/,/ŧ/,/ɖ/, /ʂ/,/ʐ/,/ð/,/θ/,/ʃ/,/r/, the so-called *sun letters*) and the elision of the second of two successive vowels (e.g. *fi al bait ('in the house')* → *fil bait*).

Morphology

Much of the complexity of the Arabic language is found at the morphology level. Many Arabic open-class words (nouns, verbs, adjectives, and, to some extent, adverbs) have the structure depicted in Figure 3, where a word consists of a stem surrounded by prefixes and/or suffixes. Affixes signal grammatical categories such as person, number and gender, as illustrated in Table 5. The stem itself can further be decomposed into a *root* (sequence of three consonants) and a *pattern* of vowels and, possibly, additional consonants. The root

letter	**Qalam** [44]	**CAT** [45]	**LDC**	**Karboul** [38]	**Buckwalter** [12]
’alif	aa	aa	aa	A	A
baa’	b	b	b	B	b
taa’	t	t	t	TH	t
thaa’	th	c͡/th	th	Tt	v
gym	j	j	j	J	j
Haa’	H	h/H	h	H	H
khaa’	kh	k/kh/K	kh	Kk	x
daal’	d	d	d	D	d
dhaal’	dh	zzh	dh	Dd	P
raa’	r	r	r	R	r
zayn	z	z	z	Z	z
syn	s	s	s	S	s
shyn	sh	s͡/sh	sh	Sc	$
Saad	S	s/S	S	Ss	S
Daad	D	d/D	D	Sd	D
Taa’	T	t/T	T	Td	T
Zaa’	Z	z/Z	Z	Dt	Z
‘ayn	‘	@	c	Ar	E
ghayn	gh	g͡/gh	R	G	g
faa’	f	f	f	F	f
qaaf	q	q	q	Q	q
kaaf	k	k	k	K	k
laam	l	l	l	L	l
mym	m	m	m	M	m
nuwn	n	n	n	N	n
haa	h	h	h	h	h
waaw	w	w/uu/oo	w	W	w
yaa’	y	y/ii	y	Y	y
hamza	’	’	C	unclear	’

Table 4: Various transliteration schemes for Arabic.

and the pattern are interspersed according to the possible consonantal slots specified in the pattern. The root is comparable to a lexeme; it gives the basic meaning to the word. The root *d-r-s*, for instance, indicates

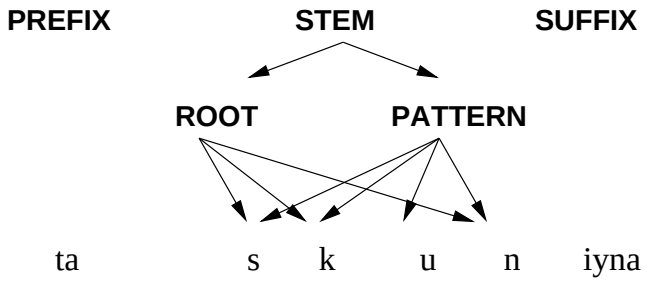

Figure 3: Morphological structure for *taskuniyna (you (f.sg.) live)*.

Word	Meaning	Word	Meaning
’a-skun(u)	I live	kitaab-iy	my book
ta-skun(u)	you (masc.) live	kitaabu-ka	your (masc.) book
ta-skun-iyna	you (fem.) live	kitaabu-ki	your (fem.) book
ya-skun(u)	he lives	kitaabu-hu	his book
ta-skun(u)	she lives	kitaabu-haa	her book
na-skun(u)	we live	kitaaabu-na	our book
ta-skun-uwna	you (masc.pl.) live	kitaabu-kum	your book
ya-skun-uwna	they live	kitaabu-hum	their book

Table 5: Examples of MSA pronominal and possessive affixes (separated from stem by ’-’).

that the word has something to do with ’study’, *k-t-b* has the basic meaning of ’write’, *s-k-n* is common to words related to ’live’, etc. The vowels and consonants of the pattern have a weak semantic meaning in that they signal voice, tense, causality, etc. Tables 6 and 7 list examples of words derived from the same root and different patterns. There are about 5000 roots in Arabic (most are triliteral, some are quadriliteral),

Root KTB	Root DRS
ka**t**a**b**a - he wrote	**d**a**r**a**s**a - he studied
ki**t**aa**b** - book	**d**a**r****s** - lesson
ku**t**u**b** - books	**d**u**r**uu**s** - lesson
’a**kt**u**b**u - I write	’a**dr**u**s**u - I study
ma**kt**a**b** - desk, office	ma**dr**a**s**a - school
ma**kt**a**b**a - library	mu**d**a**r**ri**s** - teacher
kaa**t**i**b** - writer	**d**a**r**ra**s**a - he taught

Table 6: Words related to the roots KTB (’write’) and DRS (’study’). Root consonants are marked in boldface.

several hundred patterns, and dozens of affixes. Although not all of the components can combine freely, the resulting number of possible word forms is enormous. Additionally, a small number of particles can attach to the beginning of the word, thus increasing the number of word forms even further. These are most notably the definite article *al*, as well as conjunctions and prepositions like *fa* ('so','and'), *li*, ('for', 'in order to'), or *bi* ('in', 'with').

Syntax/Grammar

Number	Pattern	Function/Meaning
I	CVCVC	base pattern
II	CVCCVC	causation, intensification
III	CVVCVC	doing something jointly
IV	'aCVCVC	causation
V	taCVCCVC	reflexive of II
IV	taCVVVCVC	reflexive of III
VII	inCVCVC	passive
VIII	iCtaCVC	no consistent meaning
X	istaCCVC	to ask for something

Table 7: Examples of past-tense verbal patterns.

Compared to the derivational morphology described above, the syntax of Arabic is less complex. Arabic has two genders (masculine, feminine), three numbers (singular, plural, dual), three cases (subject case, object case, prepositional object case) and two morphologically marked tenses. There is noun-adjective agreement for number and gender and subject-verb agreement in SVO word order. Whereas SVO is the dominant word order for MSA, SVO is preferred in dialects. Pronouns are rarely used; pronominal functions are instead signaled by the verbal morphology.

1.2 Problems for Automatic Speech Recognition

Many aspects of Arabic, such as the phonology and the syntax, do not present problems for automatic speech recognition. Standard, language-independent techniques for acoustic and pronunciation modeling, such as context-dependent phones, can easily be applied to model the acoustic-phonetic properties of Arabic. Some aspects of recognizer training are even easier than in other languages, in particular the task of constructing a pronunciation lexicon since there is a nearly one-to-one letter-to-phone correspondence.

The most difficult problems in developing high-accuracy speech recognition systems for Arabic are the predominance of non-diacritized text material, the enormous dialectal variety, and the morphological complexity.

The constraint of having to use mostly non-diacritized texts as recognizer training material leads to problems for both acoustic and language modeling. First, it is difficult to train accurate acoustic models for short vowels if their identity and location in the signal is not known. Second, the absence of diacritics leads to a larger set of linguistic contexts for a given word form; language models trained on this non-diacritized material may therefore be less predictive than those trained on diacritized texts. Both of these factors may lead to a loss in recognition accuracy. As further described in Sections 2.3 and 3, we have observed that ignoring available vowel information does indeed lead to a significant increase in both language model perplexity and word error rate.

Dialectal variety is a problem primarily because of the current lack of training data for conversational Arabic. Whereas MSA data can readily be acquired from various media sources, there is only one standardized speech corpus of dialectal Arabic available, viz. the "CallHome" corpus of Egyptian Colloquial Arabic distributed by the Linguistic Data Consortium (LDC), consisting only of 15 hours of transcribed training material.

Finally, morphological complexity is known to present serious problems for speech recognition, in particular for language modeling. A high degree of affixation, derivation etc. contributes to the explosion of different word forms, making if difficult if not impossible to robustly estimate language model probabilities. Rich morphology also leads to high out-of-vocabulary rates and larger search spaces during decoding, thus slowing down the recognition process.

1.3 Previous Work in Arabic Speech Recognition

Previous work in Arabic ASR has almost exclusively focused on the recognition of Modern Standard Arabic. Some of this work was done as part of the development of dictation applications, such such as the IBM ViaVoice system for Arabic. More recently, information extraction from Broadcast News (the BBN Tides-OnTap system [25, 26]) has emerged as an application for Arabic ASR. The current performance of the BBN Arabic Broadcast News recognizer is around 15% word error rate (WER).

To our knowledge, the only large-scale effort at developing recognizers for conversational (i.e. dialectal) Arabic as opposed to MSA was performed within the framework of the NIST CallHome benchmark evaluations in 1996 and 1997. The objective of these evaluations was a comparison of recognition systems for conversational telephone speech in several languages, one of which was Egyptian Colloquial Arabic. The systems developed by BBN [24, 23] were the most successful in these evaluations and obtained word error rates ranging between 71.1% [24] and 61.6% [23] WER. A more recent version of a BBN-developed recognizer achieved 55.8% on the same task - this system also served as the baseline system for WS02 and is described in greater detail below (Section 2.3).

1.4 Workshop Goals

The performance of current Broadcast News recognizers for Arabic is satisfactory; error rates on conversational speech, by contrast, are unacceptably high. The currently best error rate, 55.5%, is larger than those for comparable data in other languages (see Table 8). For this reason we decided to focus entirely on the recognition of conversational Arabic during the WS02 workshop. Our overall goal was to identify and tackle problems that arise from the Arabic language; we explicitly decided to ignore those problems that are common to all large-vocabulary conversational speech recognition tasks, such as speaker variability, disfluencies, noise, etc.

Corpus	WER
English CallHome	32.6% (CU-HTK 2000)
Mandarin CallHome	49.9% (CER) (BBN 2001)
Arabic Callhome	55.8% (BBN 2002)

Table 8: Current error rates for CallHome corpora in various languages. The numbers shown for Mandarin are character error rates, all others are word error rates.

Our intention was to develop solutions for the three Arabic-specific problems described above, viz.

- the mismatch between spoken and written representation (missing pronunciation information in Arabic script);
- the lack of conversational training data;
- morphological complexity.

Solutions to the first two problems could be used to render additional, out-of-corpus material (in particular non-diacritized and MSA texts) usable for training acoustic and language models for CallHome. Methods for reducing the effect of morphological complexity on language modeling, on the other hand, help to exploit available data more efficiently.

As a solution to the first problem we investigated automatic romanization, i.e. the possibility of statistically predicting diacritization information by training on a small amount of aligned script and romanized data. The resulting predictor can then generate a larger set of diacritized data from script input, which can in turn be used as training data for a speech recognizer. This work is described in detail in Section 3. Second, we developed a new approach to language modeling for morphologically complex languages, so-called *Factored Language Modeling*, described in Section 5. Finally, in order to address the lack of conversational training data we explored various ways of using Modern Standard Arabic texts as additional language modeling data. These experiments are described in Section 7.

2 Data Resources, Baseline System and Evaluation Standards

In this section we present in detail the data resources, baseline systems and evaluation criteria used during the workshop.

2.1 LDC CallHome Corpus

For our work we used the only transcribed corpus of conversational Arabic currently available, viz. the LDC CallHome corpus of Egyptian Colloquial Arabic. This is a collection of informal phone conversations between close friends and family members; one speaker is located in the U.S., the other is located in Egypt. The majority of speakers come from the Cairene dialect region. The standard LDC distribution of this corpus is divided into three different sets: the training set (80 conversations), the development set (20 conversations), and the evaluation set (20 conversations). The latter was used as the evaluation data for the 1996 NIST evaluations and is therefore referred to as the eval96 set. Just before the beginning of WS02, two additional sets were made available to us by LDC: the 1997 evaluation set (20 conversations) and 20 new conversations which had not been released so far (referred to as the "h5_new" set). The data sets and sizes are summarized in Table 9. All sets were transcribed at LDC using two different transcription conventions. First, the data was

Set	# conversations	# words
train	80	146298
dev	20	32148
eval96	20	15584
eval97	20	17670
h5_new	20	16752

Table 9: LDC ECA CallHome data sets.

transcribed in "romanized" form. This is similar to a diacritized form of Arabic script transliterated using a Western character set. However, this representation contains more information than the corresponding diacritized Arabic script; for instance, symbols particular to Egyptian Arabic were included (e.g. g for /g/), whereas the script transcriptions contain MSA letters only. In general, the romanized transcriptions provide more information about actual pronunciation and are thus closer to broad phonetic transcriptions of speech.

The romanized transcriptions were subsequently converted to Arabic script using a semi-automatic procedure, such that parallel script and romanized transcriptions are available. Details of the transcription schemes can be found in the documentation of the corpus. Additionally, utterance segmentations were provided in the transcriptions.

Two lexicons accompany this corpus. The first (version v.05, 1997) simply consists of the unique words of the CH transcription (16641 entries). The second version (v.07, 1999) consists of 51202 entries; the

additional entries were derived from Badawi & Hinds' *Dictionary of Egyptian Arabic* [4]. The lexicon provides information about the script word form, the romanized word form, the pronunciation, stress, and morphological analysis (stem and grammatical categories) of the word, the word frequency information for the different data sets (training, development, evaluation) and the source (ie. the number of times the word occurred in any of the CH data sets or, where applicable, in the external dictionary). A sample lexical entry is shown below:

rom.	script	pron.	stress	morph.	train freq	dev freq	eval freq.	source
$Axix	صَاخّ	$%xix	10	$Axix:pple-act+masc-sg	0	0	0	B

The CallHome ECA corpus is characterized by a relatively high degree of disfluencies (9% of all word tokens), foreign (in particular English) words (1.6%), and noise events, all of which affect word error rate.

2.2 Other data resources

In addition to the CallHome speech material we had several text corpora available. These were mainly collections of newspaper articles and transcriptions of broadcast news speech from the Al-Jazeera TV station. A list of corpora and their sizes is given in Table 10.

Corpus	# words	type
An-Nahar	72 million	newspaper text
Al-Hayat	160 million	newspaper text
Al-Ahram	61 million	newspaper text
AFP	65 million	newspaper text
Al-Jazeera	9 million	TV shows

Table 10: Arabic text corpora.

2.3 Baseline Systems

We used two baseline recognition systems for WS02, one system based on the romanized transcriptions and a second system based on the script transcriptions. Both were developed at BBN prior to the workshop.
The starting point for this development was the BBN Oasis speech recognizer. Oasis is a real-time automatic Broadcast-News transcription system that has been successfully applied to Modern Standard Arabic. One of the benefits of this system is that it runs in real time, such that recognition results can be obtained very quickly. However, Oasis had been optimized for Broadcast News transcription, which is very different from recognition of telephone conversations. By developing and tuning Oasis for Arabic CallHome data, it was possible to examine the effectiveness of both the existing technologies in Oasis and those developed

for English CallHome research. Before the start of the workshop BBN performed several experiments comparing the performance of the OASIS and LVCSR systems[1] on this problem. The initial results showed that the LVCSR system had a significantly lower error rate, due to several differences in the types of processing and algorithms being used. The initial word error rate (WER) for was 64.1% for OASIS and about 56% for the LVCSR system. After about one month of quick development, the WER with Oasis was reduced to 56.6%. The algorithm and processing changes that resulted in the improvement are outlined below. All of the changes consisted of making the OASIS system more similar to the LVCSR system, while maintaining real-time processing.

Segmentation

Oasis has a built-in segmenter to automatically detect speaker boundaries and reject long music and silences that are common in broadcast news. The speaker boundaries are then used by cepstral mean subtraction (CMS) and speaker adaptation to normalize speakers. In the CallHome data, however, there are usually only two speakers and the conversation sides are presented on separate channels. As a result, automatic detection is not necessary. Furthermore, we observed that the original non-speech rejection actually rejected speech in CallHome data, which caused deletion errors. To avoid this problem, a new front-end was built that bypassed the segmenter, which resulted in a gain of 0.6%. Given the fact that the non-speech detection was not reliable, we changed the CMS scheme from normalizing speech/non-speech separately - the usual Oasis mode - to normalizing all the data together. This yielded another 1.1% gain. To take advantage of the knowledge that one side of the conversation is usually from the same speaker, we let the system implement CMS and speaker adaptation on the whole side rather than on each speaker turn, which yielded additional gains of 1.0% and 0.6%, respectively. Table 11 summarizes these improvements. The next step we took was to add

Condition	WER(%)
Baseline	64.1
Bypassing the segmenter	63.4
CMS on whole conversation side	62.4
Speaker adaptation on whole conversation side	61.8
CMS one-level	60.8

Table 11: Changes to Oasis system and resulting performance improvements on CallHome.

new technologies to Oasis, most of which had been developed for English Switchboard and CallHome data. They included vocal-tract-length normalization (VTL), LPC smoothing with 40 poles, split-init training, and Heteroscedastic Linear Discriminative Analysis (HLDA). All of these technologies provided additional

[1]The LVCSR system is a different recognition system for Arabic available at BBN.

gains that improved the final word error rate of Oasis on ECA CallHome to 56.6%. Table 12 shows these improvements.

Condition	WER(%)
VTL	59.0
LPC smoothing with 40 poles	58.7
Split-init training	58.1
HLDA	56.6

Table 12: Improvements obtained by adding algorithms from the LVCSR system.

The final two system changes (split-init training and HLDA) were not used for the workshop. The system output n-best lists (up to 100 hypotheses per utterance) which were used for rescoring. Table 13 shows the baseline word error rates for the different test sets and systems. The recognition results for eval96 after rescoring were 59.9% for the script-based system and 55.8% for the romanized system. This difference is statistically significant and shows that important information is lost when ignoring short vowel and other pronunciation information. After adding another 20 training conversations obtained from LDC at the beginning of the workshop, the error rates were reduced to 59.0% and 55.1%, respectively. We used the romanized

Test set	script	romanized
dev96	61.2%	57.4%
eval96	59.9%	55.8%

Table 13: Baseline word error rates for script-based and romanized systems.

system as the main workshop baseline system. Specifically, we used the n-best lists generated by the system and rescored them with various models developed during the workshop. The oracle word error rate (obtained by selecting the hypothesis with the fewest errors from the n-best list) for the eval96 set was 46%; the "anti-oracle" (random selection of a hypothesis) was 62.7%.

2.4 Evaluation Method

The evaluation criterion for our research was the word error rate on the 1996 CallHome evaluation set. For this task, there are two different types of transcriptions that can be used as a reference for computing word error rate, viz. the romanized transcriptions vs. the script transcriptions. There is considerable discussion about which of these should generally be used for evaluating the output of an Arabic speech recognition system. The difference is that script-based transcriptions do not distinguish between words that have the same (non-diacritized) written form but different acoustic forms, whereas a romanized system uniquely

identifies each spoken word.

The arguments in favour of using script references are that non-diacritized script is the most common and most natural representation of the language, i.e. the representation that native speakers find easiest to produce and/or read. In fact, native speakers have considerable difficulties in reading or producing romanized transcriptions. For this reason, romanized transcription efforts typically require an extended period of human labeler training (months of training were used for the LDC transcription effort), and the resulting transcriptions tend to contain inconsistencies. Furthermore, for many applications of ASR, such as information extraction, audio document classification, etc., script output is perfectly adequate. In general, whenever the recognition output will be further processed by a human reader with extended knowledge of Arabic, script output will be sufficient.

On the other hand, diacritized or romanized output enhances readability for speakers with a less advanced knowledge of Arabic. Finally, it may be claimed that, from a purely scientific point of view, the goal of speech recognition and evaluation should be to uniquely identify each word exactly as it was spoken, and to recover as much information as possible from the speech signal.

Since the romanized transcriptions were used as the reference standard during the 1996/1997 NIST evaluations we adopted this standard during WS02. However, this should not necessarily be taken as a recommendation for future evaluations.

#	script	romanized	meaning
1	كَتَبَ	kataba	he has written
2	كُتِبَ	kutiba	he had written
3	كُتُبْ	kutub	books
4	كَتْبْ	katb	a script
5	كَتَّبَ	kattaba	he made s.o. write
6	كُتِّبَ	kuttiba	to make s.o. write
7	كَتِّبْ	kattib	make s.o. write
8	كَتَبَّ	katabba	like slicing
9	كَتَبّ	katabb	like the slicing

Table 14: Possible diacritized dictionary forms of كتب, from [19]

.

3 Automatic Romanization

3.1 Motivation

As explained above, the lack of diacritics in standard Arabic texts makes it difficult to use non-diacritized text for recognizer training since the location and identity of short vowels is unknown. Not only does this affect acoustic model training; it also leads to considerable lexical ambiguity. In a recent study [19] it was observed that a non-diacritized dictionary form has on average 2.9 possible diacritized forms, whereas an actual Arabic text containing 23000 forms showed an average ratio of 1:11.6.[2] The form كتب (ktb), for instance, has 21 possible diacritizations, the nine dictionary forms shown in Table 14, as well as as the case-marked forms of the nouns (entries 3 and 4), which includes three forms marked by a short vowel and three forms marked by a tanwiyn, thus adding 12 forms to the nine already given by the dictionary.

The performance of our two baseline systems demonstrates the effect of information about the diacritics on word error rate: Table 15 shows the word error rates of the systems trained on script vs. romanized transcriptions when evaluated against both reference standards. The system trained on the romanized LDC transcriptions and evaluated against romanized transcriptions achieved 55.8% WER. When evaluated against the script reference, the WER went down to 54.9%. A system trained on script and evaluated against the script reference obtained 59.0% WER. The loss in recognition accuracy thus is 4.1% absolute. Note that it is nonsensical to evaluate a script-based system against a romanized reference since only those words will be correct that do not differ in their romanized and script forms.

[2]The study did not further specify exactly which dictionary and text were used for this count.

Training	Testing	
	tested on script	tested on romanized form
trained on script form	59.0%	–
trained on romanized form	54.9%	55.8%

Table 15: WERs for systems trained and tested on romanized vs. script form.

It is obvious that diacritic information is helpful - when the evaluation standard is a romanized or diacritized transcription, diacritized data is indispensable. However, even when the evaluation standard is a script reference, using diacritic information system-internally provides better recognition results.

Our goal is to find out to what extent it is possible to use automatically derived diacritic information in a speech recognizer for Arabic. It is reasonable to assume that the ratio of non-dicritized to diacritized forms will be lower in actual spoken language. In that case it may be possible to infer correct diacritizations from the surrounding context with a high degree of accuracy. Our goal is to train a statistical predictor on a small amount of parallel script and diacritized data, to apply it to a larger data set, and to use the output as training material for a speech recognizer. The result of this procedure can be evaluated with respect to two questions:

- Does a recognizer trained on automatically diacritized script perform better than a recognizer trained on non-diacritized script?
- Does a recognizer trained on automatically diacritized script perform as well as a recognizer trained on hand-diacritized script?

If this approach turns out to be successful, large out-of-corpus data sets can in principle be automatically diacritized and used for recognizer training in the future.

3.2 Preliminary Study

Some software companies (Sakhr, Apptek, RDI) have developed commercial products for the automatic diacritization of Arabic. However, these are targeted towards Modern Standard Arabic; there are no known products for Arabic dialects. Prior to the workshop we tested one such product, the Apptek diacritizer, on three different texts. One of these was a standard newspaper text from Al-Hayat, another was a beginner's text in Modern Standard Arabic, and the third was a dialogue from the ECA CallHome corpus. All texts were sent to Apptek without information about their sources. The diacritized versions that were returned were evaluated against a hand-diacritization performed by a native speaker of Arabic in the first two cases, and against a diacritized form derived from the romanized LDC transcription in the third case. For every consonant, there are five different diacritization possibilities: fatha (short /a/), damma (short u), kasra (short /i/), sukuwn (no vowel) or shadda (consonant doubling). In many cases where there was no vowel following

the consonant, the diacritizer did not output a sukuwn but simply did not produce a diacritic at all. These cases were not counted as errors, since they do not affect the pronunciation of the diacritized text. All those cases were counted as errors where here the true diacritic was a fatha, kasra, damma or shadda and the hypothesized diacritic did either not correspond to it or no diacritic was produced. Tanween diacritics at word endings were treated in the same way.

Diacritics at word ends signaling grammatical case endings deserve special treatment since their use is somewhat optional depending on the formality of the language and on the speaker. We therefore scored the two MSA texts twice, once with case diacritics and once without. It did not seem appropriate to consider case endings for the ECA text since they are almost never used in colloquial Arabic. The accuracy rates of the automatic diacritizer are shown in Table 16. Not surprisingly, accuracy drops considerably for the

Text	with case endings	without case endings
Al-Hayat text	80.5%	91.0%
beginner's text	72.3%	83.1%
ECA text	–	51.9%

Table 16: Accuracy rates for automatic diacritizer on a variety of texts.

dialectal text. A closer analysis revealed that the accuracy rate varied between approximately 30% and 60% for different passages. Most errors were due to vowel alternations in ECA compared to MSA (e.g. /i/-/a/) and unknown words.

We show in the remaining parts of this section that utilizing automatically romanized Arabic words adds more information to the training of an Arabic ASR system than the noise that is introduced by the errors in the automatic romanization. Specifically, an Arabic ASR system that is trained on the output of an automatic romanization system has lower word error rate than the ASR system trained with implicit disregard for the unwritten portions of the words.

3.3 Data

The data we used for this work was the CallHome ECA corpus (see Table 9). The ECA corpus has both script and romanized transcriptions for all data sets. An example of the aligned transcriptions for one turn are shown in Figure 17 Notice that romanization is not equivalent to diacritization. The LDC romanization

```
script:    AlHmd_llh kwIsB       w  AntI AzIk
 roman: IlHamdulillA kuwayyisaB~ wi inti izzayyik
```

Table 17: Example script and romanized transcriptions.

contains more information (usually of a phonetic nature) than the corresponding Arabic diacritized forms

would. For instance, taa marbuwta is explicitly marked as to whether it is pronounced /a/ or /at/. This information would not be indicated by a graphemic marker but would simply be inferred from the context even in dicritized Arabic.

Since one of the motivations for our experiments was to test how well a speech recognizer would work when trained from a preponderance of automatically romanized data, we did not use the ASR training, development or eval96 set for training our automatic romanization (AR) system. Instead, we used the `eval97` and `h5new` sets to train the AR and used it to prepare a romanized form of the ASR training set, from the grapheme-only version. When configured this way, our experiments give a *lower bound* on the performance we would expect from an ASR system trained on a corpus consisting of only a little bit of data in the true romanized form and the majority of the data produced from automatic romanization. In this case, that majority is 100%.

We also run an ASR experiment in a more realistic scenario in which the AR system is trained on `eval97` and `h5new` as before, but the ASR system is trained on a corpus consisting of automatically romanized versions of `asrtrain` and `h5new`. In this setting, the majority of the ASR training data is still automatically romanized, but the portion is only 80%. That is probably a more realistic ratio currently, but we expect that in the future, as more data is transcribed and less of it in transcribed in romanized form, the other experimental setting will be valuable.

3.4 Metrics

In measuring Arabic ASR performance, one can consider either the romanized or non-romanized forms of words. This naturally leads to two different variations on the WER: WER measured on graphemes (words written without vowels) and WER measured on the full romanized form. We label the former as WERG (**G**rapheme) and the latter as WERR (**R**oman).

We measure the efficacy of our AR systems in two different ways, both measured in running text. Since we never add or remove entire words by altering their spelling, there is no need to count insertions or deletions. We can simply count the base word accuracy or word error as our first measure. Character errors are also measured to give a more fine-grained measure of our AR system's quality, as well as to give stronger statistics for noticing significant differences between systems. Character Error Rate (CER) is exactly the same as WER, but measured as if each character were a word.

In the limit as AR performs perfectly, both CER and accuracy are maximized at the same time. The AR process is prone to error, however, and the two metrics imply different impact on the larger ASR system. Higher word-level error rates are likely to cause higher out-of-vocabulary or LM back-offs, while higher character-level error rates suggest more noise for training acoustic models.

3.5 Autoromanization Baseline

An initial baseline AR system was created to empirically explore the data. The `eval97` and `h5new` datasets were used to train a simple maximum likelihood whole-word unigram model for producing the romanized forms from the raw script forms. This is in essence nothing more than a lookup table from each script form in the training set to the romanized form that was found most frequently in its place. Because the script and romanized versions of the corpus are in one-to-one correspondence, the table can be constructed without any effort spent on alignment.

status of words in training set	% of test words	% error in test	% of total test error
unambiguous	68.0	1.8	6.2
ambiguous	15.5	13.9	10.8
unknown	16.5	99.8	83.0
total	100.0	19.9	100.0

Table 18: Autoromanization baseline results

This model does not have full coverage of test set words. In order to cover those test set script forms that were not seen in the training set, the model *copies* the script form to the roman form. This is the most rational move in light of the absence of more information, a good baseline decision. In designing the model this way, we were also implicitly measuring the portion of the words that have identical script and roman forms – no short vowels or added pronunciation changes.

Table 18 shows the overall accuracy of the baseline autoromanization system. The majority of the words in the test set were unambiguous in the training set (68%), and only a few of them required a novel expanded form (1.8%). For those words, a context-independent maximum likelihood model is sufficient. Choosing the context-independent maximum likelihood expansion for the words were seen with multiple expansions in the training set gives 13.9% error on those words, which make up 15.5% of the test set. Finally, 16.5% of the words in the test set were not found in the training set and only 0.2% of those have no short vowels in them at all.

Overall, 83.0% of the baseline error comes from the unknown words, dominating the other two categories of words found in the test set. Making significant headway on this class of words is the main focus of these experiments in autoromanization because they represents the largest portion of the error, thus the class of largest potential error reduction.

3.6 Experiments

The baseline experiment described in Section 3.5 was repeated on the `dev` dataset instead of the `asr-train` dataset and characteristics of the unknown words were inspected in detail. It was observed that

approximately half of the tokens had at least one morphological variant in the training corpus. In other words, about half of the unknown tokens had at least one word in the training set with the same root as indicated by the full CallHome lexicon.

3.6.1 Knitting

The initial baseline AR system was sufficient for known words, and the maximum likelihood unigram strategy was adopted for all words that were seen in the training set. Several observations about Arabic morphology influenced our initial unknown word AR system design: Approximately half of the unknown words have at least one morphological variant in the training set. Some morphological variation is due to affixation. If an affixed version is seen in the training set, we would like to copy the vowels from that version to the novel, non-affixed form. If the non-affixed form is seen in the training set, then copying the vowels from it to a novel affixed form can get at least some of the vowels correct. Vowel *patterns* are part of the Arabic templatic morphology. Individual vowels are constrained by the patterns they are contained in, and not all possible patterns are used. For this reason, a model that is not able to capture the pattern dependencies would not be sufficient for this task. Initial exploratory experiments involving character HMMs and transformation rule lists using features of the local context supported this claim.

phase	Example	
Search	novel script	`tbqwA`
	⇒ known script	`ybqwA`
Record required edit operations	known script	`y bqwA`
	known roman	`yibqu`
	⇒operations	`ciccrd`
Apply operations	novel script	`t bqwA`
	known roman	`yibqu`
	operations	`ciccrd`
	⇒novel roman	`tibqu`

Figure 4: Producing romanized forms of novel script words.

Figure 4 diagrams the basic technique used for "knitting" the vowels from a close dictionary match into the novel script word (aka unknown word) to produce the novel romanized form. The process proceeds in three stages and utilizes the lookup table created from the training set for the maximum likelihood replacement of known script forms.

First, the known script form that is closest according to a weighted Levenshtein distance to the novel script form is found. Then the edit operations required to transform that known script form into its corresponding known romanized form are then recorded. The allowed edit operations are **c**opy, **r**eplace, **i**nsert,

and **d**elete. The insert operation is parametrized with the inserted symbol. The final step is to apply those edit operations to the novel script form. The matching in the first step produces an alignment of the novel script form with the known script form, allowing the sequence of edits to be applied.

3.6.2 Local model and N-best dictionary match

The analysis of errors in the baseline AR system produced several insights: first, certain characters in the script forms were consistently aligned with the same, non-identical characters in the corresponding romanized forms, representing consistent confusions. For instance, short /u/ only occurs between two non-vowels, /w/ occurs elsewhere. Additional transformation rules were introduced to handle those effects.

Instead of matching an unknown script form to the single closest known script form in the dictionary, the N-best closest script forms can be used. Each of these N-best script forms can be associated with a probability indicating the likelihood of the match. The corresponding romanizations of the N-best script forms each receive the same probability and the scores of identical romanized forms are summed up, i.e.

$$P(r_i) = \sum_j P(r_i, s_j) \tag{1}$$

where r_i is the i'th romanized form and s_j is the $j'th$ script form. Table 19 shows the improvements over the

Configuration	Word Acc. (%)	CER(%)
baseline	8.4	41.4
knitting	16.9	29.5
rules	19.4	27.0
n-best match (n=25)	30.0	23.1

Table 19: Summary of AR performance on `dev` set.

baseline obtained by adding transformation rules and the n-best dictionary match. Both word and character error rate are shown. The character error rate is a more fine-grained measure since it accommodates partially correct romanization results. Even partially correct results may be useful for ASR since they provide correct training data for some of the subword acoustic models.

3.6.3 Oracle result: n-best output

Table 20 shows which results can be obtained when the best dictionary matches are known, i.e. an oracle experiment. The oracle 2-best performance of 17.60% CER shows that there is additional room for improvement.

Configuration	Word Acc.(%)	CER(%)
baseline	0.21	49.9
multiple match (n=40)	20.59	24.58
oracle 2-best	32.24	17.60
oracle 3-best	38.27	14.62
oracle 4-best	42.17	12.90
oracle 5-best	45.09	11.78

Table 20: Summary of AR performance on `asrtrain` set.

3.6.4 Related work

Only one other empirical attempt to add diacritics to Arabic text is found in the literature: Gal [28] produced an HMM bigram-based language model for decoding diacritized sentences from non-diacritized ones. The technique was applied to the Quran and achieved 86% word accuracy. The data used in that work had only an 8% unknown word rate, whereas ours is in the range of 15-20%. It is natural that the technique developed in that work does not address unknown words. The focus of that work was different than ours, and utilized information about the context of a word. Performance gains may be had from combining insights of these two approaches.

3.7 Autoromanization and ASR

In this section we describe the use of the automatically romanized material for ASR training. There are two basic scenarios for using the automatic romanizer: preprocessing and postprocessing. In a preprocessing scenario the script data is first romanized and is then used to train the recognizer. The recognition output can be compared against a romanized reference transcription or can be stripped down to a script form if the reference is in script. In a postprocessing scenario the recognizer is trained on script and produces recognition output in script form. The automatic romanizer then converts the script output to a romanized form for the purpose of comparing it to a romanized reference form.

We investigated both possibilities; results for two different experimental conditions are shown in Tables 21 and 22. Table 21 shows results for the bounding experiment (i.e. the training sets for the romanizer and the speech recognizer do not overlap). Table 22 presents results for the overlap experiment, i.e. romanization and recognizer training sets overlap by a small amount (20 conversations). In both cases we see that using automatically romanized data for training helps compared to simply romanizing the output of a script-based system. It also reduces the word error rate when the final output is scored against a script reference. However, the system trained on automatically romanized data does not achieve the same performance as a system trained on manually romanized data. One problem with training on automatically romanized data is that pronunciations for new words (i.e. words that are not in the recognition lexicon, e.g. because they have

Configuration	WERR	Δ	WERG	Δ
script baseline	NA		59.8	–
AR postprocessing	61.5	–	59.8	–
AR preprocessing	59.9	-1.6	59.2	-0.6
romanized baseline	55.8	-5.7	55.6	-4.2

Table 21: Eval 96 ASR experiments, 80 training conversations.

Configuration	WERR	Δ	WERG	Δ
script baseline	NA		59.0	–
AR postprocessing	60.7	–	59.0	–
AR preprocessing	58.5	-2.2	57.5	-1.5
romanized baseline	55.1	-5.6	54.9	-4.1

Table 22: Eval 96 ASR experiments, 100 training conversations.

been wrongly romanized) need to be created, which is an additional error source.

3.8 Future Work

If one temporarily imagines that the two written forms of Arabic are really different languages, then the AR problem can be viewed as a special case of the more general problem of *transliteration*. Therefore, techniques for transliteration can be applied to this problem. Recent empirical approaches to machine transliteration include pioneering work on Japanese-English transliteration with composed finite state transducers for spelling and pronunciation [39], application of the same techniques to Arabic [49], and use of web-derived word frequencies and co-occurrences to filter n-best transducer output [2].

4 Morphology-Based Language Modeling

This section describes a several approaches to language modeling for Arabic that make use of morphological structure, viz. a particle model and morphological stream models.

4.1 Motivation

As explained above in Section 1.1, Arabic has a complex morphological structure which leads to a large number of possible word forms in the language. This affects the language modeling component adversely since it is infeasible to robustly estimate probabilities for all possible word sequences. Moreover, it leads to high out-of-vocabulary rates. An illustration of this of vocabulary growth problem is shown in Figures 5 and 6. Here the vocabulary growth rates (i.e. the number of unique words (vocabulary size) in relation to the number of word tokens in a text) on both the English and the Arabic CallHome corpora are displayed. It can be seen that the vocabulary growth rate of Arabic exceeds that of English by a large amount. Figure 6 shows the vocabulary growth rates for the stemmed versions of the same texts. The Arabic text was stemmed by looking up the stem for each word in the CallHome lexicon; the English text was stemmed by the Porter stemmer [43]. In both cases a reduction in vocabulary growth can be observed. The reduction in Arabic,

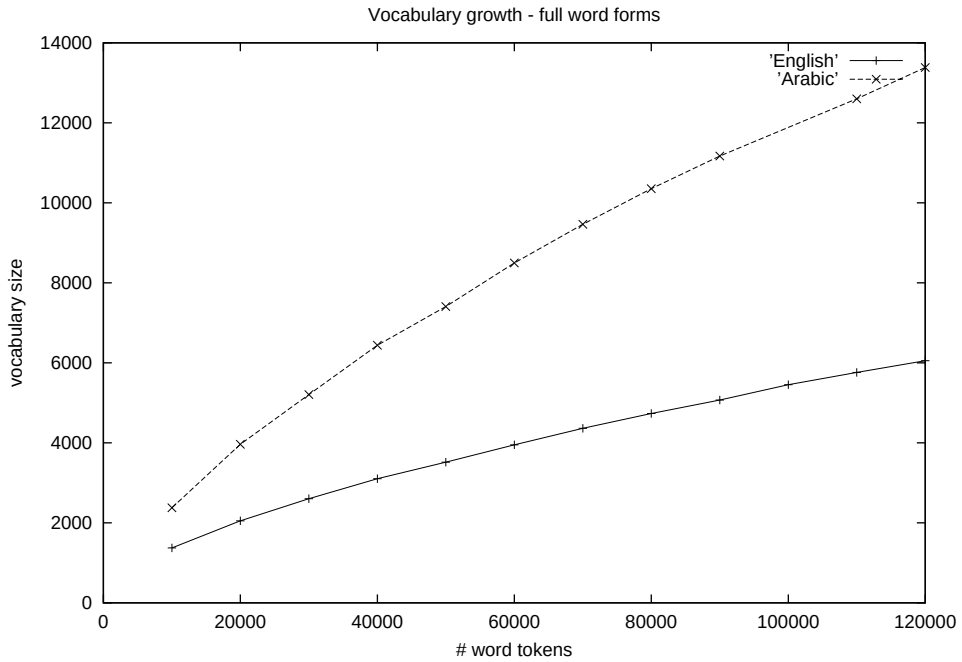

Figure 5: Vocabulary growth for full words in English and ECA.

however, is much greater than the reduction seen for English. This demonstrates that most the vocabulary growth in Arabic is indeed caused by the multiplication of word forms due to morphological affixation.

Given the difficulties in language modeling based on full word forms it would be desirable to find a way of

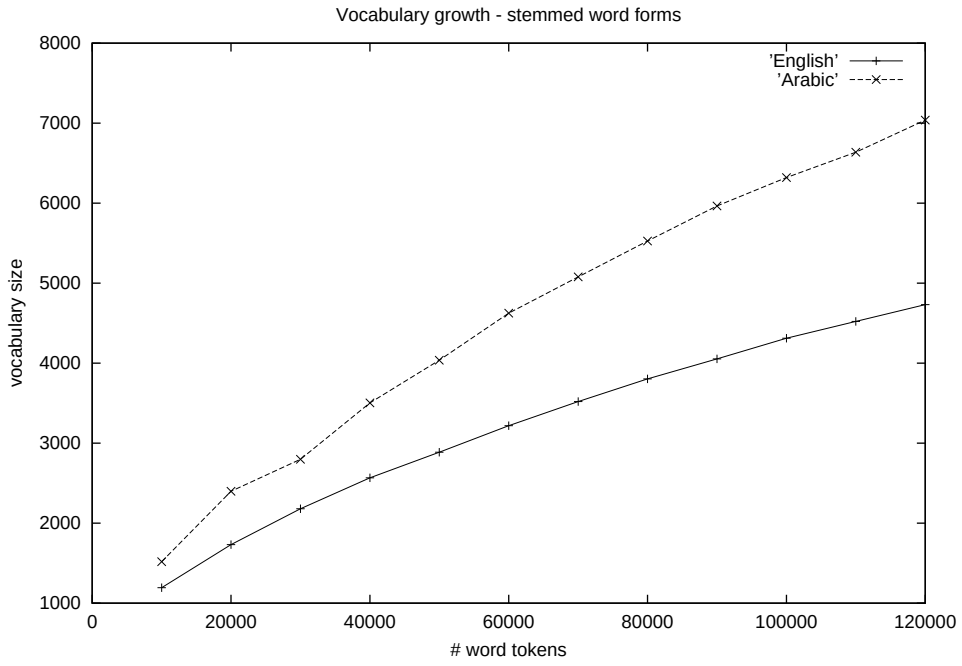

Figure 6: Vocabulary growth for stemmed words in English and ECA

.

decomposing word forms into their morphological components and to build a more robust language model based on probabilities involving individual morphological components.

4.2 Previous Work

The idea of morphologically-based language models is not new. Previous work includes e.g. [30, 37, 17, 53, 33]. In [30], language models for German were constructed by decomposing full word forms into individual morphemes by either expert morphological knowledge, or in a way referred to as “not strictly linguistic”, which was not further specified. The resulting morphs were used as modeling units both in the acoustic components and in the language modeling component. Reductions in perplexity were observed, but also an increase in word error rate. Another strategy was to keep full word forms in the acoustic component (since morphs tend to be short and therefore acoustically confusable) and to map these to their root forms for the purpose of language modeling. This approach was implemented within a lattice-rescoring framework where lattices were first generated using full word forms and the words in the lattices were converted to roots for the purpose of rescoring with a root-based language model. This yielded an improvement in word error rate of 0.7% absolute, for a baseline of 35.3%.

In [17] a morph-based recognizer for Korean was investigated. Elementary syllable units were combined

into units larger than syllables but smaller than words. These therefore corresponded not to linguistically defined morphs but to automatically derived units of approximately the same average size as morphemes. Compared to a fully word-based recognizer, the OOV rate was reduced from 43.% to less than 1% and the lattice word accuracy was increased by 23% relative. However, there was no improvement in actual word error rate measured on the first-best hypotheses.

In [37] a recognizer for Turkish was built on automatically decomposing words into roughly morpheme-sized "chunks". Chunks were then clustered according to their position in the word, and a 4-gram model over these chunks was used as a language model. Here, too, chunks were used in both the acoustic component and the language modeling component. Although the OOV rate was decreased by 50% relative, the WER increased by 1.6% to 4.9% absolute.

Whittaker [53] developed a so-called particle model for morphologically rich languages and tested it on Russian and English. The particle model is formulated as

$$P(w_n|h) = \frac{1}{Z(h)} P(u^{w_n}_{L(w_n)}|u^{w_n}_{L(w_n)-1}) P(u^{w_n}_{L(w_n)-1}|u_{L(w_n)-2}), ..., P(u^{w_{n-1}}|u^{w_{n-1}}_{L(w_{n-1})}) \quad (2)$$

This model assumed that a word w is decomposed by a unique decomposition function L into $L(w)$ particles $u_1, ..., u_{L(w)}$. The particle model computes the probability of a particle given its history consisting of all particles up to the last particle in the preceding word. The particle probabilities theoretically need to be normalized by $\frac{1}{Z_h}$ in order to produce a valid probability distribution; however, Whittaker uses the unnormalized probabilities in practice for computational efficiency. He reports perplexity results based on these probabilities (which, as he claims, represent upper bounds on the true perplexities resulting from normalized probabilities. Whittaker compared three different ways of decomposing words. First, perfect linguistic decomposition into morphemes was used. Second, a greedy search over possible units of a fixed length L was performed and those particles maximizing the likelihood of the data were retained. Third, a particle-growing technique was tested where particles were initialized to all single character units and were successively extended by adding surrounding characters such that the resulting units minimize perplexity. In all cases, n-gram models up to the order 6 were used. Significant reductions in perplexity were obtained but no ASR results were reported.

In Ircing et al. [13] a morphologically-based language model for Czech was used in a large-vocabulary continuous speech recognition system. Words were decomposed into stems and affixes and a morpheme bigram model was used. There were no WER improvements compared to the word-based baseline system.

Some rudimentary morphological processing was applied to Arabic in the BBN 1996 evaluation system [24]. The definite article in ECA, *il*, which always attaches to the following noun, was separated from its successor and treated as a separate word, both for acoustic modeling and for language modeling. This reduced the vocabulary size by 7% (relative), the perplexity by 25% (relative), and the word error rate by 1% (absolute), for a baseline word error rate of 73%.

4.3 Experimental Framework

An observation common to all previous studies on morphology-based ASR is that possible decreases in vocabulary size and perplexity are offset by the increased acoustic confusability of morphemes in those cases where they are also used as acoustic units. Morphemes tend to be rather short and can therefore not be decoded as reliably as longer units, which is why the use of morpheme-sized modeling units often leads to increases in WER. For this reason we decided not to use morphs in the acoustic modeling component but only in the language modeling component. This can be achieved within a rescoring framework: lattices or N-best lists are first generated using a lexicon consisting of full word forms and are subsequently rescored using a morph-based language model.

Our baseline system generated N-best lists containing up to 100 hypotheses per utterance. Along with the word hypotheses, acoustic and language model scores were generated. The final score used for reranking the hypotheses was a weighted combination of the acoustic score, the language model score, the number of phones, the number of words, and the number of silences. For score combination we used the framework of discriminative model combination (DMC) [6]. This approach aims at an optimal (with respect to WER) integration of independent sources of information in a log-linear model that computes the likelihood for a given hypothesis. The parameters to be trained discriminatively are the weights of the log-linear combination, and are optimized to minimize the word error rate on a held out set. In previous work [7, 52, 31] a log-linear model combining multilingual acoustic models or audio and visual models was used to rescore N-best lists in order to choose the hypothesis with the highest likelihood score. The parameters (weights) were trained on the N-best lists of a held out data set, so that the empirical word error count of the highest likelihood hypothesis was minimized. For the optimization in this work we used a simplex downhill method known as amoeba search [42], which is available as part of the SRILM toolkit.

We used two different baseline language models in our work. The first LM (BBN-LM) was part of the BBN baseline recognizer. The second LM (WS02-LM) was built prior to the workshop using the SRILM toolkit. The input text for WS02-LM was preprocessed as follows:

- all foreign words were mapped to one class
- all fillers were mapped to one class
- noise events were removed

The language model itself was a trigram with modified Kneser-Ney smoothing. The difference to the BBN language model lies mainly in the treatment of fillers etc. , which were kept distinct in the BBN language model, and different smoothing methods used.

The baseline trigram perplexity of the WS02 language model with respect to the development set was 172.4.

Affix	Function	Affix	Function
-i	1st sg poss	-ni	1st sg dir
-li	1st sg ind	-na	1st pl poss/dir/ind
-ik	2nd sg fem poss	-lik	2nd sg fem dir/ind
-ak	2nd sg masc poss	-lak	2nd sg masc dir/ind
-ha	3rd sg fem poss	-l(a)ha	3rd sg fem dir/ind
-hu	3rd sg masc poss	-lhu	3rd sg masc dir/ind
-ku(m)	2nd pl poss/dir/ind	-hum	3rd pl poss/dir/ind
Il-	definite article	bi-	preposition *in*
fi-	preposition *in*	li-	preposition *in order to, for*
fa-	conjunction *so, and*	ka-	preposition *like*
ma-	negation particle	Ha-	future tense particle

Table 23: Affixes used for word decomposition. Sg = singular, pl = plural, poss = possessive, dir = direct object ind = indirect object.

4.4 Particle model

Since an extremely simple morphological decomposition (i.e. separation of the definite article *il* from the following noun) showed promising results in earlier work on CallHome [24], we assumed that a higher degree of decomposition would be beneficial. Thus, in addition to separating the definite article from its successor, several other particles and affixes were separated from their word stems. We used the information in the LDC CallHome lexicon combined with knowledge of Egyptian Arabic morphology (see e.g. [1]) to identify recurrent morphemes. Those affixes which were used for decomposition are the possessive and object pronoun suffixes as well as the negation and future markers and prepositional particles (see Table 23). Some examples of decomposed sequences are shown below:

original: fa+ qalUli bass bAt il+ Hagat
decomposed: fa+qalUli bass bAt il+Hagat

original: Hayibqa fi nAs kitIr hinAk mayicrafU$ rUsi wi macAhum nAs yicrafu

decomposed: Ha+ yibqa fi nAs kitIr hinAk ma+ yicrafU$ rUsi wi macA +hum nAs yicraf +u
original: bi+il+raGm in il+naharda il+xamIs il+qunSul il+muqrif da cAyiz yiwaddI+ni il+$uGl
decomposed: bi+ il+ raGm in il+ naharda il+ xamIs il+ qunSul il+ muqrif da cAyiz yiwaddI +ni il+ $uGl

Word and particle representations can be converted to each other unambiguously. A language model trained on this representation models statistical regularties governing sequences of stems and affixes, as opposed to

sequences of words. The resulting language model is similar to Whittaker's model described above, which is why we retained the name *particle* model for this approach. Both 3- and 4-grams were trained on this representation. Perplexity was measured as shown in Equation 3

$$PP(w_1, ..., w_N) = 2^{-\frac{1}{N}\sum_{i=1}^{M} log(P(part_i|part_{i-1},part_{i-2}))} \quad (3)$$

where N is the number of words and M is the number of particles into which the word stream has been decomposed. Note that the log probability is accumulated over particles but the normalization factor is still the number of words, not the number of particles. This is done in order to compensate for the effect that perplexity is automatically lower for a text containing more individual units, since the sum of log probabilities is divided by a larger denominator. Without using the same normalization factor, perplexities would not be comparable. In each case, trigram models were built using modified Kneser-Ney smoothing with interpolation of bi- and trigram probabilities. The trigram perplexity obtained on the CH development set was 172.4 for the word model and 220.1 for the particle model. The increase in perplexity we observed is different from the results reported in [24], where separating *il-* from the noun resulted in a relative decrease in perplexity of 25%. However, we believe that the latter result is due to an inappropriate way of normalizing perplexity, viz. by division by the number of morphs rather than the number of words. When using this normalization we observed the same reduction in perplexity.[3]

Although the particle model achieves a higher perplexity on the development set, it may still yield significantly different information than a word-based language model. For this reason we combined the particle model scores on the N-best lists with the word-based baseline language model scores in the fashion described above. Table 24 shows that a small reduction in word error rate (0.7%) was achieved.

Model	WER
word-based model	55.1%
particle model	54.4%

Table 24: Word error rates for baseline word-based and particle based language models.

4.5 Single Stream Models

Instead of breaking words up into sequences of morphs or particles, words can also be conceived of as bundles of concurrent morphological specifications or other features; an idea which is central to the factored language modeling approach described in detail in Section 5. For example, the word *calls* could be analyzed as consisting of a stem *CALL* plus the morphological feature [noun plural] or [verb 3rd person singular], depending on the interpretation. The structure of Arabic words is richer than that of English words (cf. Section

[3]Similar observations were recently made in a study of morphological language modeling for English [33], where a morphologically decomposed model also led to higher perplexity compared to a word-based model.

1.1), suggesting a decomposition into stems, morphological affixes, roots, and patterns. A sequence of words thus is turned into a sequence of word-feature vectors; a sequence of individual vector components defines a feature stream. These individual streams can be used as alternative information sources in the log-linear combination framework described above. Either each stream can be used separately, or the variables in one stream can be predicted taking into account cross-stream dependencies. We can also use the word sequence itself as a parallel stream and combine it with the rest.

4.5.1 Word decomposition

In our experiments we tried three different ways of decomposing words into bundles of features:

- decomposition based on (mostly) expert morphological knowledge
- decomposition based on automatically learned morphology
- decomposition based on data-driven word clustering

Expert morphological knowledge
Morphological information about every word in the ECA corpus is provided in the LDC ECA lexicon (see Section 2.1, repeated here for convenience):

rom.	script	pron.	stress	morph.	train freq	dev freq	eval freq.	source
$Axix	صَاخّ	$%xix	10	$Axix:pple-act+masc-sg	0	0	0	B

This information consists of the stem (5th field before the colon) and the morphological class (5th field after the colon), which is defined by grammatical features such as word type, number, gender, etc. It can also be viewed as a very fine-grained part-of-speech classification. Both the stem and the morphological class are used as word features. The stem is then further decomposed into the root and the pattern. This information is not in the LDC lexicon. For this reason we used the automatic morphological analyzer provided by K. Darwish [18]. The pattern was obtained by subtracting the root provided by the automatic analyzer from the stem. It should be noted that the analyzer was developed for Modern Standard Arabic; it therefore does not provide accurate stem-to-root conversions for all forms, such that both the stem and the root information are errorful.

Automatic Morphology
Linguistic analysis tools require extensive expert knowledge and long development times; they are therefore not available for a wide range of languages and dialects. It would therefore be interesting to investigate the possibility of automatically acquiring morphological word decompositions from data. During WS02 we investigated the possibility of learning the morphological structure of ECA from data; this is described in detail in Section 6.

Data-driven classes

The last method is not actually a decomposition method since it does not make use of knowledge about the internal structure of words. Instead, words are associated with indices of word clusters they were assigned to during an automatic clustering procedure. This is mainly intended as a comparison to the first two methods: since word features like stems and morph classes essentially define broad word classes they should be compared against a data-driven way of defining word classes. We used the clustering methods available in the SRILM toolkit [11] as well as the method by [41].

4.5.2 Experiments

The decomposition described above produces four streams:
$S = s_1, s_2, s_3, ..., s_N$ (stems)
$R = r_1, r_2, r_3, ..., r_N$ (roots)
$P = p_1, p_2, p_3, ..., p_N$ (patterns)
$M = m_1, m_2, m_3, ..., m_N$ (morph classes)
We initially constructed models that predict the words given the stem, root, pattern, morph or word context, where the last is the normal trigram LM. The same method for used for predicting each of the other streams. Then we used DMC to combine these stream models together to provide a language model score for the hypothesis. The weights for these different language model scores were optimized jointly with the standard knowledge sources (acoustic models, insertion penalties) to minimize WER. This approach gives us 25 streams models in the log-linear combination, not all of them equally useful for predicting the correct hypothesis, and, more importantly, not completely independent of each other (since all streams are actually different functions of the same words). Thus, optimizing the weights for a log-linear model including all the streams may include redundancy which is noise for the optimization process (when free-parameters are included). We tried to address this problem by optimizing the combination in steps. We optimized the combination of the factors predicting each stream separately. Then we left out the factors that ended up with weights less than 0.05 since we believed these to be noise weights which do not significantly affect the outcome. In the following table we show the results of different combinations. All results include the acoustic scores provided by BBN for the N-best hypotheses, and the word insertion penalty, phone penalty, and silence model penalty, which are all jointly optimized using the development set data. The baseline language models (the one provided by BBN and the one built using the SRILM toolkit), are included in the optimization process as well (the standard trigram which appears as one of the 25 stream models is the second above mentioned baseline LM). Whenever the sign "+" appears in the table it denotes the described log-linear combination of models. We denote a stream model as $X|Y$ where X is the predicted stream variable and Y is the conditioning stream variable.

We see that the DMC approach for combination of language models leads to an improvement over the baseline system. Combining the two different baseline models gives us an improvement of 0.3% over the best baseline model, and adding the independent stream factors gives us an extra 0.4%. It is interesting to

LM models used	WER (%) results			
	w/ baseline LMs		w/o baseline LMs	
	Dev	Eval96	Dev	Eval96
BBN baseline LM (b1)	56.6	54.8	-	-
SRILM baseline (b2)	56.2	54.7	-	-
b1+b2	56.0	54.4	-	-
S|S + S|R + S|P + S|M + S|W	55.7	54.1	56.7	55.9
R|S + R|R + R|P + R|M + R|W	55.8	54.4	58.4	57.3
P|S + P|R + P|P + P|M + P|W	55.8	54.5	58.6	57.4
M|S + M|R + M|P + M|M + M|W	55.7	54.2	57.6	56.4
W|S + W|R + W|P + W|M + W|W	55.7	54.1	56.0	54.6
11-streams (selected from the above): M|M + M|S + P|M + P|P + P|W + R|M + R|R + R|S + S|M + S|S + W|M	55.5	54.0	-	-
11-streams + SRI100 + E100	55.2	53.8	-	-
M + S + SRI100 + E100 + M|M + S|S + R|R	55.2	53.8	-	-
11-streams + SRI100 + E100 + M + S	55.2	53.7	-	-

Table 25: WERs obtained by combinations of stream models. M = morph class, S = stem, R = root, P = pattern; SRI100 = class-based model using SRILM ngram-class tool with 100 classes, E100 = class-based model using the method in [41] with 100 classes.

observe which of the factors provide the most discriminative information. This is better observed when not including the baseline trigrams in the combination. We see that predicting the stem stream adds the most information, closely followed by the morphological class even though it has no information about the word itself. Patterns and roots do not seem to contribute much in helping to discriminate the correct hypothesis. We actually observed that the different hypotheses in the N-best lists do not exhibit much variability in patterns and roots. It is also interesting that adding the class-based models gives an extra improvement, even though we initially expected that these models are using the same information in a different way. In the best result we observe an extra 0.3% improvement over the best stream combination system adding up to a total of 0.7% absolute improvement over the baseline combination. It is important to note that the size of N-best (100-best) lists used in this task may have been a limiting factor in the optimization process. As the size of parameters grows, the danger of overtraining on the development set used for optimization increases. Typically, a larger number of hypotheses has been used in similar experiments (500-1000) or a separate held-out set has been used to evaluate the performance of the optimized parameters.

5 Factored Language Models and Generalized Backoff

In this section, we describe factored language models and generalized backoff, two novel techniques in statistical language modeling that attempt to improve statistical estimation (i.e., reduce parameter variance) in language models, and that also attempt to better describe the way in which language (and sequences of words) might be produced. This work also led to a set of significant extensions to the SRI language modeling toolkit to support the above concepts. A complete guide to the new toolkit features (which serves as toolkit documentation) is given in Section 5.4. This section provides initial perplexity results on ECA CallHome that were obtained at the workshop. Perhaps most significantly, it was found that by using a factored language model and a generalized backoff procedure, it was possible to obtain a bigram with a lower perplexity than the highly-optimized baseline trigram, something that can have significant implications for first-pass speech recognition decoding.

5.1 Background

In most standard language models, it is the case that a sentence is viewed as a sequence of T words $w_1, w_2, \ldots, w_T$. The goal is to produce a probability model over a sentence in some way or another, which can be denoted by $p(w_1, w_2, \ldots, w_T)$ or more concisely as $p(w_{1:T})$ using a matlab-like notation for ranges of words.

It is most often the case that the chain rule of probability is used to factor this joint distribution thus:

$$p(w_{1:T}) = \prod_t p(w_t|w_{1:t-1}) = \prod_t p(w_t|h_t)$$

where $h_t = w_{1:t-1}$. The key goal, therefore, is to produce the set of models $p(w_t|h_t)$ where w_t is the current word and h_t is the multi-word immediately preceding history leading up to word w_t. In a standard n-gram based model, it is assumed that the immediate past (i.e., the preceding $n-1$ words) is sufficient to capture the entire history up to the current word. In particular, conditional independence assumptions about word strings are made in an n-gram model, where it is assumed that $W_t \perp\!\!\!\perp W_{1:t-n} | W_{t-n+1:t-1}$. In a trigram ($n = 3$) model, for example, the goal is to produce conditional probability representations of the form $p(w_t|w_{t-1}, w_{t-2})$.

Even when n is small (e.g., 2 or 3) the estimation of the conditional probability table (CPT) $p(w_t|w_{t-1}, w_{t-2})$ can be quite difficult. In a standard large-vocabulary continuous speech recognition system there is, for example, a vocabulary size of at least 60,000 words (and 150,000 words would not be unheard of). The number of entries required in the CPT would be $60,000^3 \approx 2 \times 10^{14}$ (about 200,000 gigabytes using the quite conservative estimate of one byte per entry). Such a table would be far to large to obtain low-variance estimates without significantly more training data than is typically available. Even if enough training data was available, the storage of such a large table would be prohibitive.

Fortunately, a number of solutions have been proposed and utilized in the past. In class-based language models, for example, words are bundled into classes in order to improve parameter estimation robustness. Words are then assumed to be conditionally independent of other words given the current word class. Assuming that c_t is the class of word w_t at time t, the class-based model can be represented as follows:

$$\begin{aligned} p(w_t|w_{t-1}) &= \sum_{c_t, c_{t-1}} p(w_t|c_t)p(c_t|c_{t-1}, w_{t-1})p(c_{t-1}|w_{t-1}) \\ &= \sum_{c_t, c_{t-1}} p(w_t|c_t)p(c_t|c_{t-1})p(c_{t-1}|w_{t-1}) \end{aligned}$$

where to get the second equality it is assumed that $C_t \perp\!\!\!\perp W_{t-1}|C_{t-1}$. In order to simplify this further, a number of additional assumptions are made, namely:

1. There is exactly one possible class for each word, or that there is a deterministic mapping (lets say a function $\hat{c}(\cdot)$) from words to classes. This makes it such that $p(c_t|w_t) = \delta_{c_t=\bar{c}(w_t)}$, where $\delta(\cdot)$ is the Dirac delta function. It also makes it such that $p(w_t|c_t) = p(w_t|\bar{c}(w_t))\delta_{c_t=\bar{c}(w_t)}$.

2. A class will contain more than one word. This assumption is in fact what makes class-based language models easier to estimate.

With the assumptions above, the class-based language model becomes.

$$p(w_t|w_{t-1}) = p(w_t|\bar{c}(w_t))p(\bar{c}(w_t)|\bar{c}(w_{t-1}))$$

Other modeling attempts to improve upon the situation given above include particle-based language models [53], maximum-entropy language models [5], mixture of different-ordered models [36], and the backoff procedure [36, 15], the last of which is briefly reviewed in Section 5.3. Many others are surveyed in [36, 15].

5.2 Factored Language Models

In this section, we introduce a novel form of model entitled a *factored language model* or FLM, a notion that was introduced for the purposes of experimentation during the workshop. In a factored language model, a word is seen as a collection or bundle of K (parallel) factors, so that $w_t \equiv \{f_t^1, f_t^2, \ldots, f_t^K\}$. Factors of a given word can be anything, including morphological classes, stems, roots, and any other linguistic features that might correspond to or decompose a word. A factor can even be a word itself, so that the probabilistic language model is over both words and its decompositional factors. While it should be clear that such a decomposition will be useful for languages that are highly inflected (such as Arabic), the factored form can be applied to any language. This is because data-driven word-classes or simple word stems or semantic features that occur in any language can be used as factors. Therefore, the factored representation is quite general and widely applicable.

Once a set of factors for words has been selected, the goal of an FLM is to produce a statistical model over the individual factors, namely:

$$p(f_{1:T}^{1:K})$$

or, using an n-gram-like formalism, the goal is to produce accurate models of the form:

$$p(f_t^{1:K} | f_{t-n+1:t-1}^{1:K})$$

In this form, it can be seen that a FLM opens up the possibility for many modeling options in addition to those permitted by a standard n-gram model over words. For example, the chain rule of probability can be used to represent the above term as follows:

$$\prod_k p(f_t^k | f_t^{1:k-1}, f_{t-n+1:t-1}^{1:K})$$

This represents only one possible chain-rule ordering of the factors. Given the number of possible chain-rule orderings (i.e., the number of factor permutations which is $K!$) and (approximately) the number of possible options for subsets of variables to the right of the conditioning bar (i.e., $\approx 2^{nK}$) we can see that a FLM represents a huge family of statistical language models (approximately $K!2^{nK}$). In the simplest of cases, a FLM includes standard n-grams or standard class-based language models (e.g., choose as one of the factors the word class, ensure that the word variable depends only on that class, and the class depends only on the previous class). A FLM, however, can do much more than this.

Our ultimate goal and the main two research problems in forming an FLM, therefore, are twofold:

1. First, choose an appropriate set of factor definitions. This can be done in a data-driven fashion, or can use as an aide high-level linguistic knowledge about the target language (see Section 4.5.1).

2. Second, find the best statistical model over these factors, one that mitigates the statistical estimation difficulties found in standard language models, and also one that describes the statistical properties of the language in such a way that word predictions are accurate (i.e., true word strings should be highly probable and any other highly-probable word strings should have minimal word error).

The problem of identifying the best statistical model over a given set of factors can be described as an instance of the structure learning problem in graphical models [40, 27, 9]. In a directed graphical model the graph depicts a given probability model. The conditional probability $p(A|B, C)$ is depicted by a directed graph where there is a node in the graph for each random variable, and arrows point from B and C (the parents) to A (the child).

We here give several examples of FLMs and the their corresponding graphs. In both examples, only three factors are used, the word variable at each time W_t, the words morphological class M_t (the "morph" as we call it), and the words stem S_t.

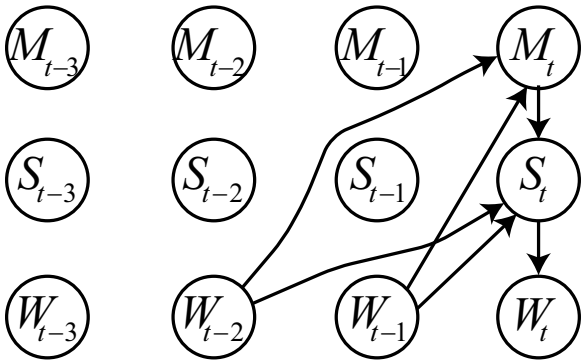

Figure 7: An example of a factored language model seen as a directed graphical model over words W_t, morphological factors W_t, and stems S_t. The figure shows the dependencies for the variables only at time t for simplicity.

The first example corresponds to the following model:

$$p(w_t|s_t, m_t)p(s_t|m_t, w_{t-1}, w_{t-2})p(m_t, w_{t-1}, w_{t-2})$$

and is shown in Figure 7. This is an example of a factored class-based language model in that the word class variable is represented in factored form as a stem S_t and morphological class M_t. In the ideal case, the word is (almost) entirely determined by the stem and the morph, so that the model $p(w_t|s_t, m_t)$ itself would have very low perplexity (of not much more than unity). It is in the models $p(s_t|m_t, w_{t-1}, w_{t-2})$ and $p(m_t, w_{t-1}, w_{t-2})$ that prediction (and uncertainty) becomes apparent. In forming such a model, the goal would be for the product of the perplexities of the three respective models to be less than the perplexity of the word-only trigram. This might be achievable if it were the case that the estimation of $p(s_t|m_t, w_{t-1}, w_{t-2})$ and $p(m_t, w_{t-1}, w_{t-2})$ is inherently easier than that of $p(w_t|w_{t-1}, w_{t-2})$. It might be because the total number of stems and morphs will each be much less than the total number of words. In any event, one can see that there are many possiblities for this model. For example, it might be useful to add M_{t-1} and S_{t-1} as additional parents of M_t which then might reduce the perplexity of the M_t model. In theory, the addition of parents can only reduce entropy which in term should only reduce perplexity. On the other hand, it might not do this because the difficulty of the estimation problem (i.e., the dimensionality of the corresponding CPT) increases with the addition of parents. The goal of finding an appropriate FLM is that of finding the best compromise between predictability (which reduces model entropy and perplexity) and estimation error. This is a standard problem in statistical inference where bias and variance can be traded for each other.

The next example of an FLM is the following:

$$p(w_t|w_{t-1}, w_{t-2}, s_{t-1}, s_{t-2}, m_{t-1}, m_{t-2}).$$

This model is similar to the standard word-based trigram but where the two previous stems and morphs have been added as parents. It is typical that both stems and morphs can be predicted deterministically from the

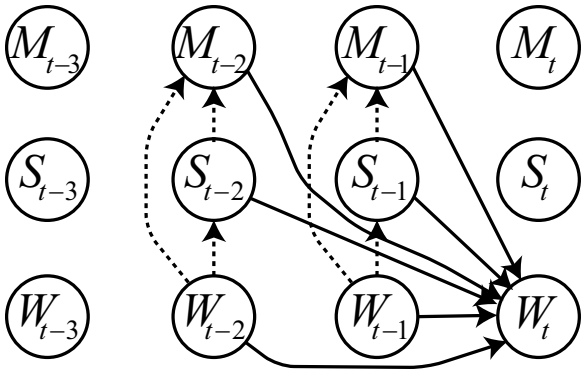

Figure 8: Another example of a factored language model seen as a directed graphical model over words W_t, morphological factors W_t, and stems S_t. The figure shows the child variable W_t and all of its parents. It also shows that the stem and morph might be (almost) deterministic functions of their parents (in this case the corresponding word) as dashed lines.

word. I.e., given a word w_t the stem s_t and the morphological class m_t can be determined with certainty[4] This model is shown in Figure 8. The word variable W_t is shown as the child with parents W_{t-1}, W_{t-2}, S_{t-1}, S_{t-2}, M_{t-1}, M_{t-2}. Also the deterministic variables are shown with their parents as dashed lines.

There are two interesting issues that become apparent when examining this model. First, the previous morph class and stem variables are (often) deterministic functions of their parents (previous words). In such a case, therefore, it should not be useful to use them as additional parents. This is because no additional information can be gleaned from a new parent if it is only a deterministic function of the set of parents that already exist. For example, given the model $P(A|B,C)$ and where D is a deterministic function of C, then the models $P(A|B,C)$ and $P(A|B,C,D)$ would have exactly the same entropy.

Even in this case, however, the model above (Figure 8) might have a benefit over a standard word trigram. The reason is that there might be word trigrams that do not occur in training-data, but the word preceded by the corresponding stems and/or morphs **do** occur in the same training data set. By using the model in the appropriate way, it might be detected when a preceding word history does not occur, but the corresponding stem and/or morph history does occur. This, therefore, would have a benefit in a backoff algorithm over just simply backing-off down to the bigram or unigram (see Section 5.3.1 for a brief overview of the standard backoff algorithm).

The second interesting issue that arises is the following. When considering just the set of parents in a word trigram W_{t-1} and W_{t-2}, and when considering a backoff algorithm, it can reasonably be argued that the best parent to drop first when backing off is the most distant parent W_{t-2} from the current word W_t. This rests on the assumption that the more distant parent has less predictive power (i.e., ability to reduce

[4]In some cases, they might not be predictable with certainty, but even then there would be only a small number of different possibilities.

perplexity) than does the less distant parent.

On the other hand, when considering a set of parents of a word such as W_{t-1}, W_{t-2}, S_{t-1}, S_{t-2}, M_{t-1}, M_{t-2}, as shown in Figure 8, it is not immediately obvious nor is there any reasonable a-priori assumption, which can be used to determine which parent should be dropped when backing off in all cases at all times. In so

me circumstances, it might be best to first drop W_{t-2}, but in other cases it might be best to drop W_{t-1}, or perhaps some other parent. In other words, what backoff order should be chosen in a general model?

The next section addresses both of these issues by introducing the notion we call generalized backoff.

5.3 Generalized Backoff

In this section, the we describe the generalized backoff procedure that was initially developed with factored language models in mind, but turns out to be generally applicable even when using typical word-only based language models or even when forming arbitrary smoothed conditional probability tables over arbitrary random variables. We will first briefly review standard backoff in language models, and then present the generalized backoff algorithms.

5.3.1 Background on Backoff and Smoothing

A common methodology in statistical language model is the idea of *backoff*. Backoff is used whenever there is insufficient data to fully estimate a high-order conditional probability table — instead of attempting to estimate the entire table, only a portion of the table is estimated, and the remainder is constructed from a lower-order model (by dropping one of the variables on the right of the conditioning bar, e.g., going from a trigram $p(w_t|w_{t-1}, w_{t-2})$ down to a bigram $p(w_t|w_{t-1})$). If the higher-order model was estimated in a maximum-likelihood setting, then the probabilities would simply be equal to the ratio counts of each word string, and there would be no left-over probability for the lower order model. Therefore, probability mass is essentially "stolen" away from the higher order model and is distributed to the lower-order model in such a way that the conditional probability table is still valid (i.e., sums to unity). The procedure is then applied recursively on down to a uniform distribution over the words.

The most general way of presenting the backoff strategy is as follows (we present backoff from a trigram to a bigram for illustrative purposes, but the same general idea applies between any n-gram and $(n-1)$-gram). We are given a goal distribution $p(w_t|w_{t-1}, w_{t-2})$. The maximum-likelihood estimates of this distribution is:

$$p_{ML}(w_t|w_{t-1}, w_{t-2}) = \frac{N(w_t, w_{t-1}, w_{t-2})}{N(w_{t-1}, w_{t-2})}$$

where $N(w_t, w_{t-1}, w_{t-2})$ is equal to the number of times (or the count) that the the word string w_{t-2}, w_{t-1}, w_t occurred in training data. We can see from this definition that for all values of w_{t-1}, w_{t-2}, we have that $\sum_w p_{ML}(w|w_{t-1}, w_{t-2}) = 1$ leading to a valid probability mass function.

In a backoff language model, the higher order distribution is used only when the count of a particular string of words exceeds some specified threshold. In particular, the trigram is used only when $N(w_t, w_{t-1}, w_{t-2}) > \tau_3$ where τ_3 is some given threshold, often set to 0 but it can be higher depending on the amount of training data that is available. The procedure to produce a backoff model $p_{BO}(w_t|w_{t-1}, w_{t-2})$ can most simply be described as follows:

$$p_{BO}(w_t|w_{t-1}, w_{t-2}) = \begin{cases} d_{N(w_t,w_{t-1},w_{t-2})} p_{ML}(w_t|w_{t-1}, w_{t-2}) & \text{if } N(w_t, w_{t-1}, w_{t-2}) > \tau_3 \\ \alpha(w_{t-1}, w_{t-2}) p_{BO}(w_t|w_{t-1}) & \text{otherwise} \end{cases}$$

As can be seen, the maximum-likelihood trigram distribution is used only if the trigram count is high-enough, and if it is used it is only used after the application of a discount $d_{N(w_t,w_{t-1},w_{t-2})}$, a number that is generally between 0 and 1 in value.

The discount is a factor that steals probability away from the trigram so that it can be given to the bigram distribution. The discount is also what determines the *smoothing* methodology that is used, since it determines how much of the higher-order maximum-likelihood model's probability mass is smoothed with lower-order models. Note that in this form of backoff, the discount $d_{N(w_t,w_{t-1},w_{t-2})}$ can be used to represent many possible smoothing methods, including Good-Turing, absolute discounting, constant discounting, natural discounting, modified Kneser-Ney smoothing, and many others [36, 14] (in particular, see Table 2 in [15] which gives the from d for each smoothing method).

The quantity $\alpha(w_{t-1}, w_{t-2})$ is used to make sure that the entire distribution still sums to unity. Starting with the constraint $\sum_{w_t} p_{BO}(w_t|w_{t-1}, w_{t-2}) = 1$, it is simple to obtain the following derivation:

$$\begin{aligned} \alpha(w_{t-1}, w_{t-2}) &= \frac{1 - \sum_{w:N(w,w_{t-1},w_{t-2})>\tau_3} d_{N(w,w_{t-1},w_{t-2})} p_{ML}(w|w_{t-1}, w_{t-2})}{\sum_{w:N(w,w_{t-1},w_{t-2})<=\tau_3} p_{BO}(w|w_{t-1})} \\ &= \frac{1 - \sum_{w:N(w,w_{t-1},w_{t-2})>\tau_3} d_{N(w,w_{t-1},w_{t-2})} p_{ML}(w|w_{t-1}, w_{t-2})}{1 - \sum_{w:N(w,w_{t-1},w_{t-2})>\tau_3} p_{BO}(w|w_{t-1})} \end{aligned}$$

While mathematically equivalent, the second form is preferred for actual computations because there are many fewer trigrams that have a count that exceed the threshold than there are trigrams that have a count that do not exceed the threshold (as implied by the first form).

While there are many additional details regarding backoff which we do not describe in this report (see [15]), the crucial issue for us here is the following: In a typical backoff procedure, we backoff from the distribution $p_{BO}(w_t|w_{t-1}, w_{t-2})$ to the distribution $p_{BO}(w_t|w_{t-1})$ and in so doing, we drop the most distant random variable W_{t-2} from the set on the right of the conditioning bar. Using graphical models terminology, we say that we drop the most distant parent W_{t-2} of the child variable W_t. The procedure, therefore, can be described as a graph, something we entitle a *backoff graph*, as shown in Figure 9. The graph shows the path, in the case of a 4-gram, from the case where all three parents are used down to the unigram where no parents are used. At each step along the procedure, only the parent most distant in time from the child w_t is removed from the statistical model.

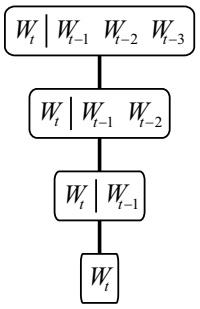

Figure 9: An example of the *backoff path* taken by a 4-gram language model over words. The graph shows that first the 4-gram language model is attempted ($p(w_t|w_{t-1}, w_{t-2}, w_{t-3})$), and is used as long as the string $w_{t-3}, w_{t-2}, w_{t-1}, w_t$ occurred enough times in the language-model training data. If this is not the case, the model "backs off" to a trigram model $p(w_t|w_{t-1}, w_{t-2})$ which is use only if the string w_{t-2}, w_{t-1}, w_t occurs sufficiently often in training data. This process is recursively applied until the unigram language model $p(w_t)$ is used. If there is a word found in test data not found in training data, then even $p(w_t)$ can't be used, and the model can be further backed off to the uniform distribution.

5.3.2 Backoff Graphs and Backoff Paths

As described above, with a factored language model the distributions that are needed are of the form $p(f_t^i | f_{t_1}^{j_1}, f_{t_2}^{j_2}, \ldots, f_{t_N}^{j_N})$ where the j_k and t_k values for $k = 1 \ldots N$ can be in the most general case be arbitrary. For notational simplicity and to make the point clearer, let us re-write this model in the following form:

$$p(f | f_1, f_2, \ldots, f_N)$$

which is the general form of a conditional probability table (CPT) over a set of $N + 1$ random variables, with child variable F and N parent variables F_1 through F_N. Note that f is a possible value of the variable F, and $f_{1:N}$ is a possible vector value of the set of parents $F_{1:N}$.

As mentioned above, in a backoff procedure where the random variables in question are words, it seems reasonable to assume that we should drop the most temporally distant word variable first, since the most distant variable is likely to have the least amount of mutual information about the child given the remaining parents. Using the least distant words has the best chance of being able to predict with high precision the next word.

When applying backoff to a general CPT with random variables that are not words (as one might want to do when using a FLM), it is not immediately obvious which variables should be dropped first when going from a higher-order model to a lower-order model. One possibility might be to choose some arbitrary order, and backoff according to that order, but that seems sub-optimal.

Under the assumption that we always drop one parent at a time[5], we can see that there are quite a large

[5]It is possible to drop more than one parent during a backoff step (see Section 5.4.3 for an example). It is also possible to add

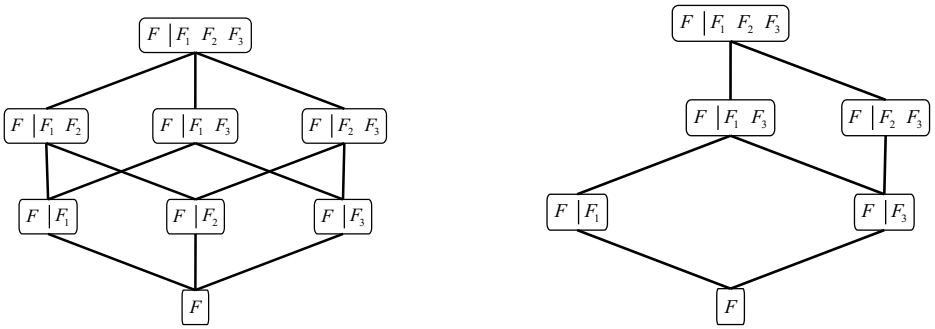

Figure 10: **Left:** An example of a general *backoff graph* for a CPT $p(F|F_1, F_2, F_3)$ showing all possible backoff paths in moving from the model $p(F|F_1, F_2, F_3)$ down to $p(F)$. **Right:** An example of a backoff graph where only a subset of the paths from the top to bottom are allowed. In this graph, the model $p(F|F_1, F_2, F_3)$ is allowed only to backoff to $p(F|F_1, F_3)$ or $p(F|F_2, F_3)$. This means that only the parents F_1 or F_2 may be dropped, but not parent F_3 since all backoff models must have parent F_3. Similarly, the model $p(F|F_2, F_3)$ may backoff only to $p(F|F_3)$, so only the parent F_2 may be dropped in this case. This notion of specifying a subset of the parents that may be dropped to determine a constrained set of backoff paths is used in the SRILM language-model toolkit [50] extensions, described in Section 5.4.

number of possible orders. We will refer to this as a *backoff path*, since it can be viewed as a path in a graph where each node corresponds to a particular statistical model. When all paths are depicted, we get what we define as a *backoff graph* as shown on the left in Figure 10 and Figure 11, the latter of which shows more explicitly the fact that each node in the backoff graph corresponds to a particular statistical model. When training data for that statistical model is not sufficient (below a threshold) in training data, then there is an option as to which next node in the backoff graph can be used. In the example, the node $p(F|F_1, F_2, F_3)$ has three options, those corresponding to dropping any one of the three parents. The nodes $p(F|F_1, F_2)$, $p(F|F_1, F_3)$, and $p(F|F_2, F_3)$ each have two options, and the nodes $p(F|F_1)$, $p(F|F_2)$, and $p(F|F_3)$ each have only one option.

Given the above, we see that there are a number of possible options for choosing a backoff path, and these include:

1. Choose a fixed path from top to bottom based on what seems reasonable (as shown in Figure 12). An example might be to always drop the most temporally distant parent when moving down levels in the backoff graph (as is done with a typical word-based n-gram model). One might also choose the particular path based on other linguistic knowledge about the given domain. Stems, for example, might be dropped before morphological classes because it is believed that the morph classes possess more information about the word the stems. It is also possible to choose the backoff path in a data-driven manner. The parent to be dropped first might be the one which is found to possess the least

parents at a backoff step. This latter case is not further considered in this report.

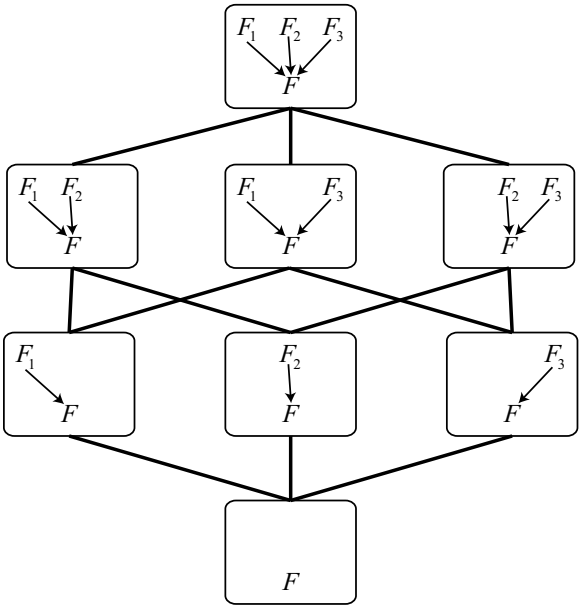

Figure 11: An example of a general *backoff graph* for a CPT $p(F|F_1, F_2, F_3)$ showing all possible backoff paths in moving from the model $p(F|F_1, F_2, F_3)$ down to $p(F)$. In this example, the individual nodes of the backoff graph show the directed graphical model for that node. If an instantiation of a child variable with its parents exists in a language model training corpus above a given number of times, then there is a language model "hit" for the corresponding node in the backoff graph, and that particular model provides a probability. If not, then the next node (or nodes) below are then attempted.

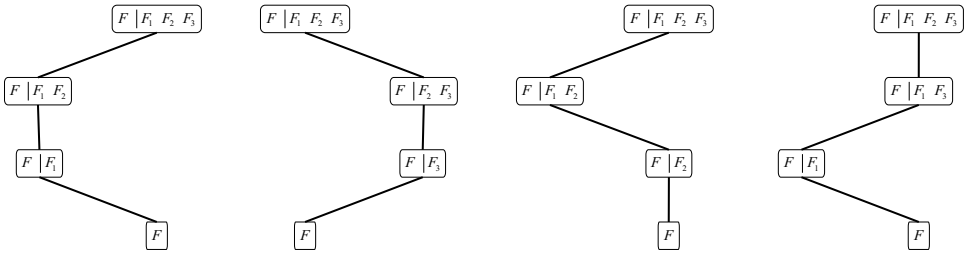

Figure 12: Examples of fixed-backoff-path backoff graphs. Each backoff graph shows a fixed path from the top-level (where all parents are present) to bottom level (where no parents are present). Also, each example shows a separate language model (or more generally a CPT) where the path is fixed at all times. If $F = W_t$, $F_1 = W_{t-1}$, $F_2 = W_{t-2}$, $F_3 = W_{t-3}$, then the left example corresponds to standard backoff in a 4-gram language model — using these definitions, however, shows that when the factors are words there are many other possible backoff schemes, most of which correspond to not dropping first the most temporally distant word.

information (in an information-theoretic sense) about the child variable. Alternatively, it might be possible to choose the parent to drop first based on the tradeoff between statistical predictability and statistical estimation. For example, we might choose to drop a parent which does not raise the entropy the least if the child model is particularly easy to estimate given the current training data (i.e., the child model will have low statistical variance).

2. Generalized all-child backoff. In this case, multiple child nodes in the backoff graph are chosen at "run-time." This is described in the next section.

3. Generalized constrained-child backoff. In this case, any of a subset of the child nodes in the backoff graph are chosen at "run-time." This is also described in the next section.

5.3.3 Generalized Backoff

During the workshop, a novel form of backoff was developed and implemented in order to produce backoff models for FLMs and any other CPT. We call this methodology *generalized backoff*.

Rather than choosing a given fixed path when moving from higher-to-lower level in a backoff graph as shown in Figure 12, it is possible to choose multiple different paths dynamically at run time. In general, there are two options here:

1. In the first case, only one path is chosen at a time at run-time, but depending on the particular sequence of words or factors that are in need of a probability, the backoff path will change. In other words, for a given set of factor instances $f^a|f_1^a, f_2^a, f_3^a$ one particular backoff path might be used, but for $f^b|f_1^b, f_2^b, f_3^b$ a different path will be used.

2. It is also possible for *multiple* backoff paths to be used simultaneously at runtime. This means that for a given $f^a|f_1^a, f_2^a, f_3^a$, multiple paths through the backoff graph will be used to produce a probability. Again, the set of multiple paths that are used might change depending on the particular factor sequence.

This approach can only improve over choosing only a fixed backoff path, since the path(s) that are used can be chosen to best suit the given factor instances $f|f_1, f_2, f_3$.

The procedure we developed is a generalization of standard backoff and probabilities are formed in our technique using the following equation:

$$p_{GBO}(f|f_1, f_2, f_3) = \begin{cases} d_{N(f,f_1,f_2,f_3)} p_{ML}(f|f_1, f_2, f_3) & \text{if } N(f, f_1, f_2, f_3) > \tau_4 \\ \alpha(f_1, f_2, f_3) g(f, f_1, f_2, f_3) & \text{otherwise} \end{cases}$$

where

$$p_{ML}(f|f_1, f_2, f_3) = \frac{N(f, f_1, f_2, f_3)}{N(f_1, f_2, f_3)}$$

is the maximum likelihood distribution (equal to the ratio of counts as seen in training data), and where τ_4 is a user-specified threshold used to determine when a "hit" occurs at the current level, or whether to backoff to the next level. The function $g(f, f_1, f_2, f_3)$ is the backoff distribution and can be any *non-negative* function of f, f_1, f_2, f_3 (we discuss g further below).

The function $\alpha(f_1, f_2, f_3)$ is produced to ensure that the entire distribution is valid (e.g., non-negative and integrates to unity), and is can be simply derived in way similar to the standard equation:

$$\alpha(f_1, f_2, f_3) = \frac{1 - \sum_{f:N(f,f_1,f_2,f_3)>\tau_4} d_{N(f,f_1,f_2,f_3)} p_{ML}(f|f_1, f_2, f_3)}{\sum_{f:N(f,f_1,f_2,f_3)<=\tau_4} g(f, f_1, f_2, f_3)}$$

It should be noted, however, that the denominator in this equation can not be simplified to use the sum form $\sum_{f:N(f,f_1,f_2,f_3)>\tau_4}$. This is because there is no guarantee that $g(f, f_1, f_2, f_3)$ will sum to unity (it is no-longer a probability distribution, and is only guaranteed to be positive). This inability could lead to significant computational increases when using such language models. As will be seen in the sections below, however, these computational costs are not prohibitive on today's fast computers — many of the experiments were run just on a portable laptop computer (Pentium-3 at 1GHz with 500Mb RAM) during the workshop. Moreover, the SRILM toolkit extensions (to be described in Section 5.4) attempted to use the $\sum_{f:N(f,f_1,f_2,f_3)>\tau_4}$ form for certain g functions when possible.

The choice of functions $g(f, f_1, f_2, f_3)$ is where the different backoff strategies are determined. For example, in a typical backoff procedure, we would chose $g(f, f_1, f_2, f_3) = p_{BO}(f|f_1, f_2)$. If the factors corresponded to words (using the mapping from factors to words given in the caption in Figure 12, then this corresponds to dropping the most temporally distant word as is typical.

In general, however, $g(f, f_1, f_2, f_3)$ can be any non-negative function, and essentially the same backoff weight algorithm for computing $\alpha(f_1, f_2, f_3)$ can be used. If $g(f, f_1, f_2, f_3)$ were allowed to be negative, then the standard algorithm for producing $\alpha(f_1, f_2, f_3)$ no longer works since the determination of when $g(f, f_1, f_2, f_3)$ is negative for a given f_1, f_2, f_3 might depend on f, something $\alpha(f_1, f_2, f_3)$ may not use. In any event, as we will see the generalization of standard backoff is quite broad even under the assumption of non-negative $g(f, f_1, f_2, f_3)$ functions.

There are a number of possible $g(f, f_1, f_2, f_3)$ that were considered (and as will be seen in Section 5.4 implemented) during the workshop. These include:

- **Fixed backoff path, standard backoff:** In this case the backoff path is fixed for all times as described in Section 5.3.2. Notationally, this may be described as follows:

$$g(f, f_1, f_2, f_3) = \begin{cases} p_{BO}(f|f1, f2) & \text{or} \\ p_{BO}(f|f1, f3) & \text{or} \\ p_{BO}(f|f2, f3) & \end{cases}$$

  where the "or" means "exclusive or" (so one and only one choice is taken) and where the choice is decided by the user and is fixed for all time. This case therefore includes standard backoff (as in a word n-gram) but also allows for arbitrary fixed (but single) backoff-paths to be specified (see Figure 12).

- **Max Counts:** In this case, the backoff probability model is chosen based on the number of counts of the current "gram" (i.e., random variable assignments). Specifically,

$$g(f, f_1, f_2, f_3) = p_{GBO}(f|f_{\ell_1}, f_{\ell_2})$$

where

$$(\ell_1, \ell_2) = \underset{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}}{\text{argmax}} N(f, f_{m_1}, f_{m_2})$$

Because this is slightly notationally cumbersome, we here describe this approach in the case when there are only two parents, thus giving the definition for $g(f, f_1, f_2)$. In this case, the above equations can be written as:

$$g(f, f_1, f_2) = p_{GBO}(f|f_\ell)$$

where

$$\ell = \underset{j\in\{1,2\}}{\text{argmax}} N(f, f_j)$$

In other words, for each set of factor values, the backoff model chosen to be the one that corresponds to the factor values that have maximum counts. This means that a different backoff path will be used for each instance, and one will always backoff down the path of maximum counts. This approach tends to prefer backoff models that have better (non-relative) statistical estimation properties, but it ignores the statistical predictability of the resulting backoff models.

- **Max Normalized Counts:** In this case, rather than the absolute counts, the normalized counts are used to determine the backoff model. Thus,

$$g(f, f_1, f_2, f_3) = p_{GBO}(f|f_{\ell_1}, f_{\ell_2})$$

where

$$(\ell_1, \ell_2) = \underset{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}}{\text{argmax}} \frac{N(f, f_{m_1}, f_{m_2})}{N(f_{m_1}, f_{m_2})} = \underset{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}}{\text{argmax}} p_{ML}(f|f_{m_1}, f_{m_2})$$

This approach therefore chooses the backoff model according to the one that has the highest probability score using the maximum-likelihood distribution. This approach therefore favors models that yield high-predictability possibly at the expense of statistical estimation quality.

- **Max Num-Words Normalized Counts:** In this case, the counts are normalized by the number of possible factor values for a given set of parent values, as computed using the training set.

The set of possible factor values for a given parent context f_1, f_2, f_3 can be expressed as the following set:

$$\{f : N(f, f_1, f_2, f_3) > 0\}$$

and the number of possible following words is equal to the cardinality of this set. Using the standard notation to express set cardinality, we obtain the following equations:

$$g(f, f_1, f_2, f_3) = p_{GBO}(f|f_{\ell_1}, f_{\ell_2})$$

where

$$(\ell_1, \ell_2) = \underset{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}}{\text{argmax}} \frac{N(f, f_{m_1}, f_{m_2})}{|\{f : N(f, f_{m_1}, f_{m_2}) > 0\}|}$$

Note that if the factors correspond to words, then the normalization is equal to the number of possible following words for a given history.

- **Max Backoff-Graph Node Probability:** In this approach, we choose the backoff model to be the one that gives the factor and parent context in question the maximum probability. This can be written as:

$$g(f, f_1, f_2, f_3) = p_{GBO}(f|f_{\ell_1}, f_{\ell_2})$$

where

$$(\ell_1, \ell_2) = \underset{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}}{\text{argmax}} p_{GBO}(f|f_{m_1}, f_{m_2})$$

This approach therefore chooses the next backoff graph node to be the one that supplies the highest probability, but where the probability is itself obtained via using a backoff procedure.

- **Max Product-of-Cardinality Normalized Counts:** In this approach, the counts are normalized by the product of the cardinalities of the underlying random variables. In this case, we use the term "cardinality of a random variable" to mean the number of possible values that random variable may take on in the training set. Again using set notation, we will notate the cardinality of a random variable as $|F|$, where the cardinality of a random variable F is taken to be:

$$|F| \triangleq |\{f : N(f) > 0\}|$$

This leads to:

$$g(f, f_1, f_2, f_3) = p_{GBO}(f|f_{\ell_1}, f_{\ell_2})$$

where

$$(\ell_1, \ell_2) = \underset{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}}{\text{argmax}} \frac{N(f, f_{m_1}, f_{m_2})}{|F||F_{m_1}||F_{m_2}|}$$

- **Max Sum-of-Cardinality Normalized Counts:** In this case the sum of the of the random variable cardinalities are used for the normalization, giving:

$$g(f, f_1, f_2, f_3) = p_{GBO}(f|f_{\ell_1}, f_{\ell_2})$$

where

$$(\ell_1, \ell_2) = \underset{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}}{\text{argmax}} \frac{N(f, f_{m_1}, f_{m_2})}{|F| + |F_{m_1}| + |F_{m_2}|}$$

- **Max Sum-of-Log-Cardinality Normalized Counts:** In this case the sum of the of the random variable natural log cardinalities are used for the normalization, giving:

$$g(f, f_1, f_2, f_3) = p_{GBO}(f|f_{\ell_1}, f_{\ell_2})$$

where

$$(\ell_1, \ell_2) = \underset{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}}{\text{argmax}} \frac{N(f, f_{m_1}, f_{m_2})}{\ln|F| + \ln|F_{m_1}| + \ln|F_{m_2}|}$$

Several other $g(f, f_1, f_2, f_3)$ functions were also implemented, including minimum versions of the above (these are identical to the above except that the the *argmin* rather than the *argmax* function is used. Specifically,

- **Min Counts**
- **Min Normalized Counts**
- **Min Num-Words Normalized Counts**
- **Min Backoff-Graph Node Probability**
- **Min Product-of-Cardinality Normalized Counts**
- **Min Sum-of-Cardinality Normalized Counts**
- **Min Sum-of-Log-Cardinality Normalized Counts**

Still, several more $g(f, f_1, f_2, f_3)$ functions were implemented that take neither the min nor max of their arguments. These include:

- **Sum:**

$$g(f, f_1, f_2, f_3) = \sum_{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}} p_{GBO}(f|f_{m_1}, f_{m_2})$$

- **Average (arithmetic mean):**

$$g(f, f_1, f_2, f_3) = \frac{1}{3} \sum_{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}} p_{GBO}(f|f_{m_1}, f_{m_2})$$

- **Product:**

$$g(f, f_1, f_2, f_3) = \prod_{(m_1,m_2)\in\{(1,2),(1,3),(2,3)\}} p_{GBO}(f|f_{m_1}, f_{m_2})$$

- **Geometric Mean:**

$$g(f, f_1, f_2, f_3) = \left(\prod_{(m_1,m_2) \in \{(1,2),(1,3),(2,3)\}} p_{GBO}(f|f_{m_1}, f_{m_2}) \right)^{1/3}$$

- **Weighted Mean:**

$$g(f, f_1, f_2, f_3) = \prod_{(m_1,m_2) \in \{(1,2),(1,3),(2,3)\}} p_{GBO}^{\gamma_{m_1,m_2}}(f|f_{m_1}, f_{m_2})$$

  where the weights γ_{m_1,m_2} are specified by the user.

Note that these last examples correspond to approaches where multiple backoff paths are used to form a final probability score rather than a single backoff path for a given factor value and context (as described earlier in this section). Also note that rather taking the sum (respectively average, product, etc.) over all children in the backoff graph, it can be beneficial to take the sum (respectively average, product, etc.) over an *a priori* specified subset of the children. As will be seen, this subset functionality (and all of the above) was implemented as extensions to the SRI toolkit as described in the next section. Readers interested in the perplexity experiments rather than the details of the implementation may wish to skip to Section 5.5.

5.4 SRI Language Model Toolkit Extensions

Significant code extensions were made to the SRI language modeling toolkit SRILM [50] during the time of the workshop in order to support both factored language models and generalized backoff.

Essentially, the SRI toolkit was extended with a graphical-models-like syntax (similar to [8]) to specify a given statistical model. A graph syntax was used to specify the given child variable and its parents, and then a specific syntax was used to specify each possible node in the backoff graph (Figure 11), a set of node options (e.g., type of smoothing), and the set of each nodes possible backoff-graph children. This therefore allowed the possibility to create backoff graphs of the sort shown in the right of Figure 10. The following sections serve to provide complete documentation for the extensions made to the SRILM toolkit during the workshop.

5.4.1 New Programs: `fngram` and `fngram-count`

Two new programs have been added to the SRILM program suite. They are `fngram` and `fngram-count`. These programs are analogous to the normal SRILM programs `ngram` and `ngram-count` but they behave in somewhat different ways.

The program `ngram-count` will take a factored language data file (the format is described in Section 5.4.2) and will produce both a count file and optionally a language model file. The options for this program are described below. One thing to note about these options is that, unlike `ngram-count`, `fngram-count` does not include command-line options for language model smoothing and discounting at each

language model level. This is because potentially a different set of options needs to be specified for each node in the backoff graph (Figure 9). Therefore, the smoothing and discounting options are all specified in the language model description file, described in Section 5.4.3.
The following are the options for the `fngram-count` program, the program that takes a text (a stream of feature bundles) and estimates factored count files and factored language models.

- `-factor-file` $< str >$ Gives the name of the FLM description file that describes the FLM to use (See Section 5.4.3).
- `-debug` $< int >$ Gives debugging level for the program.
- `-sort` Sort the ngrams when written to the output file.
- `-text` $< str >$ Text file to read in containing language model training information.
- `-read-counts` Try to read counts information from counts file first. Note that the counts file are specified in the FLM file (Section 5.4.3).
- `-write-counts` Write counts to file(s). Note that the counts file are specified in the FLM file (Section 5.4.3).
- `-write-counts-after-lm-train` Write counts to file(s) after (rather than before) LM training. Note that this can have an effect if Kneser-Ney or Modified Kneser-Ney smoothing is in effect, as the smoothing method will change the internal counts. This means that without this option, the non Kneser-Ney counts will be written.
- `-lm` Estimate and write lm to file(s). If not given, just the counts will be computed.
- `-kn-counts-modified` Input counts already modified for KN smoothing, so don't do it internally again.
- `-no-virtual-begin-sentence` Do **not** use a virtual start sentence context at the sentence begin. A FLM description file describes a model for a child variable C given its set of parent variables $P_1, P_2, \ldots, P_N$. The parent variables can reside any distance into the past relative to the parent variable. Let us say that the maximum number of feature bundles (i.e., time slots) into the past is equal to τ. When the child variable corresponds to time slot greater than τ it is always possible to obtain a true parent variable value for LM training. Near the beginning of sentences (i.e., at time slots less than τ) there are no values for one or more parent variable. The normal behavior in this case is to assume a virtual start of sentence value for all of these parent values (so we can think of a sentence as containing an infinite number of start of sentence tokens extending into the past relative to the beginning of the sentence). For example, if an FLM specified a simple trigram model over words, the following

contexts would be used at the beginning of a sentence: $< s > < s >$ `w1`, $< s >$ `w1 w2`, `w1 w2 w3`, and so on.

This is not the typical behavior of SRILM in the case of a trigram, and in order to recover the typical behavior, this option can be used, meaning do not add virtual start of sentence tokens before the beginning of each sentence.

Note that for a FLM simulating a word-based n gram, if you want to get *exactly* the same smoothed language model probabilities as the standard SRILM programs, you need to include the options `-no-virtual-begin-sentence` and `-nonull`, and make sure `gtmin` options in the FLM are the same as specified on the command line for the usual SRILM programs.

- `-keepunk` Keep the symbol $< unk >$ in LM. When $< unk >$ is in the language model, each factor automatically will contain a special $< unk >$ symbol that becomes part of the valid set of values for a tag. If $< unk >$ is in the vocabulary, then probabilities will (typically) be given for an instance of this symbol even if it does not occur in the training data (because of backoff procedure smoothing, assuming the FLM options use such backoff).

  If $< unk >$ is not in the language model (and it did not occur in training data), then zero probabilities will be returned in this case. If this happens, and the child is equal to the $< unk >$ value, then that will be considered an out-of-vocabulary instance, and a special counter will be incremented (and corresponding statistics reported). This is similar to the standard behavior of SRILM.

- `-nonull` Remove $< NULL >$ from LM. Normally the special word $< NULL >$ is included as a special value for all factors in a FLM. This means that the language model smoothing will also smooth over (and therefore give probability to) the $< NULL >$ token. This option says to remove $< NULL >$ as a special token in the FLM (unless it is encountered in the training data for a particular factor).

  Note that for a FLM simulating a word-based n gram, if you want to get *exactly* the same smoothed language model probabilities as the standard SRILM programs, you need to include the options `-no-virtual-begin-sentence` and `-nonull`, and make sure `gtmin` options in the FLM are the same as specified on the command line for the usual SRILM programs.

- `-meta-tag` Meta tag used to input count-of-count information. Similar to the other SRILM programs.

- `-tolower` Map vocabulary to lowercase. Similar to the other SRILM programs.

- `-vocab` vocab file. Read in the vocabulary specified by the vocab file. This also means that the vocabulary is closed, and anything not in the vocabulary is seen as an OOV.

- `-non-event` non-event word. The ability to specify another "word" that is set to be a special *non-event* word. A non-event word is one that is not smoothed over (so if it is in the vocabulary, it still won't effect LM probabilities). Note that since we are working with FLMs, you need to specify the corresponding tag along with the word in order to get the correct behavior. For example, you would need to specify the non-event word as "`P-noun`" for a part of speech factor, where "`P`" is the tag. Non-events don't count as part of the vocabulary (e.g., the size of the vocabulary), contexts (i.e., parent random variable values) which contain non-events are not estimated or included in a resulting language model, and so on.
- `-nonevents` non-event vocabulary file. Specifies a file that contains a list of non-event words — good if you need to specify more than one additional word as a non-event word.
- `-write-vocab` write vocab to file. Says that the current vocabulary should be written to a file.

The following are the options for the `fngram` program. This program uses the counts and factored language models previously computed, and can either compute perplexity on another file, or can re-score n-best lists. The n-best list rescoring is similar to the

- `-factor-file` build a factored LM, use factors given in file. Same as in `fngram-count`.
- `-debug` debugging level for lm. Same as in `fngram-count`.
- `-skipoovs` skip n-gram contexts containing OOVs.
- `-unk` vocabulary contains $< unk >$ Same as in `fngram-count`.
- `-nonull` remove $< NULL >$ in LM. Same as in `fngram-count`.
- `-tolower` map vocabulary to lowercase. Same as in `fngram-count`.
- `-ppl` text file to compute perplexity from. This specifies a text file (a sequence of feature bundles) on which perplexity should be computed. Note that multiple FLMs from the FLM file might be used to compute perplexity from this file simultaneously. See Section 5.4.3.
- `-escape` escape prefix to pass data through. Lines that begin with this are printed to the output.
- `-seed` seed for randomization.
- `-vocab` vocab file. Same as in `fngram-count`.
- `-non-event` non-event word. Same as in `fngram-count`.
- `-nonevents` non-event vocabulary file. Same as in `fngram-count`.

- `-write-lm` re-write LM to file. Write the LM to the LM files specified in the FLM specification file.
- `-write-vocab` write LM vocab to file. Same as in `fngram-count`.
- `-rescore` hyp stream input file to rescore. This is similar to the `-rescore` option in the program `ngram` (i.e., `ngram` supports a number of different ways of doing n-best rescoring, while `fngram` supports only one, namely the one corresponding to the `-rescore` option).
- `-separate-lm-scores` print separate lm scores in n-best file. Since an FLM file might contain multiple FLMs, this option says that separate scores for each FLM and for each item in the n-best file should be printed, rather than combining them into one score. This would be useful to, at a later stage, combine the scores for doing language-model weighting.
- `-rescore-lmw` rescoring LM weight. The weight to apply to the language model scores when doing rescoring.
- `-rescore-wtw` rescoring word transition weight. The weight to apply to a word transition when doing rescoring.
- `-noise` noise tag to skip, similar to the normal SRILM programs.
- `-noise-vocab` noise vocabulary to skip, but a list contained in a file. Similar to the normal SRILM programs.

5.4.2 Factored Language Model Training Data File Format

Typically, language data consists of a set of words which are used to train a given language model. For an FLM, each word might be accompanied by a number of factors. Therefore, rather than a stream of words, a stream of vectors must be specified, since a factored language model can be seen to be a model over multiple streams of data, each stream corresponding to a given factor as it evolves over time.

To support such a representation, language model files are assumed to be a stream of *feature bundles*, where each feature in the bundle is separated from the next by the "`:`" (colon) character, and where each feature consists of a $< tag >$-$< value >$ pair.[6] The $< tag >$ can be any string (of any length), and the toolkit will automatically identify the tag strings in a training data file with the corresponding string in the language-model specification file (to be described below). The $< value >$ can be any string that will, by having it exist in the training file, correspond to a valid value of that particular tag. The tag and value are separated by a dash "`-`" character.

[6]Note, that the `tag` can be any tag such as a purely data-driven word class, and need not necessarily be a lexical tag or other linguistic part-of-speech tag. The name "tag" here therefore is used to represent any feature.

A language model training file may contain many more tag-value pairs than are used in a language model specification file — the extra tag-value pairs are simply ignored. Each feature bundle, however, may only have one instance of a a particular tag. Also, if for a given feature a tag is missing, then it is assumed to be the special tag "W" which typically will indicate that the value is a word. Again, this can happen only one time in a given bundle.

A given tag-value pair may also be missing from a feature bundle. In such a case, it is assumed that the tag exists, but has the special value "NULL" which indicates a missing tag (note that the behavior for the start and end of sentence word is different, see below). This value may also be specified explicitly in a feature bundle, such as S-NULL. If there are many such NULLs in a given training file, it could reduce file size by not explicitly stating them and using this implicit mechanism.

The following are a number of examples of training files:

```
the brown dog ate a bone
```

In this first example, the language model file consists of a string of what are (presumably) words that have not been tagged. It is therefore assumed that they are words. The sentence is equivalent to the following where the word tags are explicitly given:

```
W-the W-brown W-dog W-ate W-a W-bone
```

If we then wanted to also include part-of-speech information as a separate tag and using the string "P" to identify part of speech, then could would be specified as follows:

```
W-the:P-article W-brown:P-adjective W-dog:P-noun
W-ate:P-verb W-a:P-article W-bone:P-noun
```

The order of the individual features within a bundle do not matter, so the above example is identical to the following:

```
P-article:W-the P-adjective:W-brown P-noun:W-dog
P-verb:W-ate P-article:W-a P-noun:W-bone
```

More generally, here is a string from the call-home Arabic corpus that has been tagged using the methodology described in Section 4.5.1. Note that the example shows one feature bundle per line only for formatting purposes in this document. In the actual file, an entire sentence of feature bundles is given on each line (i.e., a newline character separates one sentence from the next).

```
<s>
W-Tayyib:M-adj+masc-sg:S-Tayyib:R-Tyb:P-CVyyiC
```

```
W-xalAS:M-noun+masc-sg:S-xalAS:R-xlS:P-CaCAC
W-lAzim:M-adj+masc-sg:S-lAzim:R-Azm:P-CVCiC
W-nitkallim:M-verb+subj-1st-plural:S-itkallim:R-klm:P-iCCaCCiC
W-carabi:M-adj+masc-sg:S-carabi:R-crb:P-CaCaCi
W-cala$An:M-prep:S-cala$An:R-cl$:P-CaCaCAn
W-humma:M-pro+nom-3rd-plural:S-humma:R-hm:P-CuCma
W-cayzIn:M-pple-act+plural:S-cAyiz:R-cwz:P-CVyiC
W-%ah:M-%ah:S-%ah:R-%ah:P-%ah
W-il+mukalmaB:M-noun+fem-sg+article:S-mukalmaB:R-klm:P-CuCaCCaB
W-tibqa:M-verb+subj-2nd-masc-sg:S-baqa:R-bqq:P-CaCa
W-bi+il+*FOR:M-bi+il+*FOR:S-bi+il+*FOR:R-bi+il+*FOR:P-bi+il+*FOR
W-*FOR:M-*FOR:S-*FOR:R-*FOR:P-*FOR
W-cala$An:M-prep:S-cala$An:R-cl$:P-CaCaCAn
W-humma:M-pro+nom-3rd-plural:S-humma:R-hm:P-CuCma
W-biysaggilu:M-verb+pres-3rd-plural:S-saggil:R-NULL:P-NULL
</s>
```

The example shows one sentence of Arabic that provides the words (tag name "W"), has been tagged with the morphological class (tag name "M"), the stem (tag name "S"), the root of the word (tag name "R"), and the word pattern (tag name "P"). See Section 4.5.1) for a definition of these constructs and how and why they apply well to Arabic. Just like in a normal word file in SRILM, the sentence begins with the special (word) token $< s >$ to indicate the start of sentence, and $< /s >$ to indicate the end of sentence.

For the purposes of a FLM file, giving a start of sentence without any tags means that all other tags will also have a start of sentence value. Note that this is distinct from the standard missing tag behavior (see above) where a missing tag is filled in with the values NULL.

5.4.3 Factored Language Model File

An FLM file consists of one or more specifications of a given FLM. When multiple FLMs are specified in a file, each one is simultaneously trained, used for perplexity computation, or used for sentence n-best scoring (depending on the program that is called and the given command-line options, see Section 5.4.1). This section describes the format and options of these files.

An FLM file may contain comments, which consist of lines that begin with the "##" character pair — anything after that is ignored.

An FLM file begins with an integer N that specifies the number of FLM specifications that are to follow. After the N FLMs, the remainder of the file is ignored, an can be considered to be a comment. The integer N specifies the number of FLMs that are to be simultaneously used by the FLM programs.

Each FLM specification consists of a specification of a child, the number of parents, a set of that many parent names, a count file name, a language model file name, and a sequence of nodes in the corresponding backoff-gram and a set of node options. We next provide a number of simple examples:

Simple Unigram: Here is an FLM for a simple word unigram:

```
## word unigram
W : 0  word_1gram.count.gz word_1gram.lm.gz 1
   0b0 0b0 kndiscount gtmin 1
```

This is a unigram over words (the tag is `W`). It is a unigram because there are zero (0) parents. It uses a count file named `word_1gram.count.gz` and a language model named `word_1gram.lm.gz`, and it specifies one (1) additional backoff-gram node specification which follows on the next line.

The backoff-gram node specification corresponds to the node with no parents (the number `0b0` is a binary string saying that there are no parents). The next number indicates the set of parents that may be dropped, but in this case it is ignored since there are no parents, and then it says that Kneser-Ney discounting should be used, with a minimum `gtmin` count of unity (1) before backing off to (in this case) the unigram.

Simple Trigram: The next example shows a FLM for a standard word-based trigram with a number of options.

```
## normal trigram LM
W : 2 W(-1) W(-2) word_3gram.count.gz word_3gram.lm.gz 3
 W1,W2   W2  kndiscount gtmin 2 interpolate
    W1   W1  kndiscount gtmin 1 interpolate
     0    0  kndiscount gtmin 1
```

In this case, the child is again the word variable (tag is `W`), which has two parents, the word at the previous time slot (`W(-1)`), and the word two time-slots into the past (`W(-2)`). It uses a count file named `word_3gram.count.gz`, a language model file named `word_3gram.lm.gz`, and specifies options for three (3) backoff-gram nodes.

The first backoff-graph node corresponds to the complete model, meaning that there are two parents which are the two preceding words. The syntax is `W1` and `W2` which means the preceding word and two words ago. Rather than respecify the time-slot index as being negative, positive indices are used here to shorten the line lengths and since it is never the case that there is a need to specify a model that has parents coming both from the future and the past. In any event, the first backoff-graph node corresponds to the normal trigram since both parents are present, as given by the string `W1,W2`.

The next string in the line is the set of parents that may be dropped in going from this backoff-gram node to nodes in the next level below. In this case, only the parent `W2` may be dropped, so this therefore

corresponds to a normal trigram language model where parents are dropped in the priority order of most-distant temporal parent first.

Following this are the set of node options for this node. It says to use Kneser-Ney discounting, to have a minimum count of two (2) in this case, and to produce an interpolated language model between this backoff-graph node and the nodes below.

The next line specifies the node identifier, which is `W1` in this case meaning it is the model containing only the previous word. The next string is also `W1` meaning that that parent may be dropped to go down to the next level, and again a set of node options are given.

The last line gives the unigram model where there are no parents (specified by the number 0), no additional parents may be dropped (again with 0) and a set of node options.

Trigram With Time-Reversed Backoff Path: The next example shows a FLM for a word-based trigram, but where the parent variables are dropped in least-distant-in-time order, the reverse order of what is done in a typical trigram.

```
## trigram with time-reversed backoff path
W : 2 W(-1) W(-2) word_3gram.count.gz word_3gram.lm.gz 3
 W1,W2   W1  kndiscount gtmin 2 interpolate
    W2   W2  kndiscount gtmin 1 interpolate
     0    0  kndiscount gtmin 1
```

As can be seen, the backoff graph node for the case where two parents are present (node `W1,W2`) says that only one parent can be dropped, but in this case that parent is `W1`. This means that the next node to be used in the backoff graph corresponds to the model $p(W_t|W_{t-2})$. It should be understood exactly what counts are used to produce this model. $p(W_t|W_{t-2})$ corresponds to the use of counts over a distance of two words, essentially integrating all possible words in between those two words. In other words, this model will use the counts $N(w_{t-2}, w_t) = \sum_{w_{t-1}} N(w_{t-2}, w_{t-1}, w_t)$ to produce, in the maximum likelihood case (i.e., no smoothing), the model $p_{ML}(W_t|W_{t-2}) = N(w_{t-2}, w_t)/N(w_{t-2})$ where $N(w_{t-2}) = \sum_{w_t} N(w_{t-2}, w_t)$.

Note that this model is quite distinct from the case where the model $p(W_t|W_{t-1})$ is used, but where the word from two time slots ago is substituted into the position of the previous word (this is what a SRILM skip-n-gram would do). We use the following notation to make the distinction. The FLM above corresponds to the case where $P(W_t = w_t|W_{t-2} = w_{t-2})$ is used for the backoff-graph node probability. A skip-n-gram, on the other hand, would utilize (for the sake of interpolation) $P(W_t = w_t|W_{t-1} = w_{t-2})$ where the random variable W_{t-1} is taken to have the value of the word w_{t-2} from two time-slots into the past. In this latter case, the same actual counts are used $N(w_t, w_{t-1})$ to produce the probability, only the word value is changed. In the former case, an entirely different count quantity $N(w_t, w_{t-2})$ is used.

Note that a skip-n-gram would not work in the case where the parents correspond to different types of random variables (e.g., if parents were words, stems, morphs, and so on). For example, if one parent is a word and another is a stem, one could not substitute the stem in place of the word unless a super-class is created that is the union of both words and stems. Such a super-class could have an effect on smoothing (in particular for the Kneser-Ney case) and table storage size. A FLM allows different random variable types to be treated differently in each parent variable, and allows counts to be computed separately for each subset of parents.

Stem Given Morph and 2-Word Context: Our next example is where the child and parent variables are not of the same type (i.e., they are not all words). This example corresponds to a probability model for $p(S_t|M_t, W_{t-1}, W_{t-2})$ where S_t is the stem at time t, M_t is a morphological class at time t, and W_t is the word at time t. This model can be specified as follows:

```
## stem given morph word word
S : 3 M(0) W(-1) W(-2) s_g_m0w1w2.count.gz s_g_m0w1w2.lm.gz 4
  M0,W1,W2  W2 kndiscount gtmin 1 interpolate
  M0,W1     W1 kndiscount gtmin 1 interpolate
  M0        M0 kndiscount gtmin 1
  0         0  kndiscount gtmin 1
```

In this case, the stem `S` at time t is given as the child variable, and there are three parents: 1) the morph at the current time `M(0)`, 2) the word at the previous time `W(-1)`, and 3) the word two time-slots ago `W(-2)`. It produces and uses a count file named `s_g_m0w1w2.count.gz` and a language model file named `s_g_m0w1w2.lm.gz`, and specifies options for four backoff-graph nodes.

The first node, as always, corresponds to when all parent variables are present `M0,W1,W2`. This node will back-off only to the case when one parent is dropped, namely `W2`. Again it uses Kneser-Ney smoothing with a minimum count of unity and interpolation. The next node (on the next line) is for the case when the parents `M0,W1` are present (i.e., the backoff-graph node above when the parent `W2`) has been dropped. This node may only drop parent `W1`, and has similar additional node options. The next two lines specify last two backoff-graph node options, namely for nodes `M0` and for the case when there are no parents. The options should be clear at this point from the above discussion.

Note that all of the examples we have encountered so far correspond to a fixed and single backoff path from root to leaf node (as shown in the examples in Figure 12). The next several examples indicate multiple backoff-paths.

trigram with generalized backoff: The following example shows a trigram that allows a backoff path, meaning that both backoff-paths in the backoff graph are traversed in order to obtain a final probability. This

case therefore must specify and use one of the combining functions mentioned in Section 5.3.3.

Here is the example:

```
## general backoff, max trigram LM
W : 2 W(-1) W(-2) word_gen3gram.max.count.gz word_gen3gram.max.lm.gz 4
   W1,W2 W1,W2 kndiscount gtmin 2 combine max strategy bog_node_prob interpolate
   W2    W2    kndiscount gtmin 1 interpolate
   W1    W1    kndiscount gtmin 1 interpolate
   0     0     kndiscount gtmin 1 kn-count-parent 0b11
```

In this case, four backoff graph nodes are specified. The first (as always) is the case when all parents are present, in this case `W1,W2`. The next field says that both parents `W1,W2` are allowed to drop to descend down to the next level. In other words, the next field is a set of parents which may be dropped one-by-one, and implicitly defines the set of lower nodes in the backoff graph. This is then followed by a set of node options: Kneser-Ney discounting, a minimum count of two, and a combination strategy that takes the maximum `max` of the lower nodes' backoff graph probabilities `bog_node_prob` (backoff-graph node probability). This option therefore corresponds to the *Max Backoff-Graph Node Probability* case given in Section 5.3.3.

The remaining nodes in the backoff graph are specified in the same way as given in the examples above.

Dropping more than one parent at a time - Skipping a level in a FLM backoff-graph: Using the mechanism of minimum counts (`gtmin`), it is possible also to produce FLMs that skip an entire level on the backoff-graph altogether. In other words, this is the case when it is desirable to drop more than one parent at a time, going from, say, the model $P(C|P_1, P_2, P_3)$ backing off directly to either $P(C|P_1)$ or $P(C|P_2)$.

The following provides an example of how this can be done.

```
## word given word morph stem
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 2 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 combine mean
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

In this case, the first backoff-graph node `W1,M1,S1` containing all parents (and corresponding to $p(W_t|W_{t-1}, M_{t-1}, S_{t-1})$ says that only the word variable may be dropped, moving down to node `M1,S1` (model $p(W_t|M_{t-1}, S_{t-1})$). . This node, however, has a very large minimum count threshold (a `gtmin` of $100,000,000$). This means that this node in the language model will not "hit", and all scores from this backoff graph node will be a combination of the lower-level backoff graph nodes. The node says that both parents

`S1` and `M1` may be dropped, meaning that there are two models at the next lower level in the backoff graph, corresponding to models $p(W_t|M_{t-1})$ and $p(W_t|S_{t-1})$. The mean of the probabilities of these models will be used to produce a score for the `M1,S1` node (and this is specified by the string `combine mean`).

As can be seen, using a combination of a restricted set of parents that may be dropped at a given backoff-graph node, and a very large `gtmin` count, a wide variety of backoff graphs (some of which can even skip levels) can be specified.

General Description of FLM syntax: The above examples gave a taste of some of the language models that can be specified using the syntax. This section describes the syntax and functionality in the most general case. Many of these models will be used and further described in Section 5.5.

A FLM consists of a *model specifier* line of text, followed by one or more *node specifier* lines of text.

The *model specifier* line is a graphical-model inspired syntax for specifying a statistical model over the tags that exist in a set of training data. This line takes the form:

```
child : num_parents par_1 par_2 ... par_N  cnt_file lm_file num_bg_nodes
```

The name `child` is the child random variable, and corresponds to any one of the tags that exist in language model training data. For example, using the tags that are specified in the CallHome training file given in Section 5.4.2, the child could be any of the strings `W`, `M`, `S`, `R`, or `P`.

The next field is the number of parent random variables in the model, and is an integer $N \geq 0$.

The next set of fields are the N parents, each one taking the form `parent_name(time_offset)`. The string `parent_name` can again be any of the tags that exist in the training data. The `time_offset` value is an integer ≤ 0 that gives how many time-slots into the past the parent should reside.

What follows is a name to be used for the count file (keeping language model counts) and then the name of the language model file. Note that both count files and language model file are specified since counts are, for certain cases as described in Section 5.3.3, needed to determine the backoff probability.

The last field on the line is `num_bg_nodes`, which is the number of backoff-graph node specifications that are following. Only the next such-number of lines are assumed to be part of the FLM file (anything after that is either ignored, or is assumed to be part of the next FLM specified in the file). One should be careful to ensure that the number of actual backoff-graph node specifiers given is equal to this number, as it is easy to add more node specifiers while forgetting to update this field.

Once the *model specifier* is given, it is followed by `num_bg_nodes` *model specifier*s. Each *model specifier* consists of the following form:

```
parent_list drop_list [node options]
```

The `parent_list` is a comma-separated list (without spaces) of parent variables that correspond to this node. In other words, each node in a backoff graph can be identified with a set of parent variables, and

this string identifies the node by giving that set. The parents are specified in a shortened syntax — each parent is given using its name (again one of the tags in the file), and the absolute value of the time offset with no space. Examples of parent lists include `W1,W2,W3` to give the three preceding words, `W1,M1,S1` to give the preceding word, morph, and stem, and so on.

The next field `drop_list` gives the list of parents that may be dropped one at a time from this backoff-graph node. This is the field that specifies the set of lower-level backoff-graph nodes that are used in case there is not a language-model "hit" at this level. For example, if `parent_list` is `W1,M1,S1` and `drop_list` is `W1`, then the only next lower-level node used is the one with `parent_list` `M1,S1`. If, on the other hand, a `drop_list` of `W1,M1,S1` is used for parent list `W1,M1,S1`, then three nodes are used for the next lower-level — namely, the nodes with parent lists `M1,S1`, `W1,S1`, and `W1,M1`.

The node options are one or more option for this backoff-graph node. Each backoff-graph node can have its own discounting, smoothing, combination options, and so on. Many of the node options are similar to the command-line options that can be given to the SRLIM program `ngram-count`. These options are included in the FLM file because a different set might be given for each node. The following gives a list of all the node options that are currently supported.

- `gtmin [num]` the lower GT discounting cutoff. When not using Good-Turing discounting, this gives the threshold that the count needs to meet or exceed in order for there to be a language model "hit" at this backoff graph node. Therefore, `gtmin` corresponds to the τ variables in the equations given in Section 5.3.
- `gtmax [num]` upper GT discounting cutoff. Note that if only this and `gtmin` are specified, then Good-Turing smoothing is used.
- `gt [fileName string]` Good-Turing discount parameter file for this node.
- `cdiscount [double]` Says that constant discounting should be used, and this gives the discounting constant.
- `ndiscount` Says that natural discounting should be used.
- `wbdiscount` Says that Witten-Bell discounting should be used.
- `kndiscount` Says that modified Kneser-Ney discounting should be used.
- `ukndiscount` Says that *unmodified* (i.e., the original or normal) Kneser-Ney discounting should be used.
- `kn-counts-modified` says that the input counts already modified for KN smoothing (this is similar to `ngram-count`.

- `kn-counts-modify-at-end` says that the counts should be turned into KN-counts after discount estimation rather than before. In other words, if this option is specified, the quantities such as n_1, n_2, n_3, n_4, and so on (typically used for modified KN discounting) should be computed from the usual counts rather than from the meta-meta-counts needed for Kneser-Ney smoothing.
- `kn [fileName string]` The Kneser-Ney discount parameter file for this backoff-graph node.
- `kn-count-parent [parent spec]` The backoff-graph parent that is used to compute the meta-meta-counts needed for Kneser-Ney smoothing. Normally, the parent is the node immediately above the current node in the backoff graph, but it is sometimes useful to specify a different parent, especially when the set of parents do not correspond to the same random variables at different times (i.e., when the parents are truly different random variables).
- `interpolate` use interpolated estimates for computing the probabilities for this node. In other words, the probabilities should be interpolated with hits in the current nodes and whatever the $g()$ function returns for the lower node probabilities. This is therefore a generalization of the `interpolated` option in `ngram-count`.
- `write [fileName string]` Write the counts, just of this node alone, to the given file.
- `combine [option]` This option is active only when there multiple backoff paths (backoff-graph children) possible (i.e., when the `drop_list` specifies more than one parent variable. See Section 5.3.3. In this case, `[option]` is one of:
  - `max` the maximum of the next backoff-graph level nodes. This is the **default** if no `combine` option is given.
  - `min` the maximum of the next backoff-graph level nodes
  - `sum` the maximum of the next backoff-graph level nodes
  - `avg||mean` the arithmetic average (mean) of the next backoff-graph level nodes
  - `prod` the product of the next backoff-graph level nodes
  - `gmean` the geometric mean of the next backoff-graph level nodes
  - `wmean [` $< node\_specweight >$ $< node\_specweight >$ `... ]` the weighted mean of the next backoff-graph level nodes. The weights are given by providing a node specification (a list of comma separated parents) to identify the node, and then a weight for that node. For example, one might used:

    ```
    wmean W1,M2 7 W1,S2 3
    ```

    which means that backoff-grpah node `W1,M2` has a weight of 7, and node `W1,S2` has a weight of 3.

- `strategy [option]` This option also is active only when there are multiple backoff paths (backoff-graph children) possible, and when the `combine` option above is either `min` or `max`. See Section 5.3.3 for more information. In this case, `[option]` is one of:
  - `counts_sum_counts_norm` The *Max/Min Normalized Counts* cases in Section 5.3.3. This is the default case if no `strategy` option is given.
  - `counts_no_norm` The *Max/Min Counts* cases in Section 5.3.3.
  - `counts_sum_num_words_norm` The *Max/Min Num-Words Normalized Counts* cases in Section 5.3.3.
  - `counts_prod_card_norm` The *Max/Min Product-of-Cardinality Normalized Counts* cases in Section 5.3.3.
  - `counts_sum_card_norm` The *Max/Min Sum-of-Cardinality Normalized Counts* cases in Section 5.3.3.
  - `counts_sum_log_card_norm` The *Max/Min Sum-of-Log-Cardinality Normalized Counts* cases in Section 5.3.3.
  - `bog_node_prob` The *Max/Min Backoff-Graph Node Probability* cases in Section 5.3.3.

Other ways to specify backoff-graph nodes and drop sets. A backoff graph node is typically identified using a comma separated list of parents. Similarly, a set of parents that may be dropped is also specified using such a list. In some cases, specifying these lists can be tedious, and it might be more concise (albeit more error prone) to specify nodes and sets using a numeric representation.

Each parent list may also be specified using an integer, where the integer is a bit representation of the parents in the set. When the line in the FLM file is given of the form:

```
child : num_parents par_1 par_2 ... par_N  cnt_file lm_file num_bg_nodes
```

then `par_1` corresponds to bit 0, `par_2` corresponds to bit 1, and so on until `par_N` corresponds to bit $N-1$, in an N-element bit vector. This means that the left-most parent corresponds to the least significant bit, and the right most parent the most significant bit. The reason for this is, given the string `W(-1) W(-2) W(-3)` as a parent list, the right-most parent is the *highest-order* parent.

These bit vectors may be specified numerically rather than as a string form of a list of parents. The numeric form may be either a decimal integer, a hexadecimal integer (one preceded by `0x`), or a binary integer (one preceded by `0b`).

For example, a normal trigram can be specified as follows:

```
W : 2 W(-1) W(-2) word_3gram_rev.count.gz word_3gram_rev.lm 3
```

```
  0b11 0b10 kndiscount gtmin 1
  0b01 0b01 kndiscount gtmin 1
  0b00 0b00 kndiscount gtmin 1
```

a construct equivalent to the following:

```
W : 2 W(-1) W(-2) word_3gram_rev.count.gz word_3gram_rev.lm 3
  0x3 0x2 kndiscount gtmin 1
  0x1 0x1 kndiscount gtmin 1
  0x0 0x0 kndiscount gtmin 1
```

Any combination of decimal, hexadecimal, binary, and string form can be used in a FLM file, so the following is valid.

```
W : 2 W(-1) W(-2) word_3gram_rev.count.gz word_3gram_rev.lm 3
  W1,W2 0b10 kndiscount gtmin 1
  0b01 W1 kndiscount gtmin 1
  0    0 kndiscount gtmin 1
```

One way that this can be useful is when specifying very large backoff-graphs with little sparsity. For example, the example in Figure 13 shows a FLM for generalize backoff when there six parents, and all backoff-paths are taken. As can be seen, the use of a numeric representation to specify parents greatly facilitates the declaration of this model.

5.5 Perplexity Results on CallHome Arabic

In this section, we present a number of perplexity results when using FLMs and generalized backoff on the CallHome Arabic corpus. As will be seen, the approach outlined above shows great promise.

Time limitations in the short 6-week duration of the workshop prohibited us from performing anything more than perplexity experiments on these models. Nevertheless, as will be seen, it has been possible to obtain simple models that achieve marked perplexity reductions using the factored information relative to their word-only baselines. In certain cases, word bigrams (FLMs whose context does not span any more than a single word into the past) have been found that have a perplexity significantly lower than that of the optimized baseline trigram, a result with significant implications for first-pass recognition, where a bigram results has a significantly smaller search space than that of a trigram [36]..

The FLM extensions to SRILM were applied to the CallHome Arabic database and its factors. As was mentioned in Section 5.2, and Section 4.5.1, this corpus is tagged with words (tag name “W”), morphological classes (tag name “M”), stems (tag name “S”), word roots (tag name “R”), and words patterns (tag name “P”). In this case, therefore, the first of the two research issues for an FLM as given in Section 5.2 are solved —

the factors are defined. It is therefore necessary to find a good statistical model over these factors, one that minimizes perplexity and word-error rate in a speech recognition system.

It is necessary to provide all the details regarding the training and test data, the procedure by which the word factors were derived, and the language model itself. The CallHome corpus, and the training and test set as described in Section 2.1. Note that only one training and test set was utilized given the severe limitations of the amount of available data. For all language models, the factors that were used included the word itself (tag name "W"), the morphological class (tag name "M"), the stem (tag name "S"), the root of the word (tag name "R"), and the word pattern (tag name "P"). See Section 4.5.1) for more information regarding these factors.

In order to report all information regarding a FLM, it is necessary not only to report the factors that are used and the model structure over those factors, but also the generalized backoff method and the other smoothing options at each node in the backoff graph. Since all of this information is given in the FLM definition (see Section 5.4.3), we will here use the FLM definitions themselves to provide this information.

We begin by providing the perplexity for a simple optimized unigram, bigram, and trigram language model.

```
## normal unigram LM
## 0 zeroprobs, logprob= -93324.9 ppl= 279.382 ppl1= 799.793
W : 0 word_1gram.count.gz word_1gram.lm.gz 1
     0    0  kndiscount gtmin 1


## normal bigram LM
## 0 zeroprobs, logprob= -85610.4 ppl= 175.384 ppl1= 460.267
W : 1 W(-1) word_2gram.count.gz word_2gram.lm.gz 2
    W1   W1  kndiscount gtmin 1 interpolate
     0    0  kndiscount gtmin 1


## BASLINE TRIGRAM: normal trigram LM
## logprob= -85370 ppl= 172.857 ppl1= 452.409
W : 2 W(-1) W(-2) word_3gram.count.gz word_3gram.lm.gz 3
 W1,W2   W2  kndiscount gtmin 2 interpolate
    W1   W1  kndiscount gtmin 1 interpolate
     0    0  kndiscount gtmin 1
```

As can be seen, the unigram achieves a perplexity of 279, while the introduction of the previous word as a parent results in 175, and the previous two words achieves 173 (our baseline for this section). It appears

that the addition of the additional parent does not yield a significant additional amount of information.

Next we produce results for the distance-2 bigram and context-reversed trigram in order to verify this assumption.

```
## distance 2 bigram LM
## 0 zeroprobs, logprob= -94994.7 ppl= 309.005 ppl1= 901.399
W : 1 W(-2) word_2gram.count.gz word_2gram.lm.gz 2
    W2   W2  kndiscount gtmin 1 interpolate
     0    0  kndiscount gtmin 1

## distance 2 bigram LM
## 0 zeroprobs, logprob= -93324.9 ppl= 279.382 ppl1= 799.793
W : 1 W(-2) word_2gram.count.gz word_2gram.lm.gz 2
    W2   W2  kndiscount gtmin 1 interpolate
     0    0  gtmin 1

## context-reversed trigram LM
## 0 zeroprobs, logprob= -93040.3 ppl= 274.623 ppl1= 783.652
W : 2 W(-1) W(-2) word_3gram.count.gz word_3gram.lm.gz 3
 W1,W2   W1  kndiscount gtmin 2 interpolate
    W2   W2  kndiscount gtmin 1 interpolate
     0    0  kndiscount gtmin 1
```

Note that these models utilize the distance-2 counts The FLM above corresponds to case where the probability model $P(W_t = w_t|W_{t-2} = w_{t-2})$, so the previous word is entirely not present. This means that the counts for this table correspond to the quantity $N(w_t, w_{t-2})$. The verbatim section shows two models, one where the uni-gram node uses Kneser-Ney discounting (the first one) and one where that is not the case. As can be seen, neither of these perplexities (309 and 279) beat even the case of the unigram language model above (perplexity of 279). This does not mean, however, that the parent $W(-2)$ is not informative — it means only that the additional information that it provides is not sufficient to offset the loss in estimation quality. The third example shows the context-reversed trigram, where the model $p(W_t|W_{t-1}, W_{t-2})$ backs off first to $p(W_t|W_{t-2})$ and then to $p(W_t)$. This example corresponds to a different backoff-path in the backoff graph. Again, we see that while this is a bit better than the distance-2 bigram, by choosing the wrong backoff parent order (in this case, reverse time), perplexity can increase.

This last example shows how crucial it is to obtain a language model and use a set of parent random variables that balance predictability while at the same time do not decrease estimation accuracy. Predictability can be improved for example by choosing parents that have a high degree of mutual information with a child. If a given parent has a large number of values, however, estimation quality can decrease because

the amount of training data for each value of the parent variable is reduced. The balance must of course be seen in the context of language-model backoff, since backoff is an attempt to utilize high predictability in a context where estimation quality is high, but to backoff to a lower-order model (where, essentially, different sets of parent values are pooled together) in order to improve estimation quality.

Next, we present results where the word depends on a number of different factors at the same time in order to determine which factors, by themselves yield the most information about a word.

```
## word given M
## 0 zeroprobs, logprob= -39289.7 ppl= 10.7114 ppl1= 16.6782
W : 1 M(0) wgm.count.gz wgm.lm.gz 2
    M0   M0  kndiscount gtmin 1 interpolate
    0    0  gtmin 1

## word given S
## 0 zeroprobs, logprob= -21739.2 ppl= 3.71382 ppl1= 4.74486
W : 1 S(0) wgs.count.gz wgs.lm.gz 2
    S0   S0  kndiscount gtmin 1 interpolate
    0    0  gtmin 1

## word given R
## 0 zeroprobs, logprob= -31967.6 ppl= 6.88531 ppl1= 9.87163
W : 1 R(0) wgr.count.gz wgr.lm.gz 2
    R0   R0  kndiscount gtmin 1 interpolate
    0    0  gtmin 1

## word given P
## 0 zeroprobs, logprob= -38468.2 ppl= 10.1933 ppl1= 15.7253
W : 1 P(0) wgp.count.gz wgp.lm.gz 2
    P0   P0  kndiscount gtmin 1 interpolate
    0    0  gtmin 1
```

As can be seen, all of the perplexities are quite low, meaning that the factors for a word if known at the same time as the word are quite informative. We can see that the stem $P(W_t|S_t)$ is the most predictive of the word, with a perplexity of 3.7. This is followed by the root $P(W_t|R_t)$ (perplexity 6.9), pattern $P(W_t|P_t)$ (perplexity 10.2), and the morphological class $P(W_t|M_t)$ (perplexity 10.7).

Note that even though these perplexities are low, to use this during decoding in a language model, it is necessary also to obtain probability models for the corresponding parent. For example, if the $P(W_t|S_t)$ model is used in decoding, it is necessary to utilize a model $P(S_t|\text{parents})$.

In any case, it is also possible to combine these as parents as the following shows (note that in this case Witten-Bell smoothing was used as it was found to be better).

```
## word given stem morph,
## 0 zeroprobs, logprob= -10497.6 ppl= 1.88434 ppl1= 2.12099
W : 2 S(0) M(0) wgsm1.count.gz wgsm1.lm.gz 3
  S0,M0 M0 wbdiscount gtmin 1 interpolate
  S0    S0 wbdiscount gtmin 1
  0     0  wbdiscount gtmin 1

## word given stem morph,
## 0 zeroprobs, logprob= -11571.3 ppl= 2.01049 ppl1= 2.29054
W : 2 S(0) M(0) wgsm2.count.gz wgsm2.lm.gz 3
  S0,M0 S0 wbdiscount gtmin 1 interpolate
  M0    M0 wbdiscount gtmin 1
  0     0  wbdiscount gtmin 1

## word given stem root,
## 0 zeroprobs, logprob= -17954.1 ppl= 2.95534 ppl1= 3.61812
W : 2 S(0) R(0) wgsr1.count.gz wgsr1.lm.gz 3
  S0,R0 R0 wbdiscount gtmin 1 interpolate
  S0    S0 wbdiscount gtmin 1
  0     0  wbdiscount gtmin 1

## word given stem root,
## 0 zeroprobs, logprob= -18861.2 ppl= 3.12163 ppl1= 3.86099
W : 2 S(0) R(0) wgsr2.count.gz wgsr2.lm.gz 3
  S0,R0 S0 wbdiscount gtmin 1 interpolate
  R0    R0 wbdiscount gtmin 1
  0     0  wbdiscount gtmin 1
```

As can be seen, the best example uses as parents both the stem and the morph, parents that do *not* correspond to the best two individual parents in the previous example (stem and root). This example therefore indicates that the two best parents can be redundant with respect to each other. This example also shows the effect of different backoff paths. The first stem-morph model first chooses to drop the morph and then the stem (perplexity of 1.88), and the second stem-morph example drops first the stem and then the morph (perplexity of 2.01). Again, different backoff orders can lead to differences in perplexity, which can at times be quite dramatic.

Next, we investigate a number of different models for the current morph and stem (i.e., $P(M_t|\text{parents})$ and $P(S_t|\text{parents})$ where parents come from the past) to see if they might lead to low perplexity when combined with the models above for $P(W_t|S_t, M_t)$. The first set utilize the previous two words along with possibly the other corresponding factors.

```
## morph given word word
## 0 zeroprobs, logprob= -73287 ppl= 83.3629 ppl1= 190.404
M : 2 W(-1) W(-2) m_g_w1w2.count.gz m_g_w1w2.lm.gz 3
  W1,W2  W2 kndiscount gtmin 1 interpolate
  W1     W1 kndiscount gtmin 1 interpolate
  0      0  kndiscount gtmin 1

## morph given stem word word
## 0 zeroprobs, logprob= -25741.1 ppl= 4.72842 ppl1= 6.31983
M : 3 S(0) W(-1) W(-2) m_g_s1w1w2.count.gz m_g_s1w1w2.lm.gz 4
  S0,W1,W2  W2 kndiscount gtmin 1 interpolate
  S0,W1     W1 kndiscount gtmin 1 interpolate
  S0        S0 kndiscount gtmin 1
  0         0  kndiscount gtmin 1

## stem given word word
## 0 zeroprobs, logprob= -87686.6 ppl= 198.796 ppl1= 534.061
S : 2 W(-1) W(-2) s_g_w1w2.count.gz s_g_w1w2.lm.gz 3
  W1,W2  W2 kndiscount gtmin 1 interpolate
  W1     W1 kndiscount gtmin 1 interpolate
  0      0  kndiscount gtmin 1

## stem given morph word word
## 0 zeroprobs, logprob= -39275.6 ppl= 10.7023 ppl1= 16.6614
S : 3 M(0) W(-1) W(-2) s_g_m0w1w2.count.gz s_g_m0w1w2.lm.gz 4
  M0,W1,W2  W2 kndiscount gtmin 1 interpolate
  M0,W1     W1 kndiscount gtmin 1 interpolate
  M0        M0 kndiscount gtmin 1
  0         0  kndiscount gtmin 1
```

The first model is for $P(M_t|W_{t-1}, W_{t-2})$ and shows a perplexity of 83. The second model adds an additional parent $P(M_t|S_t, W_{t-1}, W_{t-2})$ reducing the perplexity to 4.7. While this might sound encouraging, this latter model could be used with the model for $P(S_t|W_{t-1}, W_{t-2})$ with a perplexity of 198.8. Therefore, these last

two models could be used with $P(W_t|S_t, M_t)$ to yield a total perplexity of $4.7 \times 198.8 \times 1.88 = 1756.6$, a value *much* higher than the baseline trigram perplexity of 172.9. Alternatively, the $P(M_t|W_{t-1}, W_{t-2})$ model could be used with the $P(S_t|M_t, W_{t-1}, W_{t-2})$ model leading to $4.7 \times 83 \times 10.7 = 4174.1$ (a graph for this model is shown in Figure 7). Apparently there is a perplexity cost to split the word into factors as such and model each factor separately (possibly due to the fact that multiple smoothings are occurring). Note that this is exactly a form of a class-based language model, or perhaps more precisely a factored-class-based language model since the word class (M_t, S_t) is represented in factored form as is class model.

Next, we provide perplexities for word-based language models where the factors additional to words can be used as parents. We present a number of models in this category, leading up to a bigram model that ultimately will improve on our baseline.

```
## word given word-1 word-2 morph-1 morph-2
## 0 zeroprobs, logprob= -85748.4 ppl= 176.85 ppl1= 464.838
W : 4  W(-1) W(-2) M(-1) M(-2) w_g4_w1w2m1m2.count.gz s_g4_w1w2m1m2.lm.gz 5
 0b1111 0b0010 kndiscount gtmin 4 interpolate
 0b1101 0b1000 kndiscount gtmin 3 interpolate
 0b0101 0b0001 kndiscount gtmin 2 interpolate
 0b0100 0b0100 kndiscount gtmin 1 interpolate
 0b0000 0b0000 kndiscount gtmin 1

## word given word-1 word-2 stem-1 stem-2
## 0 zeroprobs, logprob= -85522.1 ppl= 174.452 ppl1= 457.366
W : 4  W(-1) W(-2) S(-1) S(-2) w_g4_w1w2s1s2.count.gz s_g4_w1w2s1s2.lm.gz 5
 0b1111 0b0010 kndiscount gtmin 4 interpolate
 0b1101 0b1000 kndiscount gtmin 3 interpolate
 0b0101 0b0001 kndiscount gtmin 2 interpolate
 0b0100 0b0100 kndiscount gtmin 1 interpolate
 0b0000 0b0000 kndiscount gtmin 1

## word given word-1 word-2 morph-1 stem_2
## 0 zeroprobs, logprob= -85732.8 ppl= 176.684 ppl1= 464.319
W : 4  W(-1) W(-2) M(-1) S(-2) w_g4_w1w2m1s2.count.gz s_g4_w1w2m1s2.lm.gz 5
 0b1111 0b0010 kndiscount gtmin 4 interpolate
 0b1101 0b1000 kndiscount gtmin 3 interpolate
 0b0101 0b0001 kndiscount gtmin 2 interpolate
 0b0100 0b0100 kndiscount gtmin 1 interpolate
 0b0000 0b0000 kndiscount gtmin 1
```

```
## word given word-1 word-2 stem-1 morph-2
## 0 zeroprobs, logprob= -85543 ppl= 174.672 ppl1= 458.051
W : 4  W(-1) W(-2) S(-1) M(-2) w_g4_w1w2s1m2.count.gz s_g4_w1w2s1m2.lm.gz 5
 0b1111 0b0010 kndiscount gtmin 4 interpolate
 0b1101 0b1000 kndiscount gtmin 3 interpolate
 0b0101 0b0001 kndiscount gtmin 2 interpolate
 0b0100 0b0100 kndiscount gtmin 1 interpolate
 0b0000 0b0000 kndiscount gtmin 1
```

Note that in this set, the parent node identifiers and backoff selectors are specified using binary strings rather than a comma-separated list of parents (see Section 5.4.3). In all cases, however, the additional parents actually *increase* the perplexity relative to the baseline.

In general, the following question might be asked at this point. Why should one include additional parents to a model when the parents are a deterministic function of pre-existing parents? For example, the stem is a deterministic function of the word (see Section 4.5.1), which means that $P(W_t|W_{t-1}, S_{t-1}) = P(W_t|W_{t-1})$, or that the current word is conditionally independent of the previous stem given the previous word. The answer, however, is that we are not using the true probability distributions. Instead, we are used models that have been estimated using the backoff method. There might be a word context that did not exist in the training data, but the corresponding stems for those words that did not occur as a context in training might exist in training data via other words that had the same stems. In other words, suppose the context w_{t-1}, w_{t-2} in $P(w_t|w_{t-1}, w_{t-2})$ did not occur in training data but w'_{t-1}, w'_{t-2} as a context for w_t did occur in training data. Also suppose that both w_{t-1}, w_{t-2} and w'_{t-1}, w'_{t-2} have the same corresponding stems s_{t-1}, s_{t-2}. Then it would be the case that while $P(w_t|w_{t-1}, w_{t-2})$ would need to backoff to the unigram, $P(w_t|w_{t-1}, w_{t-2}, s_{t-1}, s_{t-2})$ would backoff only to $P(w_t|s_{t-1}, s_{t-2})$ presumably providing a better prediction for w_t than only the unigram model. The argument can be seen as a standard bias-variance tradeoff in statistics – we add bias to the model in the form of backoff in order to reduce the variance of the estimate. We will try to exploit this fact below.

Consider the following model.

```
##  logprob= -85087.9 ppl= 169.939 ppl1= 443.359
W : 6 W(-1) M(-1) S(-1) W(-2) M(-2) S(-2) dev.count.gz dev.lm.gz 7
  W1,M1,S1,W2,M2,S2  W2 kndiscount gtmin 2 interpolate
  W1,M1,S1,M2,S2     S2 kndiscount gtmin 40000000 interpolate
  W1,M1,S1,M2        S1 kndiscount gtmin 40000000 interpolate
  W1,M1,M2           M2 kndiscount gtmin 40000000 interpolate
  W1,M1              W1 kndiscount gtmin 1 interpolate kn-count-parent 0b111111
```

```
  M1                  M1 kndiscount gtmin 3
  0                   0  kndiscount gtmin 1 kn-count-parent W1,M1
```

This model can be seen to improve on our baseline perplexity by a small amount. The model first attempts to utilize the entire context — since both stems and morphs are deterministic functions of words, the context `W1,M1,S1,W2,M2,S2` is identical to that of `W1,W2`. If that context does not exist in the training data (with a count of 2 or more), we drop the parents `W2`, `W2`, and `S1` (via very large min-counts as described in Section 5.4.3) and "land" on node `W1,M1`. This node is attempted and used if a hit occurs, and otherwise the previous word is dropped, moving down until the unigram.

The next example improves on this as follows.

```
## logprob= -84924.5 ppl= 168.271 ppl1= 438.201
W : 6 W(-1) W(-2) M(-1) S(-1) M(-2) S(-2) dev.count.gz dev.lm.gz 9
  W1,W2,M1,S1,M2,S2  W2 kndiscount gtmin 3 interpolate
  W1,M1,S1,M2,S2     S2,M2 kndiscount gtmin 10000000 combine mean
  W1,M1,S1,M2        M2 kndiscount gtmin 4 kn-count-parent 0b111111
  W1,M1,S1,S2        S2 kndiscount gtmin 2 kn-count-parent 0b111111
  W1,M1,S1  W1 kndiscount gtmin 2 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 combine mean
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

This example first uses the complete context, and if a hit does not occur drops the parent `W2`. The next node `W1,M1,S1,M2,S2` uses generalized backoff to take a mean between the lower nodes where the parents `S2` or `M2` have been dropped. Each of these nodes are tried, and they respectively drop either `M2` or `S2`, meaning they both backoff to the same model `W1,M1,S1`. This node is attempted, and if a hit does not occur parent `W1` is dropped. Then at node `M1,S1` generalized backoff is used again backing off to the mean of model `M1` or `S1`, and then finally ending up at the unigram. The perplexity achieved is 168.

The next trigram improves upon this even further.

```
## logprob= -84771.4 ppl= 166.724 ppl1= 433.423
W : 6 W(-1) W(-2) M(-1) S(-1) M(-2) S(-2) dev.count.gz dev.lm.gz 9
  W1,W2,M1,S1,M2,S2  W2 kndiscount gtmin 3 interpolate
  W1,M1,S1,M2,S2     S2,M2 kndiscount gtmin 10000000 combine mean
  W1,M1,S1,M2        M2 kndiscount gtmin 4 kn-count-parent 0b111111
  W1,M1,S1,S2        S2 kndiscount gtmin 2 kn-count-parent 0b111111
  W1,M1,S1  W1 kndiscount gtmin 2 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 combine max strategy bog_node_prob
```

```
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

This example is identical to the one above, except that rather than using mean combination at node `M1,S1`, max combination is used. As can be seen the resulting perplexity achieved is 166.7.

This last example improves on this a bit more, but with a simpler model.

```
## Best perplexity found
## logprob= -84709 ppl= 166.097 ppl1= 431.488
W : 4 W(-1) W(-2) M(-1) S(-1)  w_g_w1w2m1s1.count.gz w_g_w1w2m1s1.lm.gz 6
  W1,W2,M1,S1  W2 kndiscount gtmin 3 interpolate
  W1,M1,S1  W1 kndiscount gtmin 2 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 combine max strategy bog_node_prob
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

In this case, we use a trigram but only add additional parents `M1` and `S1`. The entire context is first attempted, removing `W2` and then `W1` if hits do not occur. Then generalized backoff inspired from the above is used to finish off down to the unigram. The perplexity achieved is 166, the best perplexity we achieved on this data with any model.

While we expected that it would be possible to produce significant additional perplexity reductions with these trigram like models (trigram because they depend on nothing farther in the past than the two previous words or deterministic functions of these two previous words), the model search space is very large. Due to the time limitations of the 6-week workshop, we decided to investigate bigrams for the remainder of the time. As you recall, the baseline bigram had a perplexity of 175, much higher than the 166.7 achieved above and certainly higher than the baseline trigram (173).

The next example is our first bigram that beats the baseline trigram perplexity, achieving 171. It does this by conditioning on both the previous morph class and stem in addition to the word (so the model is for $P(W_t|W_{t-1}, M_{t-1}, S_{t-1})$, but also uses one of the generalized backoff methods, `bog_node_prob` combining using the maximum. It first attempts to use a context consisting of the previous word (`W1,M1,S1` which is equivalent to `W1` since `M1,S1` are deterministic functions of `W1`). If there is no language-model hit at that level, it jumps directly down to either using a context consisting of either only `M1` or `S1`. It does this "jump", however, via a node that takes the maximum probability of these two contexts, via the large `gtmin` parameter (so the node itself does not ever contribute a real probability, it just acts to combine lower-level backoff-graph nodes). Lastly, notice that `kn-count-parent` was used to specify that the original node should be used to form the Kneser-Ney meta-counts (the default would have been to use node `M1,S1`. It

was found in general that specifying a `kn-count-parent` node that included the word variable had a beneficial effect on perplexity, when the model was a word model (i.e., when W_t was on the left of the conditioning bar in $P(W_t|\text{parents})$).

```
##  logprob= -85204.8 ppl= 171.142 ppl1= 447.086
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 strategy bog_node_prob combine max
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

The next example shows an even simpler bigram that also improves upon the baseline bigram and almost matches the baseline trigram. In this case, generalized backoff is not used. Rather, the word context is augmented with the morph variable.

```
## bigram that gets as good as a trigram.
##  logprob= -85379.6 ppl= 172.958 ppl1= 452.721
W : 2 W(-1) M(-1)  w_g_w1m1.count.gz w_g_w1m1.lm.gz 3
  W1,M1  W1 kndiscount gtmin 1 interpolate
  M1     M1 kndiscount gtmin 3
  0      0  kndiscount gtmin 1 kn-count-parent W1,M1
```

The next several examples show that perplexity can depend on adjusting the `gtmin` parameter. In this case, changing `gtmin` for nodes `M1`, `S1`, and for no-parents might lead either to perplexity increases or decreases as the example shows.

```
## logprob= -85140 ppl= 170.474 ppl1= 445.015
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 strategy bog_node_prob combine max
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 2 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1


##  logprob= -85296 ppl= 172.087 ppl1= 450.016
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 strategy bog_node_prob combine max
```

```
  M1        M1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1

##  logprob= -85267.8 ppl= 171.794 ppl1= 449.109
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 strategy bog_node_prob combine max
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1 interpolate
  S1        S1 kndiscount gtmin 2 kn-count-parent W1,M1,S1 interpolate
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1

##  logprob= -85132.5 ppl= 170.396 ppl1= 444.776
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 strategy bog_node_prob combine max
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

The next example next shows that by using a different combination method (the mean in this case) it is possible to retain the perplexity improvement. This is particularly relevant because the mean combination rule does not lead to a costly language model to evaluate as does some of the other combination procedures.

```
## logprob= -85129.9 ppl= 170.37 ppl1= 444.693
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 strategy bog_node_prob combine mean
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

The following example shows that using a weighted mean, it is possible to achieve still better perplexity. In this case, most of the weight is given to the `M1` node rather than to the `S1` node.

```
## logprob= -85066.8 ppl= 169.723 ppl1= 442.691
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 strategy bog_node_prob \
```

```
                          combine wmean M1 7 S1 3
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

Changing the strategy to `counts_prod_card_norm` does not seem to show any appreciable change.

```
##   logprob= -85187 ppl= 170.959 ppl1= 446.518
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 \
                          strategy counts_prod_card_norm combine max
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

Changing the strategy to `counts_sum_counts_norm` and combining method to `gmean` leads to a slight increase in perplexity.

```
## logprob= -85252.5 ppl= 171.635 ppl1= 448.616
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 \
                      strategy counts_sum_counts_norm combine gmean
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1

## logprob= -85156.7 ppl= 170.646 ppl1= 445.548
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 strategy counts_sum_counts_norm \
                                              combine mean
  M1        M1 kndiscount gtmin 2 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1

##  logprob= -85150.3 ppl= 170.58 ppl1= 445.346
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
```

```
  W1,M1,S1  W1 kndiscount gtmin 1 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 strategy counts_sum_counts_norm \
                                                combine mean
  M1        M1 kndiscount gtmin 4 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

Using a weighted mean with `counts_sum_counts_norm` improves things a bit further.

```
## logprob= -84948.9 ppl= 168.519 ppl1= 438.966
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 2 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 strategy counts_sum_counts_norm \
                                                  combine wmean M1 7 S1 3
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

Using the mean, however, produces a slight modification.

```
## logprob= -84967 ppl= 168.703 ppl1= 439.536
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 2 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 combine mean
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

The last example shows our best bigram, achieving a perplexity of 167.4, which is not only better than the baseline trigram, but is close to the much more expensive generalized-backoff trigram perplexity seen above.

```
##  logprob= -84845.2 ppl= 167.468 ppl1= 435.719
W : 3 W(-1) M(-1) S(-1)  dev.count.gz dev.lm.gz 5
  W1,M1,S1  W1 kndiscount gtmin 2 interpolate
  M1,S1     S1,M1 kndiscount gtmin 100000000 combine max strategy bog_node_prob
  M1        M1 kndiscount gtmin 3 kn-count-parent W1,M1,S1
  S1        S1 kndiscount gtmin 1 kn-count-parent W1,M1,S1
  0         0  kndiscount gtmin 1 kn-count-parent W1,M1,S1
```

This model increases `gtmin` to 2 for the `W1,M1,S1` node yielding the further improvements.

As can be seen, the space of possible models is quite large, and searching through such a model space by hand can be quite time consuming, and this only shows perplexity improvements. Ideally in these experiments we would perform word-error experiments as well. Even so, a bigram model that achieves a perplexity lower than a trigram should have beneficial word-error effects for first-pass decoding, and since it is a bigram (rather than a trigram), the search space in such a decoding is not exorbitantly large (relative to a trigram). Therefore, decoder pruning thresholds and beam-widths can be more forgiving than if the model was a trigram, and word errors would likely be improved. Moreover, we expect that automatic data-driven structure learning procedures to produce these models might yield improvements over the models above (all of which were derived and specified by-hand).

```
## word given word word morph morph stem stem model, all backoff paths.
## gtmin set to the number of bits set in the node. This model will
## take about 2 days to train on a typical machine.
W : 6 W(-1) W(-2) S(-1) S(-2) M(-1) M(-2) w_g_w1w2s1s2m1m2.count.gz w_g_w1w2s1s2m1m2.lm.gz 64
 0x3F 0xFF gtmin 6 kndiscount strategy bog_node_prob combine max
 0x3E 0xFF gtmin 5 kndiscount strategy bog_node_prob combine max
 0x3D 0xFF gtmin 5 kndiscount strategy bog_node_prob combine max
 0x3C 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x3B 0xFF gtmin 5 kndiscount strategy bog_node_prob combine max
 0x3A 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x39 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x38 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x37 0xFF gtmin 5 kndiscount strategy bog_node_prob combine max
 0x36 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x35 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x34 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x33 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x32 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x31 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x30 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x2F 0xFF gtmin 5 kndiscount strategy bog_node_prob combine max
 0x2E 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x2D 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x2C 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x2B 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x2A 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x29 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x28 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x27 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x26 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x25 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x24 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x23 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x22 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x21 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x20 0xFF gtmin 1 kndiscount strategy bog_node_prob combine max
 0x1F 0xFF gtmin 5 kndiscount strategy bog_node_prob combine max
 0x1E 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x1D 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x1C 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x1B 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x1A 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x19 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x18 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x17 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x16 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x15 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x14 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x13 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x12 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x11 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x10 0xFF gtmin 1 kndiscount strategy bog_node_prob combine max
 0x0F 0xFF gtmin 4 kndiscount strategy bog_node_prob combine max
 0x0E 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x0D 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x0C 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x0B 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x0A 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x09 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x08 0xFF gtmin 1 kndiscount strategy bog_node_prob combine max
 0x07 0xFF gtmin 3 kndiscount strategy bog_node_prob combine max
 0x06 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x05 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x04 0xFF gtmin 1 kndiscount strategy bog_node_prob combine max
 0x03 0xFF gtmin 2 kndiscount strategy bog_node_prob combine max
 0x02 0xFF gtmin 1 kndiscount strategy bog_node_prob combine max
 0x01 0xFF gtmin 1 kndiscount strategy bog_node_prob combine max
 0x00 0xFF gtmin 0 kndiscount strategy bog_node_prob combine max
```

Figure 13: A large FLM model for W with six parents, and all $2^6 = 64$ backoff nodes are utilized.

6 Data-driven Morphological Knowledge: Knowledge-free Induction of Arabic Morphology

It has been shown in the previous sections that speech recognition performance can be improved when one takes advantage of morphology. A question that arises is whether it is actually necessary to obtain that morphology from a human expert, or if it might somehow be possible to generate that morphology strictly by corpus-based, knowledge-free induction. If it were possible to get meaningful gains using purely induced morphology, then morphology induction could be a subcomponent of speech recognition across a wide range of languages. A number of efforts have been explored in the past for performing morphology induction. Some of these efforts have taken advantage of knowledge sources as a means of bootstrapping performance [29, 54]. Others, though knowledge-free, have focused primarily on identifying common affixes [10, 21, 29]. Goldsmith [32] and Schone and Jurafsky [48, 46] showed that it is possible to induce complete morphologies for inflectional languages using corpus-based techniques alone. Goldsmith's technique was applied to inflectional languages and made use of word orthography and minimum description length in order to find the most concise representation for words in terms of stems and suffixes. Schone and Jurafksy generated morphologies, again, for inflectional languages – using a different, semantics-based method for inflectional languages. Since our technique heavily relies upon that of Schone and Jurafsky, we describe it in additional detail. Schone and Jurafsky's approach began with a search for possible inflections which used a trie-based algorithm for finding words that shared common beginnings and endings (which we will here call "stems"). If two words shared a common stem and a high-frequency affix, the words were treated as pairs of potential morphological variants ("PPMVs"). Schone and Jurafsky then used Latent Semantic Analysis [20] as a means of identifying PPMVs which were deemed to be strongly related semantically. Latent Semantic Analysis is a knowledge-free algorithm for inducing pseudo-semantic relationships between words. Semantically-related PPMVs were treated as true morphological variants. Any word pairs that were not related because of their semantics were then further scrutinized by a sequence of steps involving orthographic information, local syntax, and transitivity. Pairs of words that satisfied criteria in at least one of these three additional processes were also added to the list of "true" morphological variants. Schone and Jurafsky applied their technique to large document sets (i.e., over two million words) from English, German, and Dutch, and they focused specifically on words that occurred at least 10 times in their respective corpora. Lastly, they scored their derived word conflation sets against those observed in the human-labeled CELEX corpus. A conflation set for a word w is the set of all words that are morphological variants of w (including w itself). For an English example, the conflation set for the word *do* would be *do, did, done, doing, undo, undid, undoing, undone, redo*, etc. If we let X_w be the conflation set that the automatic algorithm generates, and if Y_w is that of CELEX, Schone and Jurafsky indicated the number of correct (C), insertions (I), and

deletions (D) as

$$C = \sum_w \frac{|X_w \cap Y_w|}{|Y_w|} \tag{4}$$

$$D = \sum_w \frac{|Y_w - (X_w \cap Y_w)|}{|Y_w|} \tag{5}$$

$$I = \sum_w \frac{|X_w - (X_w \cap Y_w)|}{|Y_w|} \tag{6}$$

where $|X|$ denotes cardinality of the set X. From these values, one can compute precision, recall, F-score, or error rate. Precision, P, is defined to be $\frac{C}{(C+I)}$. Recall, R, is $\frac{C}{(C+D)}$. F-Score is defined to be $\frac{2PR}{(P+R)}$ and error rate is $\frac{100*(D+I)}{(C+D)}$. Schone and Jurafsky reported having obtained F-scores for finding suffixing relations which ranged between 85% and 92%. When they scored full suffix/prefix/circumfixing relationships, their performance was obviously degraded.

6.1 Arabic Morphology Induction

Establishing Baselines

As it has been mentioned previously, Arabic morphology consists of both inflectional and templatic components. If one desired to induce morphology for Arabic, it would seem that a reasonable approach would be to start with the Schone/Jurafsky technique and proceed from there. It is clear that this strategy will fall short of producing a complete morphological analysis, but the degree to which it falls short depends on the amount of templatic behavior in Arabic. We therefore start essentially with the Schone/Jurafsky technique(S/J) to provide a baseline for our morphology induction system.

Preliminaries

Three things should be born in mind with regard to obtaining a baseline and scoring it. The first is that Schone and Jurafsky focused exclusively on words with frequencies of at least ten. They were motivated by a desire to induce an electronic dictionary which contained the more meaningful words from a large data set. However, for the task of improving speech recognition, the low-frequency words are of prime importance. Hence, we need to generate morphological analyses for all words, not just for the higher-frequency words. Induced semantics of low-frequency words are not very robust, so we also need to perform some smoothing of low-frequency semantic relationships. (We will not report results on smoothing, but when we use it, we observed definite performance improvements for low-frequency events.)

The second issue to bear in mind is that Schone and Jurafsky used data sets containing at least two million words for their induction. We only have about 180,000 in the CallHome training set. This, then, will also result in fewer important affixes being found and, again, in less robust induced semantics.

The last notion which needs to be considered is that no performance measures can be obtained without a gold standard. The CELEX lexicon that Schone and Jurafsky made use of only contains annotations

Exp #	Algorithm	Words with Freq = 10			All words		
	Suf	Pref	Gen	Suf	Pref	Gen	Suf
1	S/J Semantics alone	20.9	11.7	39.8	29.7	20.6	60.7
2	Semantics + Freq Info	19.2	11.5	39.0	27.6	16.8	57.6

Table 26: Error Rates (%) for (Smoothed) Schone-Jurafsky Algorithm Applied to CH Arabic

for German, Dutch, and English. Therefore, we need to either build our own lexicon from scratch or take advantage of some existing structure. The latter is obviously preferable. Almost every word from the Call-Home Egyptian Arabic corpus is represented in the corresponding pronunciation lexicon provided by LDC. However, not only are pronunciations available, but the root form and part of speech for each word are also provided. If a word w_1 is said to have a root w_2 which itself has a root form w_3, and if w_3 is its own root form, then (assuming root-form transitivity is allowable) we can say that the conflation set for w_1 is at least w_1, w_2, w_3. Any other words whose final root forms are also w_3 will likewise be contained in the gold standard conflation set for word w_1.

The main question is whether this root-form transitivity is allowed. This depends on whether or not a word can have multiple roots. In English, for example, a word can have multiple roots. The word *footings* is a plural noun whose root form is *footing*; but *footing* can also be a verb whose root form is *foot* (as in *to foot the bill*). However, *footings* is not a morphological variant of *foot*. Although we might expect that some words of Arabic could have multiple roots, a native speaker of Egyptian Arabic confirmed that root-form transitivity is mostly true for Arabic. Therefore we used the above strategy as our means for generating the gold standard lexicon.

Establishing the S/J baseline

Now that we have a gold standard, we can compute a baseline error rate. We apply the (smoothed) S/J algorithm to our data and observe performance under a number of conditions. First, we assess how well the S/J technique would do if it were being applied in the paradigm for which it was built: namely, working with words with frequencies of at least 10. We also test how well it works in the condition we are more interested in where we have no frequency-based thresholds. For both tests, we see how well the automatic systems finds conflations based only on suffixes (Suf); based only on prefixes (Pref); and based on all affixing and templatic relationships (Gen). In Table 26, we present these results in terms of error rates (as opposed to F-scores as in [48, 47]). These results were obtained by inducing morphology on the words from the original 80 CH Arabic training transcripts.

The first experiment here illustrates how well the S/J technique would work if we used only its semantic component. The second experiment also incorporates the S/J technique for making improvements based on orthographic frequency. We do not include the S/J syntactic or branching transitivity results since these are computationally expensive and result in very minor improvements (less than 0.3% F-score at best) on

Exp #	Algorithm	Words w Orig Freq = 10			All words from Train80		
		Suf	Pref	Gen	Suf	Pref	Gen
3	More data, plus S/J Semantics	20.3	12.5	39.6	27.6	16.7	56.9
4	Exp 3 + Freq Info	23.5	14.5	38.7	25.4	15.4	55.1

Table 27: Induction error rates for words from 80 training files given 100+20+20 data set.

other languages. We note from the table that if we were considering the suffixes or prefixes alone under the condition used by Schone and Jurafsky, performance would be fairly reasonable. Yet when we consider that we are interested in the general condition for all words, we see that performance is incredibly bad. This is largely due to the fact that the system is producing significant deletions by not finding all of the conflations for each of the Arabic words. This may be somewhat expected since 3576 words in Callhome Egyptian Arabic have over 50 words in their particular conflation sets. This means that even if the system accurately found 10 of the 50 morphological variants for each word (which seems fairly reasonable), it would still be introducing over 2800 errors. We must attempt, therefore, to capture these huge conflation sets without significantly damaging the precision of the other, shorter conflation sets.

Are problems with low-frequency words strictly due to data sparseness?

Clearly, we need to improve performance. Yet before discussing strategies for improvement, there is an issue which we should consider out of curiosity. We see in the table above that there is significant degradation in performance between the system when applied to lower-frequency words versus the higher-frequency ones. Is this due to the fact that the system had too few examples of the low frequency words in order to make its decisions, or is it some inherent property of the low-frequency words themselves? If we had some additional data where we could supplement the frequencies of all words but only score on the same words as above, this would help us determine if the problem was primarily due to data sparseness or to word peculiarity. Fortunately, we do have some additional data that we can make use of. The experiments from Table 26 were made using 80 ASR training files from CallHome. However, during the course of the workshop, LDC also delivered an additional 20 files. Additionally, since morphology induction is knowledge-free, all data sets that the algorithm analyzes are actually like evaluation sets; i.e., it makes no special use of training data. Hence, any development sets or other evaluation sets that have no need for being held out for ASR can therefore be fairly used for morphology induction. Therefore, we score performance of the system on only the words evaluated in Table 26 (which set we will call "Train80") but we actually induce morphology for all words from a larger corpus consisting of the 100 training conversations, the 20 development conversations, and the 20 from the 1997 evaluation set. (We will call this larger set the "100+20+20" data set). Table 27 shows the results of using this additional data.

From Tables 26 and 27, we observe that the data sparsity is definitely a problem since the incorporation of the new data reduced the error rate of the all-words/general category by an absolute 2.5%. Additional

	-	k	a	t	a	b
-	0	1	2	3	4	5
m	1	2	3	4	5	6
a	2	3	2	3	4	5
k	3	2	3	4	5	6
t	4	3	4	3	4	5
a	5	4	3	4	3	4
b	6	5	4	5	4	3

Table 28: Minimum edit distance between *katab* and *maktab*.

data also helped induction of morphology in the high-frequency condition as well, but performance only improves by 0.3%. We should also note the interesting phenomenon that suffixes and prefixes were induced less well for the high-frequency set than they were before, but again, the full set of words from Train80 have better induction performance overall. For the remainder of this discussion, we treat the results of Table 27 as the baseline, and we will apply the same paradigm of evaluating against Train80 data while performing induction on the "100+20+20" data set.

6.1.1 IR-based MED: Expanding the Search for Potential Morphological Variants

A first obvious path for trying to improve performance is to allow our system to find more potential variants at the onset of processing. To do this, we must have an algorithm which considers more options than tries would. For purely inflectional languages, one would expect that tries would be sufficient since it is almost always true that variants share a common contiguous substring which can then be captured by the trie structure. Since Arabic has templatic morphology, this is not always true. For example, katab and maktab are related morphologically, but they share the characters *k-t-b*, which are not contiguous. An option, then, is to look for morphological variants by using minimum edit distance (MED) instead of tries.

Application of MED

MED tries to find an alignment between two words which results in the smallest penalty for insertions, substitutions, and deletions. For example, if we wanted to align the words *katab* and *maktab* and if insertions and deletions counted as a single edit and incorrect substitutions counted as two, we would get a table such as that Table 28.

For any given entry (r, c) of the table, the numbers in (r, c) denote the number of substitutions, insertions, or deletions that one would have to make in order to convert from the first $(r-1)$ characters of *maktab* into the first $(c-1)$ characters of *katab*. In order to convert from the whole word *maktab* to *katab* therefore requires three edits which concurs with what we see in the lower right entry of the table. The arrows of the

table (identified by backtracking through the table along a least-cost path) indicate the appropriate transformation from *maktab* to *katab*. In particular, it suggests that the alignment is ma-k-tab and k-a-tab. Note that there are actually multiple paths that could result in a minimum edit. For example, m-a-k-tab and k-a-tab also has three edits. To overcome ambiguity, we favor edits which result in longer contiguous substrings. Moreover, though we still treat substitutions as a deletion and an insertion, we weight the insertions/deletions as the negative log probability of the character being inserted/deleted. This means, for example, that substitution of one vowel for another would be a less expensive substitution than, say, a *k* for a *b*. It is clear that this approach has a chance of doing a much better job at finding more potential variants of Arabic than the trie-based method did supposing that templating is a frequent phenomenon in conversational speech. As we see above, we can find a relationship between any two words regardless of whether the characters they have in common are contiguous or not. We claim that the closer the relationship is between two words, the more likely the two words are morphological variants. We also expect that any words that would have been found as potential variants using a trie-based approach will still be observed using MED, except now we will have more possible variants to choose from. Note that we can still apply the LSA-based evaluation that Schone and Jurafsky had performed between suspected variants in order to find pairs that are semantically related, so we expect the identification of additional benefit to us rather than hurt us.

Time improvements though use of IR

The ability to identify additional possible variants does not come without a cost, though. There is a cost in terms of time. The trie-based approach can rapidly identify words with common stems. However, in order for MED to find all PPMVs would require us to do $O(m^2N^2)$ operations, where N is the number of words in the dataset and m is the average word length. This number gets enormous. We need to effectuate a speed up.

The technique we use for doing the speed up is based on information retrieval (IR). In IR, one is faced with the task of scoring all documents which contain at least one word from among a certain set of query terms. Only the best-scoring results are reported to the user. To follow this same strategy, we therefore treat each word whose variants we want to find as a query, and each character of that word as a "query term". We order these "query terms" from least frequent to most frequent, and we stop adding query terms when it is clear that there will be no other "documents" containing those terms which will have scores in the top T of the final document list. Some additional speed improvements can also be made by considering n-grams (or skip n-grams), which we also did but do not expound on here. This significantly improves our ability to process the data. However, since we are only considering the top T potential variants, we may fail to find some valid relationships. Yet we accept this shortcoming in order to achieve a desirable processing speed.

Evaluation of IR/MED-based search

Now that we have an approach which should help us find templatic as well as inflectional relationships, we can compare our performance to that of the Schone/Jurafsky trie-based approach. Table 29 provides the results of using the MED paradigm as compared to the results we saw before using the S/J strategy.

Exp #	Algorithm	Words w/ Orig Freq = 10			All words from Train80		
		Suf	Pref	Gen	Suf	Pref	Gen
4	S/J: Semantics alone	20.3	12.5	39.6	27.6	16.7	56.9
5	MED: Semantics alone	19.5	13.0	39.8	27.2	17.5	57.2

Table 29: Error rates for incorporating minimum edit distance.

Much to our surprise, the MED strategy seems to be somewhat worse than the S/J method. Although this may be surprising, it could be expected. The S/J technique from before only analyzed words that share long, contiguous substrings whereas MED allow many more kinds of relationships. Some of these MED relationships will be spurious, so if semantics cannot reject those relationships, then insertions will result. Moreover, when we look at the most frequent relationships that the MED is finding, we see that the top rules are all affixing relationships! For example, given that a "*" represents some full word, say W, and *il+** indicates another word from the data of the form "il+W," then the first six MED-based rules, with their corresponding frequencies, are

$* \rightarrow il + *$	1178
$* \rightarrow *u$	635
$* \rightarrow *i$	455
$*i \rightarrow *u$	377
$* \rightarrow fa + *$	375
$* \rightarrow bi + *$	366

All of these would be found by algorithms that look for affixing relationships. This suggests that the although templatic morphology is a strong component of Arabic, affixation is at least as prevalent or more. This means that one might expect the S/J technique to do well. Nonetheless, there are other ways we can take advantage of the additional relationships identified through MED; in particular, we can do clustering thereon, which is the subject of the next section.

6.1.2 Replacing Frequency Strategy with Clustering

In experiment 2, we indicated performance of the S/J technique using orthographic information in addition to semantics. The particular experiment actually used the frequency-based technique from [47] as opposed to that of [46]. Schone reported that this updated frequency-based technique provided better results than the original routine. However, what was not stated was the fact that the technique is very slow ($O(N2)$) and it seeks to find words that the trie-based or semantic approach might have missed but which are true variants.

Since MED finds most of the important relationships between words, we could choose to use it in place of the frequency-based approach. One would have to wonder why MED should be used instead of frequencies. A particular reason is that IR-based MED will typically be faster than frequency-based processing. Another reason is that even though [47] indicates improvements when frequency-based processing is used, his approach only compares potential variants pair-by-pair instead of considering how one word compares to all others. The MED is, for the most part, doing the one-to-many comparison. We will therefore attempt to use it. One method for using MED is to do clustering on MED's output. In normal hierarchical agglomerative clustering, one is initially given P points in some high dimensional space and a measure or score is provided which can be used to indicate the similarity between any two. Each of the P points are then treated as trivial clusters and clustering begins. At each step of clustering, the two clusters x_1 and x_2 with the highest similarity are fused and the relationships between the new cluster and all other clusters are updated. Given that $|z|$ indicates the number of elements in cluster z, the updating of the similarity between cluster z and the new cluster x_1x_2 is typically chosen from among the following strategies:

1. nearest neighbor: $sim(z, x_1 1 x_2) = max(sim(z, x_1), sim(z, x_2))$
2. farthest neighbor: $sim(z, x_1x_2) = min(sim(z, x_1), sim(z, x_2))$
3. average neighbor: $sim(z, x_1x_2) = (|x_1|sim(z, x_1) + |x_2|sim(z, x_2))/|x_1x_2|$.

Average neighbor is the usual preferred strategy since it frequently provides the best clustering performance. We experimented with each strategy and confirmed this for our data.

Thus, we do average agglomerative clustering on our MED-provided word relationships. We say that the initial similarity between two words w_1 and w_2 is based on the frequency of the rule that maps w_1 to w_2 (call it $f_{rule(w_1,w_2)}$) as well as on the frequency of the residual that they have in common after the rule components have been removed.

It is easy for the MED-based process to determine the frequency of the rule. To count the residual, we ascertain the number of word pairs that have that residual and compute the square root as an estimate of the residual's frequency (which we will represent by $f_{paired-stems(w_1,w_2)}$. Using both of these frequency counts, we choose $sim(w1, w2)$ to be the product rule

$$\frac{(w1, w2) * (f - paired - stems(w1, w2))}{275} \tag{7}$$

Clustering begins by clustering words with the highest remaining similarity. We only consider words pairs where $f - rule(w_1, w_2)$ is three or more. Clustering ends when a threshold criterion is met which we defined to be one-half of the initial average similarity of the data set. The use of average similarity in the threshold is due to the fact that once we see similarities lower than it, we begin to think of them as more likely to be random. On the other hand, since we did not consider any $f - rule(w_1, w_2) = 2$ events in our data, we must drop the average to account for these events which were not originally counted.

Exp #	Algorithm	Words w/ Orig Freq = 10			All words from Train80		
		Suf	Pref	Gen	Suf	Pref	Gen
3	S/J: Semantics only	20.3	12.5	39.6	27.6	16.7	56.9
4	S/J: Semantics+Freq	23.5	14.5	38.7	25.4	15.4	55.1
5	MED: Semantics only	19.5	13.0	39.8	27.2	17.5	57.2
6	MED-based Clusters	17.2	11.8	36.6	24.8	15.9	55.5
7	MED Semantics w/ MED-based Clusters	16.2	10.8	35.8	23.7	14.3	54.5
8	S/J Semantics w/ MED-based Clusters	17.5	10.6	35.9	24.2	13.9	54.2

Table 30: Error rates for Clustering and Fusing

We now evaluate the performance of our system using agglomerative clustering. Table 30 (in experiment 6) illustrates the performance when clustering is used. As was mentioned before, agglomerative clustering on the orthographic information is a process very similar to the S/J frequency component; so in the first few rows of the table we have relisted the earlier experiments where we saw the frequency information displayed. We see from the table that clustering provides an improvement over MED-based semantics even though it did not take any semantics into account. It also provides a better result than S/J for inducing morphology on the high-frequency events, but it is slightly worse than S/J for the larger set of words in Train80.

Note that this clustering approach did not take any semantic information into account whereas the S/J frequency component had already used semantics. Clustering and semantics seem to provide quite distinct information, so one might hope that by fusing the results of these two efforts, additional improvements would be observed. Schone and Jurafsky had combined results using the “noisy-or” condition. Our clustering process does not generate probabilities, though, so we must fuse our data in a different manner. Hence, we will use the union of conflation sets as our means for combining MED and semantics. That is, if the conflation set for word W were given to be A,B,C using semantics and B,C,D,E using clustering, we will now say that its conflation set is A,B,C,D,E. Experiments 7 and 8 reflect the results of this unifying process. Whether we take our semantics from the MED-based process (experiment #7) or the S/J process (experiment #8), we get comparable results; but in either case, those results are better than without performing the union. In fact, these results are the best we have observed thus far.

6.1.3 Strengthening Transitivity

A last strategy for improving performance is to strengthen the S/J method for finding transitive relationships between words. Suppose we use the symbol $\sim$ to indicate a morphological variance relationship. In other

Exp #	Algorithm	Words w/ Orig Freq = 10			All words from Train80		
		Suf	Pref	Gen	Suf	Pref	Gen
4	Baseline	23.5	14.5	38.7	25.4	15.4	55.1
7	MED Semantics + MED Clusters	16.2	10.8	35.8	23.7	14.3	54.5
8	S/J Semantics + MED Clusters	17.5	10.6	35.9	24.2	13.9	54.2
9	Exp 7 + new transitivity	14.9	8.4	33.9	22.4	12.3	53.1
10	Exp 8 + new transitivity	16.4	8.4	33.6	23.3	12.3	52.7

Table 31: Error rates for incorporating transitivity.

words, if we say $X \sim Y$, we are stating that X is deemed to be a morphological variant of Y and vice versa. Suppose we induce the relationship $X \sim Y$ and $Y \sim Z$ and we want to know if $X \sim Z$. The Schone/Jurafsky technique made a determination for associating X and Z which was based strictly on the probabilities of the proposed relationships $X \sim Y$ and $Y \sim Z$. Although the use of probabilities for finding transitive relationships is a good practice, it has the limitation that it does not address the question of whether X and Z look alike, which one would hope would be the case. To get around this problem, Schone and Jurafsky claimed that X could not be a variant of Z if X was not originally in ZŠs conflation set. This strategy does provide some modest gains (about 0.1% F-score), but it severely limited the possible benefits of transitivity. We therefore propose a new method for imposing transitivity. In particular, we claim that if $X \sim Y$ and $Y \sim Z$, then $X \sim Z$ if the following two conditions hold: (1) X and Y share at least three bytes in common, and (2) the intersection of X and Y is contained in Z. The first of these restrictions may limit the set of languages where the search for transitivity is useful, but for inflectional and templatic languages, it is a reasonable constraint. The second constraint is valuable since it suggests that one would only treat X and Z as related if the characters they have in common are at least those that X has in common with Y.

When we evaluate transitivity performance (see Table 31), we note that it improved every aspect of our induction process. Transitivity, in conjunction with clustering and minimum edit distance, have collectively yielded a 4.5-5% absolute reduction in error in the whole system versus that which was seen in the original baseline. We will later apply these approaches to the English, German, and Dutch data akin to what Schone and Jurafsky had used to see if the process results in improvements. For the time being, though, we would now like to see if this induced morphology can be beneficial for reducing the word error rate of the ASR system.

6.2 Applying Morphology Induction to Language Modeling

There are several methods whereby one could incorporate induced morphology into language modeling. Earlier in this report, we described a method whereby one could build class-based language models and that

System	Word Error Rate
Baseline: base1 + base2	54.5%
Baseline + Induced Root	54.3%
Baseline + Induced Class	54.4%
Baseline + Induced Root + Class	54.4%
Previous best: Baseline + Actual Root + Class	53.8%

Table 32: Word error rates on Eval96 doing LM rescoring through induced morphology

one could use the development set as a means of finding optimization parameters for taking best advantage of those classes at evaluation time. We follow much the same practice here. Earlier experiments built classes from the stems, roots, rules, patterns, and part-of-speech classes that were provided from the CallHome lexicon. We do not have that same kind of information here, but we can use the conflation sets to provide approximations. If we state once again that X_w is the conflation set for word w, we define stems, roots, and so on as follows:

Stem: the smallest word, Z, where $Z \in X_w$ and $Z = W = Z$
Root: $Z_1(Z_2...Z_m)$, where $Z_i \in X_w$ for each $i \in [1, m]$
Rule: map(W=>Stem)
Pattern:map(Stem=>Root)
Class: map(W=>Root)

We need to be able to rescore eval96 using our choice of the elements above which we think will give the best results. It was shown earlier that using root forms and part-of-speech classes gave the best results when we had perfect data, so we try them again here. In Table 32, we see that by incorporating induced morphology, our gains are only very slight. The induced root and class were the sources of improvement, but there was no gain obtained from coupling the two and neither came close to achieving the performance that was seen before by using human-provided morphology and part of speech information. (However, it would have been valuable to see how well the use of induced roots did in comparison to the use of human-provided root forms.) This, of course, is a disappointing result, because a significant effort in induction contributed very little to word error rate reduction. Nonetheless, the effort that went into morphology induction was not wasted in that it improved the state of the art. Additionally, there are other places where morphology induction could help ASR besides just in language model rescoring. We discuss one such possibility in the next section.

Pseudo-word Induction Method	# Pseudos	Proposed Coverage	Number observed as words in data
Rule only	993398	41.3%	0.1%
Rule+1-char stem agreement	98864	25.0%	1.1%
Rule+2-char stem agreement	35092	14.9%	1.8%

Table 33: Performance measurements for pseudo-word induction.

6.3 Auxiliary uses for Morphology Induction

Another place where morphology induction could be beneficial is in the induction of Arabic pseudo-words. That is, we can use what we have learned about morphology to try to induce pseudo-words that have a strong chance of being actual Arabic words which have not been seen in the CallHome corpus. These words can then be used in the lexicon as a possible means of reducing out of vocabulary words (OOV). It is well known that every OOV contributes to about two recognition errors; so if it were possible to reduce the OOV meaningfully, one could hope to have a reduction in word error rate. On the flip side, adding new words to the lexicon creates more confusability for the recognizer which can increase error rate. Worse still, if we induce character strings, there is no guarantee that they are real words, so we may end up hypothesizing strings that never should have occurred. This will also impact error rate in a negative way. Nonetheless, it is a good experiment to try since it could work to oneŠs advantage. The idea of using morphology to hypothesize new words is not a new one. Other researchers have used human-provided knowledge of morphology to predict all possible forms. A beauty of using morphology induction for this process, though, is that the system can actually hypothesize commonly reoccurring alternations besides just those attributable to morphology. Moreover, information regarding the frequency of certain string transformations are stored as a matter of course in the induction system, which allows one to rank-order the pseudo-words according to likelihood. There is an additional process that we add to our system for word induction which, to our knowledge, has not been present in other systems but could have been. In particular, we require our system to refrain from predicting some types of pseudo-words unless certain substring properties hold. More specifically, say we have a word W_1 which can be decomposed into a root R_1 and an affix A_1. Suppose further that we know that A_1-type words are frequently variants of words with affix A_2. We can therefore hypothesize word W_2 which has root R_1 and affix A_2. Yet we can also require that the first and last n characters of R_1 must be characters that have been observed to start/end the roots forms of other A_2-type words. For training data, we use the Train80 files that were mentioned earlier, along with the morphology that was induced thereon. The evaluation data set was the rest of the data that the team had access to. Table 6.3 shows the performance of our system in terms of three metrics. The first metric is the number of pseudo-words that the strategy generated. The second shows the percentage of originally out-of-vocabulary words which were covered by our pseudo-word generation scheme. The last metric indicates the percentage of pseudo-words that were

actually observed in the evaluation set. We see from the table that the last two columns are especially promising. By supplementing the lexicon with either 35K or 100K new words, we would, respectively, have coverage for 14.9% or 25.0% of the previously unknown words. Unfortunately, there was insufficient time during the workshop to test the impact of this pseudo-word generator. We leave it as a future experiment to perform which could serve as a means of using automatically-induced morphology to improve ASR.

7 Using Out-of-corpus Text Data

7.1 Motivation

As mentioned above, the amount of language model data available for ECA CallHome is extremely small: the number of words in the extended training set (plus the h5_new conversations) only amounts to 180K words. For languages like English, for example, millions of words are typically used for language model estimation. We investigated several approaches to address this problem, which can be grouped into two classes: first, we tried to locate additional conversational training data for Egyptian Arabic; second, we attempted to use large amounts of Modern Standard Arabic data to improve our language model for CallHome.

7.2 Additional Conversational Text

Prior to the workshop we tried to locate conversational Egyptian Arabic text on the Internet. The general problem of finding conversational text is rather difficult, since it involves measuring similar language styles. However, for Arabic, this is made easier by the fact that the spoken dialect contains many common words that are never used in the Modern Standard Arabic. Therefore, we simply identified a set of about 20 words that are common in Egyptian Arabic and searched for those words on the World Wide Web. One problem was to define the spelling of those words since they are not normally part of the written language. We considered a small number of obvious spellings. We found some small amounts of text that contained these words. However, these were often jokes or chat lines. The amount of data found was very small, on the order of several hundred words. It is possible that chat rooms could be monitored for some period in order to collect substantial amounts of text, but we could not do this within the time available for the workshop.

Searches of websites of courses on Egyptian Arabic did not produce a significant amount of data either. Larger amounts of data may exist somewhere (e.g. close captions or scripts of TV shows in ECA) but were not available to us.

7.3 Using MSA Text Data

An alternative source of additional data was the large collection of available news texts in Modern Standard Arabic (see 2). We had more than 300M words of news text and about 9M words of speech transcriptions from broadcast news. The hope was that we could use this text to improve the language model for the CallHome task. We tried several different techniques, namely

- language model interpolation
- text selection plus language model interpolation
- class-based models

- constrained interpolation

All of these experiments (described in detail below) were performed using the script-based recognition system since it matched the (non-diacriticized) news text material. We compared the results to the error rates obtained using only the CallHome training text in script form.

7.3.1 Language Model Interpolation

The simplest approach for combining two language models is to estimate a model from each corpus and take a weighted average of the two estimates for each probability. The weights for these two models are chosen such that they minimize the perplexity (maximize the likelihood) of a held-out training set. We used the SRI Toolkit, which allowed us to estimate the two models separately and then to estimate the optimal interpolation weight. We estimated one language model on the CallHome training data and the other on the corpora listed in Section 2. We found that the optimal weight for the MSA language model was 0.03 (vs. 0.97 for the CallHome language model). This reduced perplexity from 203 to 197. However, it had no measurable effect on the word error rate.

7.3.2 Text Selection plus Language Model Interpolation

One of our assumptions was that it might be beneficial to identify and only use those utterances in the MSA data that are closest in style to conversational speech. For this experiment we used only the Al-Jazeera corpus since it included transcriptions of talk shows and interviews, which we thought might have some dialectal features.

Previous work on text selection in English (e.g. [35]) showed that part-of-speech sequences are a good indicator of style (e.g. formal vs. informal). Given two POS-based language models for two different styles (estimated on text samples of those styles), each utterance in a new corpus can be scored by both models and the following likelihood ratio can be used to identify the model to which it is closer.

To implement this text selection method we first needed to build a tagger for the CallHome corpus. For this purpose we used the CH corpus as well as a pre-release of Arabic Treebank corpus (made available to us by LDC), which is a collection of newspaper articles in MSA of approximately 130K words. The Treebank corpus is hand-annotated with part-of-speech tags (i.e. both with possible and true POS tag for each word). The original tag set that came with the Treebank corpus contained 419 items. The CH corpus, on the other hand, does not provide true POS tag assignments but does contain the list of possible POS tags for each word in the lexicon. We manually clustered the original Treebank tag set such that the resulting tags would be compatible with the CH tags; the resulting set contained 21 tags. We used a statistical trigram tagger trained to find the most likely tag sequence given the observed word sequence, using the model in Equation 8.

$$T^* = argmax_T P(T|W) = argmax_T \frac{P(W|T)P(T)}{P(W)} \quad (8)$$

#	Experiment	Accuracy	OOV Accuracy	Baseline
1	TB, unsupervised	79.5	19.3	75.9
2	TB & CH, unsupervised	74.6	17.6	75.9
3	TB, supervised	90.9	56.5	90.0
4	TB & CH, partially supervised	83.4	43.6	90.0

Table 34: Tagging accuracies (%) for Treebank (TB) and CallHome (CH).

and the following approximation (i.e. a trigram tagger):

$$P(W|T)P(T) \approx \prod_i P(w_i|t_i)P(t_i|t_{i-1}, t_{i-2}) \tag{9}$$

The tagger was implemented using the Graphical Modeling Toolkit [8] developed at the 2001 JHU Workshop. Simple maximum-likelihood estimates were used for the probabilities in Equation 8. In order to provide a probability estimate for unknown words, we distributed a fraction of the singleton probability mass (0.5) across unknown words. This was done by doubling the training data with all singletons replaced by a special token representing unknown words.

Training was performed on the combined Treebank and Ch data sets and was unsupervised way, i.e. no knowledge of the true tag was provided for either the Treebank or CH data. To assess the performance of our tagger we also performed supervised and unsupervised training on the Treebank portions of the data only, and partially supervised training where the combined data was used but only the Treebank portion had supervised tags. Table 34 shows the tag accuracies, which were obtained as averages over a 10-fold cross-validation experiment. Note that in both experiments 3 and 5, the accuracies dropped as Arabic CallHome data were added. This is expected, because Arabic Treebank and Arabic CallHome are quite different sub-languages, and the resulting POS-base language model when using both data sets reflects styles in both sub-languages.

We used our best tagger for the combined corpus and tagged all 9M words of Al Jazeera data. POS trigram models were the trained on Treebank and CH separately. The following scoring method was used to score each tagged Al Jazeera utterance:

$$Score(S) = log(P(S|C)^{1/N_S}/P(S|M)^{1/N_S}) \tag{10}$$

where S is the utterance in question, N_S is the length of the utterance, C is the CallHome language model and M is the MSA language model. We used this method to select a subset of the Al Jazeera data which had approximately the same size as the CallHome data set. An analysis of the selected utterances showed that they were indeed somewhat more conversational or livelier in style (e.g. sports news as opposed to political news). The language model interpolation experiment was repeated with the selected data set; however, no improvement in perplexity or word error rate was observed. A further analysis of the log probability curve

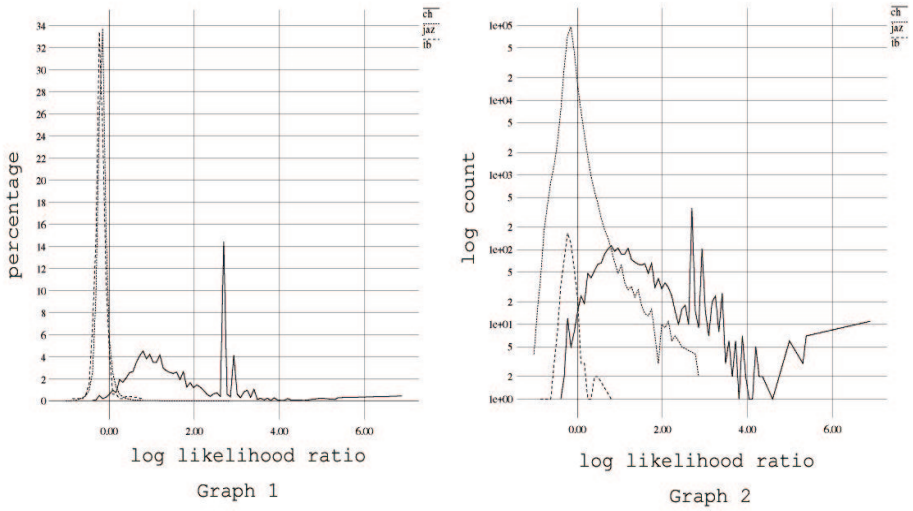

Figure 14: Log-likelihood ratio distributions for CallHome and Al-Jazeera data.

of the two models on the different data sets (Figure 14) showed that the POS n-grams of Al Jazeera and Treebank have a very similar behaviour, indicating that the two corpora are very similar in style.

7.3.3 Class-Based Models

One technique that is sometimes used when there is insufficient data to estimate a reliable language model is to group the vocabulary into word classes, as explained above in Section 5. Then, we use the approximation

$$P(w_3|w_1, w_2) \approx P(w_3|c_3)P(c_3|c_1, c_2) \tag{11}$$

The question is how to obtain the classes. The problem is that, if there is not enough data to estimate a reliable language model, then there is likely to also not be enough data to estimate which words behave in similar ways. For example, how could we propose that *Monday* and *Tuesday* are in the same class if we do not even have enough data to observe the word *Tuesday*?

Our assumption was that although the styles of MSA and CH are different there would still be a significant overlap in the vocabulary. For example, the days of the week would be similar in both corpora, although they might be used differently in the two styles. Therefore, we attempted to estimate the sets of word classes from the large MSA text corpus and to use those classes to estimate the probabilities of class sequences and/or class membership probabilities from the smaller CH corpus.

More specifically, we combined the MSA and CH texts before estimating classes and specified the vocabulary in advance as that belonging to CH. We did this because the vocabulary of the MSA texts was over 1.8M distinct words, while the vocabulary of the conversational speech was only 15K words. Even though the amount of conversational speech text was small, it was essential to include it in the training data

to ensure that all of the words in CH would be included in a class. We then tried estimating the probabilities from the CH training text. The resulting class language model was combined with the original language model, with a weight that was optimized to minimize perplexity. However, the resulting perplexity of the test set did not decrease below that of the original language model derived from CH alone.

7.3.4 Constrained Interpolation

One concern in combining a large general corpus with a small specific one is that, although we might observe more examples of different ngrams of words, their probabilities are likely to be different. Thus, if we just interpolate the two language models, having incorrect probabilities might offset any benefits obtained by having greater coverage. Therefore, we implemented an algorithm to constrain the amount by which the original ngram probabilities obtained from CH were allowed to change.

Consider the case of a trigram that was not observed in the conversational text. That is, even though the appropriate history occurred several times, this particular third word was never observed to follow it. We do not know whether the third word was not observed because there was not enough training data, or because it is an extremely unlikely, maybe impossible, combination. Suppose we then observe that this trigram does occur in the news text and that the probability is reasonably high. The interpolation approach would interpolate this high probability with the original (zero) one to obtain a new value. The rationale for interpolating the two probabilities is that we assume that the probability obtained from the news text corpus is a reasonable estimate of the corresponding probability for the conversational speech, but that we simply did not have enough data available for the conversational speech model. However, we must also consider the possibility that the reason why the two probabilities are different is that these two probabilities are truly different in the two styles of speech. How can we tell which is the case?

Consider the case in which the resulting interpolated probability was such that, given the number of times the history for the trigram (the preceding two words) occurred, the third word should have occurred five times. However, it was never observed in the conversational speech. What is the probability that this would happen if this interpolated probability were correct? The probability is approximately $e-5$. We would consider it more likely that the two corresponding probabilities were simply not the same.

Given how different the two sub-languages are, we felt that the chances that any particular probability would be different were fairly high. Therefore, we set a limit on the amount that the original probability could change. The limit was such that the probability of the observation was approximately $e-1$. It turns out that a good approximation of this is the following set of rules: The amount of change depends on the number of occurrences of the target word.

1. If it has never been observed, then the upper limit on the probability is that corresponding to one observation.

2. If it was observed once, then the upper limit is 2.

3. For higher values, the limit in the change is equal to the square root of the original number of observations.

We also implemented lower bound limits, although these were probably less important. In this case, the limit on the change was also the square root of the original count, except for an original count of one, which was limited to a change of 0.5 counts.

We did not have much time to try this technique at the end of the workshop, but we tried it with an interpolation weight of 0.1 for the news text (higher than the optimal weight of 0.03 with no constraint). However, we observed no benefit for this technique.

7.4 Analysis

After trying several techniques with no success we began to wonder whether there was any similarity at all between the styles of language in the MSA news text and the CH speech. At the very least, we hoped that adding the large news text (which had 2000 times as much data as the conversational corpus) would increase the likelihood that trigrams in the test data would be observed at least once in the combined corpus. This gain would, of course, be offset by some inappropriate observations or by different probabilities in the larger corpus.

We measured the percentage of time that each trigram in a test set of CH was observed in its language model training set. This percentage was only 24.5%, i.e. 75.5% of the time, the three words in the CH evaluation set never occurred in the CH training set. The language model probability would then have to back off to bigram or unigram estimates. When we added the 300M words of news text, this "trigram hit rate" only increased to 25%! Thus, there was almost no increased coverage. This indicates that the style of language is radically different. We would have expected at least some increased coverage, possibly offset by equal or greater negative effects.

This non-increase in hit rate explained all of the negative results in our attempts at combining the large news corpus with the small conversational corpus. The conclusion is that ECA and MSA are different to the point where we can almost think of them as two different languages. Naturally, there are also domain differences that need to be taken into account: all MSA texts were obtained from media sources and focus on topics such as politics, sports, etc. whereas the CH dialogues are of a different nature.

8 Out-Of-Vocabulary Problem

One problem of highly inflectional languages is that the number of out-of-vocabulary (OOV) words encountered in the test data is typically very high. This is certainly true of the ECA corpus: the OOV rate on the development/test data is close to 10%. An analyses of the OOV words showed that most of these (50%) are morphological variants of related words in the training data. 10% are proper names and 40% are other

previously unobserved words. In order to reduce the number of errors due to OOV words we tried adding additional words to the ECA dictionary. In the first case these words were taken from the BBN Broadcast News dictionary (MSA); in the second case, additional inflected words were generated by the finite-state transducer distributed with the ECA corpus. The Broadcast News dictionary contains 1.8M distinct words; adding all of these would reduce the OOV rate from 10% to 3.9% but would naturally overwhelm the ECA dictionary, which only consists of 16K words. Adding only the most frequent 15K (25K) Broadcast News words reduces the OOV rate to 8.9% (8.4%), i.e. only by an insignificant amount. The finite-state transducer is designed to generate all possible inflected forms of ECA verb stems. Running it on all verb stems in the ECA dictionary produced more than 1M additional words, which reduce the OOV rate to 7%. However, inclusion of these words into the dictionary is as infeasible as in the case of the Broadcast News words.

9 Summary and Conclusions

Our stated goals for this workshop were (a) to explore methods of making MSA data usable for training models for colloquial Arabic; and (b) to develop novel statistical models, in particular language models, to better exploit the small amount of training data available for dialectal Arabic.

The first objective was addressed by exploring automatic romanization techniques that enrich Arabic script with vowel and other phonetic information, as well as by the attempt to use MSA data for language modeling. We saw that it is possible to train an automatic romanizer from a small amount of parallel script and romanized data. The use of ASR training data processed by this tool did not lead to the same recognition performance as when training on manual romanized transcriptions. However, performance was significantly better compared to only training on script data. This indicates that using diacritic information system-internally is helpful. It furthermore suggests that future Arabic data collection efforts should include a small amount of hand-diacritized or romanized material, such that similar automatic diacritization tools can be trained.

Using MSA data for language modeling did not yield any improvements. Our analysis showed that MSA and ECA behave like two distinct languages. This means that the only way to obtain more language model training data is to collect additional dialectal data or to transform the MSA data, perhaps according to manually specified grammars, or by using a statistical translation model that converts MSA-style text into dialectal text. The second solution, however, also requires sufficient amounts of training data.

The second goal was met by investigating three different types of languages models designed to better exploit the available ECA data: particle models, morphological stream models, and factored language models. The latter is a more general model, which is not only applicable to morphologically rich languages like Arabic, but can incorporate a wide range of features which may be helpful for predicting words, such as parse structures or semantic features. This approach is accompanied by novel strategies for smoothing called generalized backoff. All these morphology-based language models yielded slight but consistent reductions

in word error rate when combined with standard word-based language models. The overall reduction obtained through improvements to language modeling was statistically significant (difference of proportions test, significance level 0.05). Our goal is to verify these results on a wider range of data sets and in combination with different recognition systems.

A number of useful resources resulted from this effort, including software for the automatic romanization of ECA, a statistical part-of-speech tagger for Arabic, significant extensions to the SRILM language modeling toolkit, and software for automatically inducing Arabic morphology. Most of these resources are publicly available.

References

[1] E.T. Abdel-Massih. *An Introduction to Egyptian Arabic*. The University of Michigan Press, Ann Arbor, 1975.

[2] Yaser Al-Onaizan and Kevin Knight. Translating named entities using monolingual and bilingual resources. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 400–408, 2002.

[3] International Phonetic Alphabet. http://www.arts.gla.ac.uk/IPA/ipachart.html.

[4] S. Badawi and M. Hinds. *Dictionary of Egyptian Arabic*. Librairie du Liban, 1987.

[5] A. L. Berger, S.A. Della Pietra, and V.J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[6] P. Beyerlein. Discriminative model combination. In *Proc. ICASSP*, pages 481–484, 1998.

[7] P. Beyerlein, W. Byrne, J.M. Huerta, S. Khudanpur, B. Marthi, J. Morgan, N. Peterek, J. Picone, D. Vergyri, and W. Wang. Towards language independent acoustic modeling. In *Proc. ICASSP*, 2000.

[8] J. Bilmes and G. Zweig. The Graphical Models Toolkit: An open source software system for speech and time-series processing. *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, 2002.

[9] J. A. Bilmes. Graphical models and automatic speech recognition. In R. Rosenfeld, M. Ostendorf, S. Khudanpur, and M. Johnson, editors, *Mathematical Foundations of Speech and Language Processing*. Springer-Verlag, New York, 2003.

[10] M. Brent, S.K. Murthy, and A. Lundberg. Discovering morphemic suffixes: a case study in MDL induction. In *Proceedings of 5th Internationl Worksh on Artificial Intelligence and Statistics*, 1995.

[11] P.F. Brown, V.J. Delle Pietra, P.V. deSouza, J.C. Lai, and R.L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

[12] T. Buckwalter. http://www.xrce.xerox.com/competencies/content-analysis/arabic/info/translit-chart.html.

[13] B. Byrne, J. Hajic, P. Ircing, F. Jelinek, S. Khudanpur, P. Krbec, and J. Psutka. On large vocabulary continuous speech recognition of highly inflectional language - Czech. In *Proceedings of Eurospeech*, 2001.

[14] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of*

the Association for Computational Linguistics, pages 310–318, San Francisco, 1996. Association for Computational Linguistics, Morgan Kaufmann Publishers.

[15] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. Technical Report Tr-10-98, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, August 1998.

[16] CP1256. http://www.microsoft.com/globaldev/reference/sbcs/1256.htm.

[17] T. Schultz D. Kiecza and A. Waibel. Data-driven determination of appropriate dictionary units for korean LVCSR. In *Proceedings of ICASSP*, 1999.

[18] K. Darwish. Building a shallow arabic morphological analyser in one day. In *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, Philadelphia, PA, 2002.

[19] F. Debili, H. Achour, and E Souissi. De l'étiquetage grammatical à la voyellation automatique de l'arabe. *Correspondances de l'Institut de Recherche sur le Maghreb Contemporain*, 17, 2002.

[20] S. Deerwestr, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41:391–407, 1990.

[21] H. DeJean. Morphemes as necessary concepts for structures: discovery from untagged corpora. In *Workshop on paradigms and grounding in natural language learning*, pages 295–299, Adelaide, Australia, 1998.

[22] Y. A. El-Imam. Synthesis of arabic from short sound clusters. *Computer, Speech, and Language*, 15:355–380, 2001.

[23] G. Zavagliakos et al. The BBN Byblos 1997 large vocabulary conversational speech recognition system. In *Proceedings of ICASSP*, 1998.

[24] J. Billa et al. Multilingual speech recognition: The 1996 byblos callhome system. In *Proceedings of Eurospeech*, pages 363–366, 1997.

[25] J. Billa et al. Arabic speech and text in Tides Ontap. In *Proceedings of HLT*, 2002.

[26] J. Billa et al. Audio indexing of broadcast news. In *Proceedings of ICASSP*, 2002.

[27] N. Friedman and D. Koller. Learning Bayesian networks from data. In *NIPS 2001 Tutorial Notes*. Neural Information Processing Systems, Vancouver, B.C. Canada, 2001.

[28] Ya'akov Gal. An HMM approach to vowel restoration in Arabic and Hebrew. In *Proceedings of the Workshop on Computational Approaches to Semitic Languages*, pages 27–33, Philadelphia, July 2002. Association for Computational Linguistics.

[29] E. Gaussier. Unsupervised learning of derivational morphology from inflectional lexicons. In *Proceedings of the ACL Workshop on Unsupervised Learning in Natural Language Processing*, University of Maryland, 1999.

[30] P. Geutner. Using morphology towards better large-vocabulary speech recognition systems. In *Proceedings of ICASSP*, pages 445–448, 1995.

[31] H. Glotin, D. Vergyri, C. Neti, G. Potamianos, and J. Luettin. Weighting schemes for audio-visual fusion in speech recognition. In *Proc. ICASSP*, 2001.

[32] J. Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 2001.

[33] M. Huckvale and A.C. Fang. Using phonologically-constrained morphological analysis in continuous speech recognition. *Computer, Speech and Language*, 16:165–181, 2002.

[34] ISO. http://www.iso.ch/cate/35040.html.

[35] R. Iyer. *Improving and predicting performance of statistical language models in sparse domains*. PhD thesis, Boston University, 1998.

[36] F. Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1997.

[37] P. Geutner K. C,arki and T. Schultz. Turkish LVCSR: towards better speech recognition for agglutinative languages. In *Proceedings of ICASSP*, 2000.

[38] O. Karboul-Zouari. Die hocharabische Sprache und Romanisierung ihrer Schrift. Technical report, University of Karlsruhe, 1999. Senior Project Report.

[39] Kevin Knight and Jonathan Graehl. Machine transliteration. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 128–135, Somerset, New Jersey, 1997. Association for Computational Linguistics.

[40] S.L. Lauritzen. *Graphical Models*. Oxford Science Publications, 1996.

[41] S. Martin, J. Liermann, and H. Ney. Automatic bigram and trigram clustering for word classes. *Speech Communication*, pages 19–37, 1998.

[42] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computing Journal*, 7(4):308–313, 1965.

[43] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[44] Qalam. http://eserver.org/langs/qalam.txt.

[45] CAT scheme. http://almashriq.hiof.no/general/400/410/418/classical_arabic_transliteratio.

[46] P. Schone. Knowledge-free induction of inflectional morphologies. In *Proceedings of NAACL*, Pittsburgh, PA, 2001.

[47] P. Schone. *Towards Knowledge-Free Induction of Machine-Readable Dictionaries*. PhD thesis, University of Colorado at Boulder, 2001.

[48] P. Schone and D. Jurafsky. Knowledge-free induction of morphology using latent semantic analysis. In *Proceedings of CoNLL*, Lisbon, Portugal, 2000.

[49] Bonnie Glover Stalls and Kevin Knight. Translating names and technical terms in arabic text. In *Proceedings of the 1998 COLING/ACL Workshop on Computational Approaches to Semitic Languages*, 1998.

[50] A. Stolcke. SRILM- an extensible language modeling toolkit. In *Proc. Int. Conf. on Spoken Language Processing*, Denver, Colorado, September 2002.

[51] Unicode. http://www.unicode.org/.

[52] D. Vergyri, S. Tsakalidis, and W. Byrne. Minimum risk acoustic clustering for multilingual acoustic model combination. In *Proc. ICSLP*, 2000.

[53] E.W.D. Whittaker and P.C. Woodland. Particle-based language modeling. In *Proc. Int. Conf. on Spoken Language Processing*, Beijing, China, 2000.

[54] D. Yarowski and R. Wicentowski. Minimally supervised morphological analysis by multimodal alignment. In *Proceedings of the ACL*, Hong Kong, 2000.