

Abstract of “Creating Algorithms for Parsers and Taggers for Resource-Poor Languages Using a Related Resource-Rich Language” by Dmitriy Genzel, Ph.D., Brown University, May 2006.

Modern statistical natural language processing techniques require large amounts of human-annotated data to work well. For practical reasons, the required amount of data exists only for a few languages of major interest. In my work I show how a resource-rich language can be leveraged to produce the necessary resources and tools for related resource-poor languages.

The work consists of two parts. The first part focuses on building a word-to-word translation model from parallel corpora. This involved a variety of methods, some well-known and some new. The new methods focus on exploiting lexical and syntactic similarities of the languages. The second part utilized the word-to-word model created in the first part, to first assign parts of speech and then parse the text in several related resource-poor languages.

Creating Algorithms for Parsers and Taggers for Resource-Poor Languages Using a Related
Resource-Rich Language

by

Dmitriy Genzel

B. S., Drexel University, 2001

M. S., Drexel University, 2001

Sc. M., Brown University, 2002

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island

May 2006

© Copyright 2005 by Dmitriy Genzel

This dissertation by Dmitriy Genzel is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____
_____ Eugene Charniak, Director

Recommended to the Graduate Council

Date _____
_____ Mark Johnson, Reader

Date _____
_____ Tom Dean, Reader

Approved by the Graduate Council

Date _____
_____ Sheila Bonde
Dean of the Graduate School

Vita

Degrees

- ScM in Computer Science, 2002, Brown University
- MS in Computer Science, 2000, Drexel University
- BS in Computer Science and in Mathematics (double major), 2000, Drexel University

Teaching Experience

- Teaching Certificates I and III, Sheridan Center for Teaching and Learning, Brown University
Received instruction in pedagogy and professional development
- Instructor, Summer Program, Brown University, 2003 and 2004
Taught 3-week (3 hours a day) intensive courses *Computer Programming in Perl*
Designed and presented all course material, graded, and assisted students in the lab
- Teaching assistant, *Operating Systems*, Brown University, Fall 2001
Assisted students with course material
Graded assignments and tests
- Teaching assistant, *Artificial Intelligence* (graduate level), Drexel University, Fall 1999
Prepared and graded homework assignments
Assisted students with course material

Research and Technical Experience

- Computational Linguistics (Brown Laboratory for Linguistic Information Processing)
Dissertation topic: Creating Linguistic Tools and Resources for a Resource-Poor Language
Using a Related Resource-Rich Language
Advisors: Eugene Charniak and Mark Johnson
See *Publications* for more detail
- Computational Psycholinguistics
Thesis topic: Information theoretic complexity and sentence reading times
Advisor: Julie Sedivy
- Internship at Microsoft Research, Natural Language Processing group, Summer 2002
Advisor: Bob Moore
Produced an anaphora resolution system
- Time-Critical Reasoning and Robotics, Drexel University, 6/1999-8/2000
Advisor: Lloyd Greenwald
Built an interface in C for sharing data between Lego robot and PC.
Developed software agent for RoboCup soccer simulation in C++ using resource-bounded reasoning.
- Automated Mechanical Assembly Planning, Drexel University, 9/1998-6/1999
Advisor: William Regli
Presented a poster on Drexel Research Day, 4/1999.
Developed several tools in SDRC I-DEAS to access CAD/CAM data.
Created an Automated Mechanical Assembly Planner in Java, Tcl/Tk.
- Programmer (co-op program), Unisys Corporation, Malvern, PA, 9/1997-9/1998
Participated in a team effort to develop BioWare (Fingerprint-based identification system)

Ported C code from Unix to Windows NT.

Honors

- IGERT Fellowship, Brown University, 2000-present
- A.J. Drexel Scholarship; Drexel Dean's List (1997-2000)
- Ranked 203rd in the 1998 National Putnam Mathematical Competition
- 4th place in All-Ukrainian Mathematics competition 1996
- 2nd place in All-Ukrainian Computer Science competitions 1995, 3rd in 1994

Refereed Conference Publications

- Dmitriy Genzel. *Inducing a bilingual dictionary from a parallel corpus in related languages.*
Presented at HLT/EMNLP-05, Vancouver, Canada.
- Dmitriy Genzel. *A Paragraph Boundary Detection System.* Presented at CIC-Ling-05, Mexico City, Mexico
- Dmitriy Genzel and Eugene Charniak. *Variation of Entropy and Parse Trees of Sentences as a Function of the Sentence Number.* Presented at EMNLP-03, Sapporo, Japan.
- Dmitriy Genzel and Eugene Charniak. *Entropy Rate Constancy in Text.* Presented at ACL-02, Philadelphia, PA, USA

Invited Talks

- *Entropy Rate Constancy in Text*
AT&T Research
University of Utah

- *Paragraph Boundaries and Sentence Structure*

Drexel University

Administrative Activities

- Graduate Students' Faculty Search Committee Member, CS Department, Brown University, 2004

Duties include providing feedback on faculty candidates

- Graduate Students' Faculty Search Committee Czar, CS Department, Brown University, 2002
Duties include collecting graduate students' input on faculty candidates and presenting to the faculty

- Info Khan, Computer Science Department, Brown University, 2001-2002

Responsible for graduate students' portion of the departmental web site

Professional Activities

- Assisted in 2002 ACL Best Paper Award paper review
- Assisted in 2003 and 2004 ACL officers election
- Co-chair of ACL Student Workshop, 2004

Acknowledgments

I would like to start by thanking my advisor, Eugene Charniak, for helping me muddle through to the very end, and for his advice and direction. I also thank my committee members, Mark Johnson and Tom Dean for taking the time to read this boring document. In addition, my thanks go to Mark for acting as a second advisor and a source of help for linguistic and mathematical questions. I would also like to acknowledge the help I received from various members of BLLIP throughout my years at Brown. I am grateful to all faculty members at Brown who have taught courses I took or interacted with me in other ways, especially Thomas Hofmann and Shriram Krishnamurthi.

My PhD would not have been possible if it were not for help and encouragement I received from my undergraduate professors at Drexel, especially my advisor, Lloyd Greenwald.

I thank my mother for nagging and badgering me to finish soon, and my sister for not doing so. Finally, and most importantly, I owe this work to my wife Mariya who suggested that I go to graduate school, helped and supported me throughout, proofread all my papers, and kept me on track to finish. I also thank my son, Alexey, for being born when I was almost done and benevolently deciding not to drive me mad although he had every right and opportunity to do so.

Contents

List of Tables	xii
List of Figures	xiii
1 Introduction and Motivation	1
2 Previous and Related Work	4
3 The Slavic Languages and Their Grammars	6
3.1 Historical Overview	6
3.2 Vocabulary	7
3.3 Grammar	9
4 Building the dictionaries	11
4.1 Obtaining the Data	12
4.2 Preprocessing	12
4.2.1 Automatically Learning the Letter Case System	13
4.2.2 Automatically Learning the Punctuation	14
4.2.3 The Algorithm	15
4.3 The Pairwise Word Translation Model	15
4.3.1 GIZA (forward)	16

4.3.2	GIZA (backward)	17
4.3.3	Character-based model	17
4.3.4	Prefix Model	19
4.3.5	Suffix Model	20
4.3.6	Constituency Model	21
4.4	Building the Joint Model	22
4.4.1	Combining Models of Hidden Data	22
4.4.2	Combining Pairwise Models	26
4.5	Evaluation	28
5	Building tools	31
5.1	Introduction	31
5.2	Building a tagger	32
5.2.1	Finding good translations (no other data)	33
5.2.2	Finding good translations (with other data)	34
5.2.3	Training taggers	34
5.2.4	The algorithm	35
5.3	Evaluating a tagger	36
5.3.1	Error analysis	36
5.4	Building a parser	37
5.4.1	Building an indirect parser	37
5.4.2	Building a direct parser	38
5.4.3	Parsing results	39
5.4.4	Error analysis	39
6	Conclusions and Future Work	41
6.1	Conclusions	41

6.2 Future Work 42

★ Parts of this thesis have been previously published as (Genzel, 2005)

List of Tables

3.1	Genesis 1:1 in Slavic languages and English (with transliteration for convenience) . . .	8
3.2	Genesis 2:6 in Slavic languages and English with transliteration and glosses	8
3.3	Word similarity: matching roots out of 10 sentence pairs	9
4.1	Evaluation for Russian-Ukrainian (with Belorussian to tune)	29
4.2	Evaluation for Russian-Ukrainian (with Belorussian and Polish)	30
5.1	Tagging performance	36
5.2	Parser for Russian	39

List of Figures

3.1	Genesis 1:1 in Russian, parsed	10
4.1	Combining imperfect models	23
5.1	A sample PCFG	38

Chapter 1

Introduction and Motivation

Modern statistical natural language processing techniques require large amounts of human-annotated data to work well. For practical reasons, the required amount of data exists only for a few languages of major interest, either commercial or governmental. As a result, many languages have very little computational research done in them, especially outside their borders. Some of these languages are in fact major languages, with hundreds of millions of speakers. Of the top 10 most spoken languages (The World Factbook (2004), 2000 estimate), the Linguistic Data Consortium at University of Pennsylvania, the premier provider of corpora in the United States, provides text corpora in 7 languages (English, German, Japanese, Chinese, Korean, Spanish, Portuguese), the 3 remaining languages being Hindi, Bengali, and Russian. LDC also provides data for a few languages not in the top 10, such as French, Arabic, and Czech. Of the second-tier languages, numbering between ten and a hundred million of speakers, most are Asian, and most of the remainder are Eastern European. They belong to relatively few language families. Very few of these languages seem to have published computational linguistics work done in them, at least at the international conferences, such as the ACL¹.

¹The search through ACL Anthology, for e.g. Telugu (~70 million speakers) shows only casual mention of the language.

The situation is not surprising, nor is it likely to significantly change in the future. Luckily, most of these less-represented languages belong to language families with several prominent members. As a result, some of these languages have siblings with more resources and work done in them.² Interestingly, the better-endowed siblings are not always the ones with more native speakers, since political considerations are often more important³.

If one is able to use the resources available in one language (henceforth referred to as *source*) to facilitate the creation of tools, such as parsers, for all the languages related to the language in question (*target*), the above-mentioned problem would be alleviated. This is the ultimate goal of this project.

There exists a recent research paradigm, in which the researchers work on algorithms that can rapidly develop machine translation and other tools for an obscure language that suddenly becomes important. This work falls into this paradigm, under the assumption that the language in question has a less obscure sibling.

Moreover, the problem is intellectually interesting. While there has been significant research in using resources from another language to build, for example, parsers, there have been very little work on utilizing the close relationship between the languages to produce high quality tools. Furthermore, there has been little work exploiting the fact that for a given source there are multiple related languages which can help in mutually improving the overall quality of parsers we can build.

We are, therefore, interested in the following questions:

1. Can we build an algorithm that takes advantage of a close relationship between languages to overcome the scarcity of resources in a language and produce a high-quality parser for this language?
2. Furthermore, can we take advantage of having multiple target languages to improve such a parser's quality?

²Telugu's fellow Dravidian language *Tamil* (~65 million speakers) has seen some papers at the ACL.

³This is the case with Tamil vs. Telugu.

Specifically, we produce an algorithm that can take a parser in a given source language, and some parallel texts, and produce parsers for several (two or more) related languages. Such an algorithm needs to be general and not specific to any language family. For practical reasons, I am focusing on one language family, namely Slavic, but the algorithm will need to be tested on another language family, to demonstrate that it is in fact general.

Among the Slavic languages, Czech has the most resources and tools, and will serve as my source language. I will focus on several target languages. First, Russian, because it is most widely spoken among the Slavic languages. Second, Ukrainian, as the more obscure, yet very widely spoken (~40 million speakers). Third, Polish, for similar reasons (~45 million). Fourth, Belorussian, because Belarus is essentially closed to the West, and thus attracts very little interest in the Western-style computational linguistics research, and also because it might be of political interest to the U.S. government which considers it a dictatorship (The World Factbook (2004)).

I have produced a system that can serve as a framework of tools which can be used on target languages unknown to a researcher with some hope of producing a useful output. It may even be usable by someone who is not fluent in the source language.

Chapter 2

Previous and Related Work

There has not been much work done on related languages, especially in syntax. Most of the work focused on the lexicon. One overview is provided by Melamed (2000). There is work on lexicon induction using string distance or other phonetic/orthographic comparison techniques, such as Mann and Yarowsky (2001) or semantic comparison using resources such as WordNet (Kondrak, 2001). Such work, however, primarily focuses on finding cognates, whereas we are interested in translations of all words. Moreover, while some techniques (e.g., Mann and Yarowsky (2001)) use multiple languages, the languages used *have* resources such as dictionaries between some language pairs. We do not require any dictionaries for any language pair.

There has also been some work on using a parallel corpus to induce tools for another language, but such approaches are more general and are intended for unrelated languages (Yarowsky et al., 2001; Smith and Smith, 2004; Pado and Lapata, 2005). This project's goal, on the other hand, is to exploit the similarity of the languages in question, and we can achieve significantly better performance¹.

Other recent work focuses on exploiting some supervised data (along with more unsupervised

¹We are not be able to do a direct comparison, because the languages worked on by these researchers are not *minor* and do not fall within our paradigm

cross-language data) which may exist for a given language (Xi and Hwa, 2005).

An important element of this work is focusing on more than a pair of languages. There is an active research area focusing on multi-source translation (e.g. Och and Ney (2001)). In our case, however, we work in a *multi-target* rather than *multi-source* setting. Furthermore, we do not work on translation, but rather on parsing, for which we are not aware of any work involving more than two languages.

There has also been some work on translation by parsing a parallel multitext (which in practice is always a bitext), such as that of Melamed (2004). This work demonstrates that if one has a grammar (or equivalently, a parser) for multiple languages one can build an MT system from them. Therefore, having built a parser in this project, we may also be able to automatically build machine translation systems for target languages as well.

Finally, there has been some work on machine translation for related languages (Hajic et al., 2000) which uses simple techniques to translate Czech into Slovak. Since they had parsers and other tools for both languages and were trying to translate using them whereas we want to create tools by translating, this is in a way a reverse problem.

Chapter 3

The Slavic Languages and Their Grammars

3.1 Historical Overview

Since we are focusing on the Slavic languages, it would be appropriate to discuss what kind of task we are facing and whether it would be similar or different for another language family. A more detailed overview of the Slavic languages and their relationships is provided by Entwistle and Morison (1949).

We are working on five languages, three of which belong to the East Slavic subfamily (Russian, Ukrainian and Belorussian) and two to the West Slavic one (Czech and Polish). In fact, we are crossing the subfamily boundary when going from Czech to Russian and Ukrainian which must be taken into account during evaluation.

Historically, all Slavic languages have split off a common root, the Old Slavonic language, starting approximately in the 10th–12th century¹. The languages changed both through tremendous geographic dispersal during this period and through contact with other language groups.

¹The original Old Slavonic survives in part through its use in the liturgy of the Russian and Serbian Orthodox churches.

The West Slavic languages are heavily indebted to German and Romance (mainly Latin) languages. Czech speakers have lived under German-speaking Austrian rule for nearly half a millennium, and during the nineteenth century had to survive a serious Germanization effort by the Austrian authorities. Polish is also strongly influenced by German, as well as by Russian because of Poland's common border with the East Slavic countries. Polish is additionally heavily influenced by Latin, which was commonly spoken by the Polish nobility, and Lithuanian because of the Polish-Lithuanian union during the 15th–18th centuries.

The East Slavic languages had significant influence from Greek (through the Orthodox Church), Tartar (through Mongol domination) and the Finno-Ugric languages (through contact with various indigenous peoples). The latter two influences are particularly strong for Russian. Ukrainian and Belorussian were more influenced by Polish (both territories being a part of Poland for a long time) and indirectly by German and Latin.

Of course, by the modern times all of these languages were heavily influenced by the important world languages of the time: Latin, French, German, and most recently English.

Nevertheless, all of these languages retained the basic grammar of the Old Slavonic, even though the vocabulary may have drifted apart. Even the vocabulary, though, is mostly mutually comprehensible. In fact, the most challenging aspect for a speaker of one of these languages learning another is pronunciation which varies significantly. Luckily, it is not a problem for us.

The East Slavic languages use Cyrillic script, while the West Slavic ones use Latin script.

A sample sentence in all of these languages is given in Table 3.1.

3.2 Vocabulary

As can be seen from these sample sentences, the basic noun terms, such as *God*, *earth* and *heaven* are still the same in all of these languages, as is the main verb. On the other hand we have sentences like Genesis 2:6 (Table 3.2) which use more obscure words. In this example the Czech version is

Table 3.1: Genesis 1:1 in Slavic languages and English (with transliteration for convenience)

English	In the beginning God created the heaven and the earth.
Belorussian	На пачатку стварыў Бог неба і зямлю.
(in Latin script)	Na pachatku stvaryw Bog neba i zyamlyu
Russian	В начале сотворил Бог небо и землю.
(in Latin script)	V nachale sotvoril Bog nebo i zemlyu.
Ukrainian	На початку створив Бог небо та землю.
(in Latin script)	Na pochatku stvoryv Bog nebo ta zemlyu.
Czech	Na počátku stvořil Bůh nebe a zemi.
Polish	Na początku Bóg stworzył niebo i ziemię.

Table 3.2: Genesis 2:6 in Slavic languages and English with transliteration and glosses

English	But there went up a mist from the earth, and watered the whole face of the ground.
Belorussian	Але пара падымалася зь зямлі і расіла ўсё ўлоньне зямлі.
(latin script)	Ale para padymalasya z zyamli i rasila vse ulonne zyamli.
(gloss)	But steam was-rising from earth and watered all womb of-earth.
Russian	Но пар поднимался с земли и орошал все лице земли.
(latin script)	No par podnimalsya s zemli i oroshal vsyo lice zemli.
(gloss)	But steam was-rising from earth and watered all face of-earth
Ukrainian	І пара з землі підіймалась, і напувала всю землю.
(latin script)	I para z zemli pidijmalas, i napuvala vsyu zemlyu.
(gloss)	And steam from earth was-rising, and let-drink all earth.
Czech	Jen záplava vystupovala ze země a napájela celý zemský povrch.
(gloss)	But flood rose-above out-of earth and let-drink all earth surface.
Polish	I rów kopał w ziemi, aby w ten sposób nawadniać całą powierzchnię gleby.
(gloss)	And ditch dug-out in earth, so-that by that method flood all surface of-earth.

no longer comprehensible to an East Slavic speaker. The Polish version is even more interesting: it is almost comprehensible, but it seems to mean something different². The East Slavic languages, however, remain very close.

Generally, it is believed that the vocabulary of the Slavic languages is so close that there have even been efforts to create an auxilliary Slavic language from mid-XIX century (Herkel, 1826; Hosek, 1907) to the modern day projects, such as Slovio (Hucko, 2005). The latter uses Esperanto-like grammar and those words which are more or less the same in all Slavic languages. The coverage of

²While all the other versions talk about the mist coming from the Earth to water it, the Polish one seems to be talking about digging a ditch to accomplish the same thing. This pinpoints a common problem: the translations are often not so parallel.

Table 3.3: Word similarity: matching roots out of 10 sentence pairs

Language Pair	Word pairs	No root match	Root match	% root match
Russian vs. Ukrainian	111	33	78	70%
Russian vs. Polish	112	46	66	59%
Russian vs. Czech	116	74	42	36%
Russian vs. English	105	96	9	9%

this language is apparently sufficient to allow efficient communication between Slavic speakers, and can be understood by them (although not spoken or written) without any training.

To test the vocabulary similarity more quantitatively, we extract 10 random sentences from several bibles and check how many aligned word pairs have a common root³. The results are shown in Table 3.3. Russian-English provided as a comparison⁴.

These results show that there is a real advantage in dealing with related languages, as far as matching vocabulary is concerned.

3.3 Grammar

The Slavic languages remain very similar in their basic grammar⁵ (a more detailed overview is provided by Bernshtein (1961).) All of these languages are basically free order, although there is an unmarked order, subject-verb-object. Interestingly, the Bible is written in a very marked way, presumably because translators tried to keep to the original Hebrew word order (verb-subject-object) which makes the text sound quaint. As can be seen in Table 3.1 three of the four languages say literally *created God heaven and earth*.

More generally, the Slavic languages have the following features:

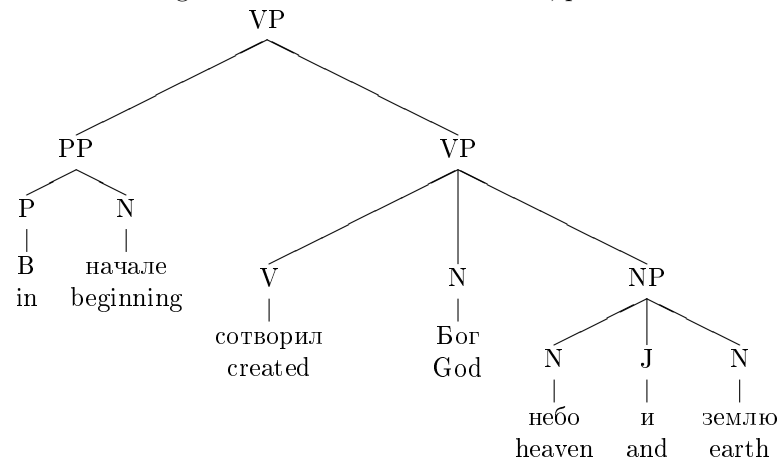
- three noun/adjective genders
- gender and number agreement for nouns and adjectives (and sometimes verbs)

³The random Russian sentences remain the same across all the pairs.

⁴There was a single match which was not a proper noun.

⁵Except Bulgarian, but we are not dealing with it here.

Figure 3.1: Genesis 1:1 in Russian, parsed



- developed grammatical case system (6-7 cases)
- number and person agreement for subjects and verbs
- lack of concept of definiteness (and of articles *a* and *the*)

An example of a parsed Russian sentence is given in Figure 3.3. Generally we may conclude that CFG rules for these languages are nearly the same (as long as they do not involve lexical items) and it is probably reasonable to expect that PCFG rules would have probabilities that are close.

Chapter 4

Building the dictionaries

The first stage of the work involves building a high-quality word translation models between the source and target languages. These models are needed for the second stage of the project. Some bilingual dictionaries already exist for the languages we will be working on, but they do not have a high enough coverage for our purpose, nor are they probabilistic. In particular, they usually provide a mapping only for the base forms of each word, whereas we need the mapping between *all* word forms. It is important to note that since the languages in question are related we can reasonably assume that most of the word forms that exist in one language would also exist in the other. For example, a typical noun in Russian has 12 forms (6 cases in 2 numbers, although some may coincide) and one in Ukrainian may have 14 (it has an extra case), but that extra case (*vocative*) is extremely rare and often has the same form as *nominative* and so for many words the number of forms is in fact the same.

Our goal for this part of the project was to produce an end-to-end system that can take a sentence-aligned multitext in the (one) source and (multiple) target languages and produce high quality word translation models for all language pairs. No special tools for any of the languages are assumed. In fact, at this stage there is no distinction between source and target languages.

The general approach is to build models for each language pair, with each model having a

number of free parameters. The parameters can then be set in such a way as to make all models most consistent with each other. Ultimately, we obtain a joint word translation model over all the languages, which is as consistent as possible with all the pairwise models.

4.1 Obtaining the Data

The first problem was deciding on the data to be used and obtaining it. We decided to use the Bible as the basic text, since it is universally available for a large number of languages and for free. It also provides an additional benefit of being verse-aligned. We were able to obtain electronic versions of the Bibles for all the languages we needed, in some cases even multiple versions. A minor problem involved differences in structure between Protestant, Catholic, and Orthodox versions, but we were able to align the texts with only a small amount of work. The texts (at least the ones that clearly allow redistribution) will be made available online.

4.2 Preprocessing

Before learning could be applied, it was necessary to prepare the texts, by converting them into a proper format. This involved standardizing the cases (by converting all text into lowercase) and tokenizing (by separating punctuation). This required a case mapping table and a list of the punctuation characters. While a native speaker of a language could easily produce these (or they can be obtained from the Internet or even come with an operating system¹), it seemed useful not to have to rely on this information, if it can be learned automatically. Automatic preprocessing would make it unnecessary to “localize” the code, whenever a new language is added. Also, we would not need to specify in which particular languages and/or encodings my texts are.

¹Unix locale system contains much of the needed information.

4.2.1 Automatically Learning the Letter Case System

If a language has a letter case system, it means that it may be that two different sequences of characters in fact refer to the same word, except that one of them occurs at the beginning of a sentence, and so has the first character uppercased. This is not a useful distinction for my purposes, and so it is helpful to convert all text into one case.

A given language may or may not have a case system, so it is necessary to determine this fact for each input file. If it does not, then nothing needs to be done; otherwise a table mapping each uppercase character to a corresponding lowercase character is produced, and the mapping is performed². In order to learn the mapping, we rely upon the following principles, which seem to be valid for the languages we are interested in:

- Uppercase letters occur almost exclusively at the beginning of words
- This is not true for lowercase letters³
- Uppercase letters are less frequent than lowercase letters
- If a language has a case system, about half the letters are uppercase. This implies that if there do not appear to be enough uppercase letters, the language has no case system.
- An uppercase letter maps to a lowercase letter if there are many word pairs that would map into each other under this mapping
- Most encodings have blocks of lower- and uppercase letters, with offset between the mapping letters being more or less constant
- Punctuation characters are not letters and have no case⁴

²In principle, it makes no difference whether the text is converted to lower or upper case, as long as it is consistent for that text. We, however, always convert to lower case.

³There may be some lowercase letters which often occur as the first letter, and so this principle is not completely correct.

⁴To use this, we need to learn which characters are punctuation. We do not assume that any specific character is.

- Digits are not letters and have no case⁵.

4.2.2 Automatically Learning the Punctuation

It is necessary to have all tokens in a text separated by a space, in order for the system to work well. Unfortunately, in most languages punctuation characters often follow (or precede) the word with no intervening space. Moreover, the set of punctuation characters used for a given language often differs slightly from that used by another language, even though the standard ASCII punctuation characters are used by many languages. Also, a text might use special and non-standard characters for formatting. Even a native language speaker might easily miss one or two such characters⁶. Finally, following the principle of not requiring the user to specify the language the text is in, it is clear that the set of punctuation characters needs to be learned automatically.

We rely upon the following principles:

- Letters are not punctuation characters. If something has a case, it is a letter.
- Digits are not punctuation characters. Digits almost always occur together with digits, and never with any other characters.
- Punctuation characters do not normally occur in the middle of a word. If they do, all the characters either to the left or to the right must be punctuation as well.
- Most punctuation occurs to the right of a word.
- A character is likely punctuation if there are many word pairs, such that the removal of the character in question maps one of the words into the other.

⁵We need to learn which characters are digits, we do not assume that ASCII codes for 0-9 are in fact digits.

⁶This has, in fact, happened to us.

4.2.3 The Algorithm

To achieve the best performance, it appears to be most optimal to learn the case system and the punctuation at the same time, as well as the list of digits. The algorithm to do so is entirely heuristic, based on the two sets of principles above. It involves first learning a reasonably high quality case mapping table (or determining that there isn't one), then learning digits and punctuation, and then going over the remaining characters to see if they might be letters and can be fitted into a case mapping table.

4.3 The Pairwise Word Translation Model

At this stage the text is pre-processed and we are ready to start learning pairwise word translation models. Our goal at this stage is to take a parallel bitext in related languages A and B and produce a joint probability model $P(x, y)$, where $x \in A, y \in B$. Equivalently, since the models $P_A(x)$ and $P_B(y)$ are easily estimated by maximum likelihood techniques from the bitext, we can estimate $P_{A \rightarrow B}(y|x)$ or $P_{B \rightarrow A}(x|y)$. Without loss of generality, we will build $P_{A \rightarrow B}(y|x)$.

The model we build will have a number of free parameters. These parameters will be set at a later stage, by an algorithm using multiple pairwise models.

In this section we will assume that the parameters are fixed.

Our model is a mixture of several components, each discussed in a separate section below:

$$\begin{aligned}
 P_{A \rightarrow B}(y|x) &= \lambda_{fw}(x)P_{fwA \rightarrow B}(y|x) + \lambda_{bw}(x)P_{bwA \rightarrow B}(y|x) \\
 &+ \lambda_{char}(x)P_{charA \rightarrow B}(y|x) + \lambda_{pref}(x)P_{prefA \rightarrow B}(y|x) \\
 &+ \lambda_{suf}(x)P_{sufA \rightarrow B}(y|x) + \lambda_{cons}(x)P_{consA \rightarrow B}(y|x)
 \end{aligned} \tag{4.1}$$

where all λ s sum up to one. The λ s are free parameters.

The components represent various constraints that are likely to hold between related languages.

4.3.1 GIZA (forward)

This component is in fact GIZA++ software, originally created by John Hopkins University’s Summer Workshop in 1999, improved by Och (2000). This software can be used to create word alignments for sentence-aligned parallel corpora as well as to induce a probabilistic dictionary for this language pair.

The general approach taken by GIZA is as follows. Let L_A and L_B be the portions of the parallel text in languages A and B respectively, and $L_A = (x_i)_{i=1\dots n}$ and $L_B = (y_i)_{i=1\dots m}$. We can define

$$P(L_B|L_A) = \max_{P_{A \rightarrow B}} \max_{P_{\text{aligns}}} \sum_{i=1}^n \sum_{j=1}^m P_{A \rightarrow B}(y_j|x_i) P_{\text{aligns}}(x_i|j)$$

The GIZA software does the maximization by building a variety of models, described in detail by Brown et al. (1990; Brown et al. (1993). In particular, it starts by assuming all words are equally likely to map into any other word in the same sentence. Later models modify this assumption, by first taking into account that not all positions are equally likely to map, and then further by modeling the number of words that a given word might translate into.

GIZA can be tuned in various ways, most importantly by choosing which models to run and for how many iterations. We treat these parameters as free, to be set along with the rest at a later stage.

As a side effect of GIZA’s optimization, we obtain the $P_{A \rightarrow B}(y|x)$ that maximizes the above expression. It is quite reasonable to believe that a model of this sort is also a good model for our purposes. This model is what we refer to as $P_{fwA \rightarrow B}(y|x)$ in the model overview.

It is not, however, perfect. GIZA builds several models, some quite complex, yet it does not use all the information available to it, notably the lexical similarity between the languages. As far as GIZA is concerned, each word it sees in either language is a completely opaque symbol.

Furthermore, GIZA tries to map words (especially rare ones) into other words if possible, even if the sentence has no direct translation for the word in question. In the case where the sentences are not literal word-for-word translations, but are rephrasings or idioms, this is perhaps not unreasonable

for an MT system, but it is unacceptable for us. For example, if English expression *red herring* is rendered into Russian as a non-idiomatic expression literally meaning *distracting maneuver* (which is what the dictionary claims it should be translated as), we do not want to link *red* with *distracting*, and *herring* with *maneuver* in our dictionary, despite the fact that a phrase-based MT system might find this very useful.

These problems are addressed by using other models, described in the following sections.

4.3.2 GIZA (backward)

In the previous section we discussed using GIZA to try to optimize $P(L_B|L_A)$. It is, however, equally reasonable to try to optimize $P(L_A|L_B)$ instead. If we do so, we can obtain $P_{fwB \rightarrow A}(x|y)$ that produces maximal probability for $P(L_A|L_B)$. We, however need a model of $P_{A \rightarrow B}(y|x)$. This is easily obtained by using Bayes' rule:

$$P_{bwA \rightarrow B}(y|x) = \frac{P_{fwB \rightarrow A}(x|y)P_B(y)}{P_A(x)}$$

which requires us to have $P_B(y)$ and $P_A(x)$. These models can be estimated directly from L_B and L_A , by using maximum likelihood estimators:

$$P_A(x) = \frac{\sum_i \delta(x_i, x)}{n}$$

and

$$P_B(y) = \frac{\sum_i \delta(y_i, y)}{m}$$

where $\delta(x, y)$ is the Kronecker's delta function, which is equal to 1 if its arguments are equal, and to 0 otherwise.

4.3.3 Character-based model

This and the following models all rely on having a model of $P_{A \rightarrow B}(y|x)$ to start from. In practice it means that this component is estimated following the previous components and uses the models they provide as a starting point.

The basic idea behind this model is that in related languages words are also related. If we have a model $P_c(b|a)$ of translating character a in language A into character b in language B, we can define the model for translating entire words.

We can define the model for translating word x in language A into a word y (any sequence of valid characters) in language B as follows. Let n be the length of word x and m be the length of word y . Then:

$$P_{uchar}(y|x) = P_{uchar_fixlen}(y|x, m)P_{length}(m|x)$$

We make a further simplifying assumption that in fact $P_{length}(m|x) = P_{length}(m|n)$, i.e. that the length of the translated word depends only on the length of the source word, and not on the word itself. The resulting model is easily estimated directly from the raw data.

The first model, $P_{uchar_fixlen}(y|x, m)$ is harder to estimate.

First, let us define a model for translating words of the same length. Let word x in language A consist of characters x_1 through x_n , and word y in language B consist of characters y_1 through y_m and $m = n$. Then,

$$P_{uchar_fixlen}(y|x, m, \text{if } m=n) = \prod_{i=1}^n P_c(y_i|x_i)$$

In other words, we try to match up the corresponding characters.

Let y^j be word y with j 's character removed. Let us now consider the case when $m > n$. We define:

$$P_{uchar_fixlen}(y|x, m, \text{if } m>n) = \sum_{i=1}^m \frac{1}{m} P_{uchar_fixlen}(y^i|x, m-1)$$

Since y^i has length $m-1$ it is now either the same length as x , in which case we are done, or not, in which case the formula above goes into recursion, for as many as $m-n$ steps. Essentially, it removes any $m-n$ characters from y and tries to match the resulting word against x , in the same way as does the equal-length algorithm above.

Similarly, if $n > m$:

$$P_{uchar_fixlen}(y|x, m, \text{if } m<n) = \sum_{i=1}^n \frac{1}{n} P_{uchar_fixlen}(y|x^i, m)$$

This removes characters from x instead of y , but otherwise is exactly the same.

It is easy to see that this is a valid probability model over all sequences of characters, if $P_{length}(m|n)$ is a valid model. However, y is not a random sequence of characters, but a word in language B , moreover, it is a word that can serve as a potential translation of word x . So, to define a proper distribution over words y given a word x and a set of possible translations of x , $T(x)$

$$P_{char}(y|x) = P_{uchar}(y|x, y \in T(x)) = \frac{P_{uchar}(y, y \in T(x)|x)}{\sum_{y' \in T(x)} P_{uchar}(y'|x)}$$

This is the complete definition of P_{char} , except for the fact that we are implicitly relying upon the character-mapping model, P_c , which we need to obtain somehow. To obtain it, we rely upon GIZA again. As we have seen, GIZA can find a good word-mapping model if it has a bitext to work from. If we have a $P_{A \rightarrow B}$ word-mapping model of some sort, it is equivalent to having a parallel bitext with words y and x treated as a sequence of characters, instead of indivisible tokens.

Each (x, y) word pair would occur $P_{A \rightarrow B}(x, y)$ times in this corpus. GIZA would then provide us with the P_c model we need, by optimizing the probability of language B part of the model given the language A part.

4.3.4 Prefix Model

This model and the model that follows are built on the same principle. Let there be a function $f : A \rightarrow C_A$ and a function $g : B \rightarrow C_B$. These functions group words in A and B into some finite set of classes. If we have some $P_{A \rightarrow B}(y|x)$ to start with, we can define

$$P_{fgA \rightarrow B}(y|x) = P(y|g(y)) P(g(y)|f(x)) P(f(x)|x)$$

The last term is equal to 1, since $f(x)$ uniquely depends on x . The first term, by the definition of conditional probability and the above definitions is

$$P(y|g(y)) = \frac{P(y, g(y))}{P(g(y))} = \frac{P(y)}{\sum_{y': g(y')=g(y)} P(y')}$$

and the second term, by the same principles

$$P(g(y)|f(x)) = \frac{P(g(y), f(x))}{P(f(x))} = \frac{\sum_{x':f(x')=f(x)} \sum_{y':g(y')=g(y)} P(x', y')}{\sum_{x':f(x')=f(x)} P(x')}$$

and so finally we obtain:

$$P_{fgA \rightarrow B}(y|x) = P(y) \frac{\sum_{x':f(x')=f(x)} \sum_{y':g(y')=g(y)} P(x', y')}{\left(\sum_{x':f(x')=f(x)} P(x')\right) \left(\sum_{y':g(y')=g(y)} P(y')\right)}$$

For the prefix model, we rely upon the following idea: words that have a common prefix often tend to be related. Related words probably should translate as related words in the other language as well. In other words, we are trying to capture word-level semantic information. So we define the following set of f and g functions:

$$f_n(x) = \text{prefix}(x, n)$$

$$g_m(y) = \text{prefix}(y, m)$$

where n and m are free parameters, whose values we will determine later. We therefore define $P_{prefA \rightarrow B}$ as P_{fg} with f and g specified above.

4.3.5 Suffix Model

Similarly to a prefix model mentioned above, it is also useful to have a suffix model. Words that have the same suffixes are likely to be in the same grammatical case, or to share some morphological feature which may persist across languages. In either case, if a strong relationship exists between the resulting classes, this relationship provides good evidence to give higher likelihood to the word belonging to these classes.

The functions f and g are defined based on a set of suffixes S_A and S_B which are learned automatically. $f(x)$ is defined as the longest possible suffix of x that is in the set S_A , and g is defined similarly, for S_B .

The sets S_A and S_B are built as follows. We start with all one-character suffixes and add them to the list. This ensures that each word will match at least one suffix.

Now, let w be some word, n be its length, and w^i be the i 'th character of that word, counting from the end, so that w^1 is the last character, w^2 is the next to last, and so on. Also, let $C(c, i)$ be the number of word types that have character c in the i 'th position from the end and $C_s(suf)$ be the number of word types ending in suf . Let N be the total number of word types. We want to choose two-letter suffixes (c^2c^1) such that

$$\frac{C_s(c^2c^1)}{N} > k \frac{C(c^2, 2)}{N} \frac{C(c^1, 1)}{N}$$

or

$$C_s(c^2c^1) > k \frac{C(c^2, 2)C(c^1, 1)}{N}$$

where k is a free parameter of the model. This allows us to pick suffixes that occur significantly more often than could be expected based on letter frequencies.

Once all two-letter suffixes are added, we proceed to add three-letter suffixes, using a similar approach. Let ($c^3c^2c^1$) be such a suffix. We add it to the list if

$$C_s(c^3c^2c^1) > k \frac{C(c^3, 3)C_s(c^2c^1)}{N}$$

4.3.6 Constituency Model

If we had information about constituent boundaries in either language, it would have been useful to make a model favoring alignments that do not cross constituent boundaries. We do not have this information at this point. However, we can assume that any sequence of three words is a constituent of sorts, and build a model based on that assumption.

As before, let $L_A = (x_i)_{i=1\dots n}$ and $L_B = (y_i)_{i=1\dots m}$. Let us define as $C_A(i)$ a triple of words (x_{i-1}, x_i, x_{i+1}) and as $C_B(j)$ a triple (y_{j-1}, y_j, y_{j+1}) . If we have some model $P_{A \rightarrow B}$, we can define

$$P_{C_A \rightarrow C_B}(j|i) = \frac{P_{A \rightarrow B}(y_{j-1}|x_{i-1})P_{A \rightarrow B}(y_j|x_i)P_{A \rightarrow B}(y_{j+1}|x_{i+1})}{\sum_{k=1}^m P_{A \rightarrow B}(y_{k-1}|x_{i-1})P_{A \rightarrow B}(y_k|x_i)P_{A \rightarrow B}(y_{k+1}|x_{i+1})}$$

and

$$\begin{aligned}
P_{consA \rightarrow B}(y|x) &= \sum_{i=1}^n \sum_{j=1}^m P(y|C_B(j))P_{C_A \rightarrow C_B}(j|i)P(C_A(i)|x) \\
&= \sum_{i:x_i=x} \sum_{j=1}^m P(y|C_B(j))P_{C_A \rightarrow C_B}(j|i) \\
&= \frac{1}{\sum_{j=1}^m \delta(y_j, y)} \sum_{i:x_i=x} \sum_{j:y_i=y} P_{C_A \rightarrow C_B}(j|i)
\end{aligned}$$

Now we can start by learning $P_{C_A \rightarrow C_B}(j|i)$ from the GIZA model, and then build a model $P_{consA \rightarrow B}(y|x)$ based on that model.

4.4 Building the Joint Model

It is our goal to produce a joint word translation model over all the languages we use, relying upon the parameterized pairwise models we built in the preceding sections. More formally, we have a set of languages $\{L_i\}_{i=1}^n$ and a set of pairwise models, $\{P(x, y|\theta_{ij})\}_{x \in L_i, y \in L_j, i \neq j}$ where θ_{ij} are the free parameters of the pairwise model for languages L_i and L_j . We want to obtain a model $P(L_1, \dots, L_n)$ that is most consistent with all of the pairwise models under the best possible setting of the θ 's.

It is easy to see that if we have at least three languages to work with, we will in effect have multiple ways of estimating the same model, and thus can find a good value for the θ 's.

To solve this problem properly, let us consider a slightly simpler and more general setting.

4.4.1 Combining Models of Hidden Data

Let X be a random variable with distribution $P_{\text{true}}(x)$, such that no direct observations of it exist. However, we may have some indirect observations of X and have built several models of X 's distribution, $\{P_i(x|\theta_i)\}_{i=1}^n$, each parameterized by some parameter vector θ_i . Our goal is to find a good estimate of $P_{\text{true}}(x)$.

The main idea is that if some P_i and P_j are close (by some measure) to P_{true} , they have to be close to each other as well. We will therefore make the assumption that if some models of X are close to each other they are also close to the true distribution. Moreover, we would like to set the parameters θ_i in such a way that $P(x_i|\theta_i)$ is as close to the other models as possible.

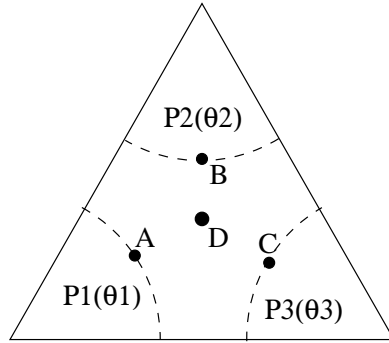


Figure 4.1: Combining imperfect models

For a more intuitive view of this concept consider Figure 4.4.1. The area inside of the triangle represents the model space. The dashed lines represent the more limited model space for some models we may have. The ideal model is at point D, and is not achievable by any of our models. Our problem is, first, to find the optimal parameter setting for each of our models (i.e. points A, B, C), and second, to find the point D, if possible. Our observation is that if our models are good enough, point A is not only the closest parameter setting for model 1 to the true model, but also closest to the models 2 and 3, especially to their best points: B and C. Furthermore, point D is the point that is closest to points A, B, and C. In other words, we are looking for points A, B, C, and D, such that A belongs to model 1, B to model 2, C to model 3, and D can be any point, such that the sum $|AD| + |BD| + |CD|$ is minimal.

More formally, this leads us to look for an estimate that is as close to all of our models as possible, under the optimal values of θ_i 's, or:

$$P_{\text{est}} = \arg \min_{\hat{P}(X)} \min_{\theta_1} \dots \min_{\theta_n} d(\hat{P}(X), P_1(X|\theta_1), \dots, P_n(X|\theta_n))$$

where d measures the distance between \hat{P} and all the P_i under the parameter setting θ_i . Since we have no reason to prefer any of the P_i , we choose the following symmetric form for d :

$$d(\hat{P}(X), P_1(X|\theta_1), \dots, P_n(X|\theta_n)) = \sum_{i=1}^n D(\hat{P}(x) || P_i(X|\theta_i))$$

where D is a reasonable measure of distance between probability distributions. The most appropriate

and the most commonly used measure in such cases in the Kullback-Leibler divergence, also known as relative entropy:

$$D(p||q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}$$

It turns out that it is possible to find the optimal \hat{P} under these circumstances. Taking a partial derivative of d over $\hat{P}(\bar{x})$ (where \bar{x} is fixed) with the Lagrange multiplier to insure that $\sum_{x \in X} \hat{P}(x) = 1$, we obtain:

$$\left(\sum_{i=1}^n d(\hat{P}(X), P_i(X|\theta_i)) \right)' + \left(\lambda \left(\sum_{x' \in X} \hat{P}(x') \right) - 1 \right)' = 0$$

The derivative over $\hat{P}(\bar{x})$ of the second part is simply λ . For the first part, taking the derivative of each d term:

$$\begin{aligned} d'(\hat{P}(X), P_i(X|\theta_i)) &= \left(\sum_{x \in X} \hat{P}(x) \log \frac{\hat{P}(x)}{P_i(x|\theta_i)} \right)' \\ &= \left(\hat{P}(\bar{x}) \log \frac{\hat{P}(\bar{x})}{P_i(\bar{x}|\theta_i)} \right)' \\ &= \left(\hat{P}(\bar{x}) \log \hat{P}(\bar{x}) \right)' - \left(\hat{P}(\bar{x}) \log P_i(\bar{x}|\theta_i) \right)' \\ &= 1 + \log \hat{P}(\bar{x}) - \log P_i(\bar{x}|\theta_i) \\ &= \log \frac{\hat{P}(\bar{x})}{\log P_i(\bar{x}|\theta_i)} + 1 \end{aligned}$$

And we, therefore, obtain the following constraint:

$$\sum_{i=1}^n \left(\log \frac{\hat{P}(\bar{x})}{P_i(\bar{x}|\theta_i)} + 1 \right) - \lambda = 0$$

or, moving constants and exponentiating,

$$\prod_{i=1}^n \frac{\hat{P}(\bar{x})}{P_i(\bar{x}|\theta_i)} = e^{\lambda-n}$$

finally solving for $\hat{P}(\bar{x})$:

$$\hat{P}(\bar{x}) = e^{\frac{\lambda-n}{n}} \prod_{i=1}^n P_i(\bar{x}|\theta_i)^{1/n}$$

The first expression on the right includes λ which is a normalizing constant. Since $\sum_{x \in X} \hat{P}(x) = 1$, we find that

$$e^{\frac{\lambda-n}{n}} = \frac{1}{\sum_{x \in X} \prod_{i=1}^n P_i(x|\theta_i)^{1/n}}$$

Therefore, we eliminate the lambda, and finally obtain, for all x :

$$\hat{P}(x) = \frac{\prod_{i=1}^n P_i(x|\theta_i)^{1/n}}{\sum_{x' \in X} \prod_{i=1}^n P_i(x'|\theta_i)^{1/n}}$$

Substituting this value into the expression for function d , we obtain the following distance measure d_{nohat} between the P_i 's:

$$\begin{aligned} d_{\text{nohat}}(P_1(X|\theta_1), \dots, P_n(X|\theta_n)) &= \min_{\hat{P}} d(\hat{P}, P_1(X|\theta_1), \dots, P_n(X|\theta_n)) \\ &= \sum_{i=1}^n \sum_{x \in X} \hat{P}(x) \log \frac{\hat{P}(x)}{P_i(x|\theta_i)} \\ &= \sum_{x \in X} \hat{P}(x) \sum_{i=1}^n \log \frac{\hat{P}(x)}{P_i(x|\theta_i)} \\ &= \sum_{x \in X} \hat{P}(x) \log \prod_{i=1}^n \frac{\hat{P}(x)}{P_i(x|\theta_i)} \\ &= \sum_{x \in X} \hat{P}(x) \log e^{\frac{\lambda-n}{n}} \\ &= \sum_{x \in X} \hat{P}(x) \log \frac{1}{\sum_{x \in X} \prod_{i=1}^n P_i(x|\theta_i)^{1/n}} \\ &= \log \frac{1}{\sum_{x \in X} \prod_{i=1}^n P_i(x|\theta_i)^{1/n}} \sum_{x \in X} \hat{P}(x) \\ &= \log \frac{1}{\sum_{x \in X} \prod_{i=1}^n P_i(x|\theta_i)^{1/n}} \\ &= -\log \sum_{x \in X} \prod_{i=1}^n P_i(x|\theta_i)^{1/n} \end{aligned}$$

This function has some interesting properties. It is a generalization of the well-known Bhattacharyya distance for two distributions (Bhattacharyya, 1943):

$$b(p, q) = \sum_i \sqrt{p_i q_i}$$

It is easy to see that our distance d is (in the case of two variables) simply $-\log b(P_1(x|\theta_1), P_2(x|\theta_2))$.

Also, by geometric-arithmetic means inequality,

$$\sum_{x \in X} \prod_{i=1}^n P_i(x|\theta_i)^{1/n} \leq \sum_{x \in X} \frac{\sum_{i=1}^n P_i(x|\theta_i)}{n} = 1$$

with equality only when all $P_i(x|\theta_i)$ are equal to each other for each x . This means that d is always greater or equal to 0, except when all of its arguments have the same distribution. This is, of course, precisely how the sum of Kullback-Leibler divergences should behave.

These results suggest the following algorithm to optimize d (and d_{nohat}):

- Set all θ_i randomly
- Repeat until change in d is very small:
 - Compute \hat{P} according to the above formula
 - For i from 1 to n
 - * Set θ_i in such a way as to minimize $D(\hat{P}(X)||P_i(X|\theta_i))$
 - Compute d according to the above formula

Each step of the algorithm minimizes d . It is also easy to see that minimizing $D(\hat{P}(X)||P_i(X|\theta_i))$ when \hat{P} is fixed involves minimizing two terms, one of which, $\sum_{x \in X} \hat{P}(x) \log \hat{P}(x)$ is fixed, and minimizing the other: $-\sum_{x \in X} \hat{P}(x) \log P_i(x|\theta_i)$ is the the same as setting the parameters θ_i in order to maximize $\prod_{x \in X} P_i(x|\theta_i)^{\hat{P}(x)}$. The latter problem can be interpreted as maximizing the probability under P_i of a corpus in which word x appears $\hat{P}(x)$ times. In other words, we are now optimizing $P_i(X)$ given an observed corpus of X , which is a much easier problem. In many types of models for P_i the Expectation-Maximization algorithm (Hartley, 1958) is able to solve this problem.

4.4.2 Combining Pairwise Models

Following the ideas outlined in the previous section, we can find an optimal $P(L_1 \dots L_n)$ if we are given several models $P_j(L_1 \dots L_n|\theta_j)$. Instead, we have a number of pairwise models. Depending on which independence assumptions we make, we can define a joint distribution over all the languages in various ways. For example, let t be a (directed) tree over nodes $1 \dots n$, and let us refer to its root as t_{root} and for any non-root node x , we refer to its parent as $t(x)$. We can define the following model:

$$P_t(x_1 \dots x_n)_{x_i \in L_i} = P(x_{t_{root}}) \prod_{i \neq t_{root}} P(x_i|x_{t(i)})$$

There are n^{n-1} different trees and therefore models of this sort (although some are equivalent), each involving $n - 1$ pairwise models and one model of word distribution in a single language, which

can be estimated directly and is not parameterized. We can now choose a set of trees T (perhaps one that is symmetric over all the languages), and apply the result from the previous section. For simplicity, let us do so with three languages, A , B , and C , and the smallest symmetric set of trees:

$$\begin{aligned} P_1(A, B, C) &= P(A|B)P(B|C)P(C) = \frac{P(A,B)P(B,C)}{P(B)} \\ P_2(A, B, C) &= P(C|A)P(A|B)P(B) = \frac{P(A,C)P(A,B)}{P(A)} \\ P_3(A, B, C) &= P(B|C)P(C|A)P(A) = \frac{P(A,C)P(B,C)}{P(C)} \end{aligned}$$

and

$$\begin{aligned} d'(\hat{P}, P_1, P_2, P_3) &= D(\hat{P}||P_1) + D(\hat{P}||P_2) + D(\hat{P}||P_3) \\ &= -3H(\hat{P}) + H(\hat{P}, P_1) + H(\hat{P}, P_2) + H(\hat{P}, P_3) \\ &= -3H(\hat{P}) + H(\hat{P}(A, B), P(A, B)) + H(\hat{P}(B, C), P(B, C)) - H(\hat{P}(B), P(B)) \\ &\quad + H(\hat{P}(A, C), P(A, C)) + H(\hat{P}(A, B), P(A, B)) - H(\hat{P}(A), P(A)) \\ &\quad + H(\hat{P}(B, C), P(B, C)) + H(\hat{P}(A, C), P(A, C)) - H(\hat{P}(C), P(C)) \\ &= 2H(\hat{P}(A, C), P(A, C)) + 2H(\hat{P}(A, B), P(A, B)) + 2H(\hat{P}(B, C), P(B, C)) \\ &\quad - 3H(\hat{P}) - H(\hat{P}(A), P(A)) - H(\hat{P}(B), P(B)) - H(\hat{P}(C), P(C)) \end{aligned}$$

where $H(\cdot)$ is entropy, $H(\cdot, \cdot)$ is cross-entropy, and $\hat{P}(A, B)$ means \hat{P} marginalized to variables A, B . The last 3 cross-entropy terms involve monolingual models which are not parameterized and are obtained directly from the data by maximum likelihood estimation. The entropy term does not involve any of the pairwise distributions. Therefore, to maximize d' , we need to maximize each of the bilingual cross-entropy terms.

This means we can apply the algorithm from the previous section with a small modification:

- Set all θ_{ij} (for each language pair i, j) randomly
- Repeat until change in d is very small:
 - Compute P_i for $i = 1 \dots k$ where k is the number of language dependency trees we have chosen
 - Compute \hat{P} from $\{P_i\}$

- For i, j such that $i \neq j$
 - * Marginalize \hat{P} to (L_i, L_j)
 - * Set θ_{ij} in such a way as to minimize $D(\hat{P}(L_i, L_j) || P_i(L_i, L_j | \theta_{ij}))$
- Compute d according to the above formula

Most of the *theta* parameters in our models can be set by performing EM, and the rest are discrete with only a few choices and can be maximized over by trying all combinations of them.

4.5 Evaluation

The output of the system so far is a multilingual word translation model. We will evaluate it by producing a tri-lingual dictionary (Russian-Ukrainian-Belorussian), picking a highest probability translation for each word, from the corresponding Bibles. Unfortunately, we do not have a good hand-built tri-lingual dictionary to compare it to, but only one good bilingual one, Russian-Ukrainian⁷. We will therefore take the Russian-Ukrainian portion of our dictionary and compare it to the hand-built one.

Our evaluation metric is the number of entries that match between these dictionaries. If a word has several translations in the hand-built dictionary, matching any of them counts as correct. It is worth noting that for all the dictionaries we generate, the total number of entries is the same, since all the words that occur in the source portion of the corpus have an entry. In other words, precision and recall are proportional to each other and to our evaluation metric.

Not all of the words that occur in our dictionary occur in the hand-built dictionary and vice versa. An absolute upper limit of performance, therefore, for this evaluation measure is the number of left-hand-side entries that occur in both dictionaries.

In fact, we cannot hope to achieve this number. First, because the dictionary translation of the word in question might never occur in the corpus. Second, even if it does, but never co-occurs in the

⁷The lack of such dictionaries is precisely *why* we do this work.

Table 4.1: Evaluation for Russian-Ukrainian (with Belorussian to tune)

Stage	Pair	Joint
Forward (baseline)	62.3%	71.7%
Forward+chars	77.1%	84.2%
Forward+chars+backward	81.3%	84.1%
Fw+chars+bw+prefix	83.5%	84.5%
Fw+chars+bw+prefix+suffix	84.5%	85%
Fw+chars+bw+pref+suf+const	84.5%	85.2%
“Oracle” setting for λ 's	84.6%	

same sentence as its translation, we will not have any basis to propose it as a translation.⁸ Therefore we have a “achievable upper limit”, the number of words that have their “correct” translation co-occur at least once. We will compare our performance to this upper limit.

Since there is no manual tuning involved, we do not have a development set and use the whole bible for training (the dictionary is used as a test set, as described above).

We evaluate the performance of the model with just the GIZA component as the baseline, and add all the other components in turn. There are two possible models to evaluate at each step. The pairwise model is the model given in equation 4.1 under the parameter setting given by Algorithm 2, with Belorussian used as a third language. The joint model is the full model over these three languages as estimated by Algorithm 2. In either case we pick a highest probability Ukrainian word as a translation of a given Russian word.

The results for Russian-Ukrainian bibles are presented in Table 1. The “oracle” setting is the setting obtained by tuning on the test set (the dictionary). We see that using a third language to tune works just as well, obtaining the true global maximum for the model. Moreover, the joint model (which is more flexible than the model in Equation 4.1) does even better. This was unexpected for us, because the joint model relies on three pairwise models equally, and Russian-Belorussian and Ukrainian-Belorussian models are bound to be less reliable for Russian-Ukrainian evaluation. It appears, however, that our Belorussian bible is translated directly from Russian rather than original

⁸Strictly speaking, we might be able to infer the word’s existence in some cases, by performing morphological analysis and proposing a word we have not seen, but this seems too hard at the moment.

Table 4.2: Evaluation for Russian-Ukrainian (with Belorussian and Polish)

Tuned by	Pair	Joint
Belorussian (prev. table)	84.5%	85.2% &
Polish	84.6%	78.6%
Both	84.5%	85.2%
“Oracle” tuning	84.5%	

languages, and parallels Russian text more than could be expected.

To ensure our results are not affected by this fact we also try Polish separately and in combination with Belorussian (i.e. a model over 4 languages), as shown in Table 2.

These results demonstrate that the joint model is not as good for Polish, but it still finds the optimal parameter setting. This leads us to propose the following extension: let us marginalize joint Russian-Ukrainian-Belorussian model into just Russian-Ukrainian, and add this model as yet another component to Equation 4.1. Now we cannot use Belorussian as a third language, but we can use Polish, which we know works just as well for tuning. The resulting performance for the model is **85.7%**, our best result to date.

Chapter 5

Building tools

5.1 Introduction

Our ultimate goal is to build parsers in a semi-automated manner from several plaintext bibles we have available. In particular, we build parsers for all the languages we work on (Russian, Ukrainian, Belorussian, and Polish), but we will evaluate only for two of them (Russian and Ukrainian) for practical reasons.

This part of the project builds upon the prepared word-to-word translation models built in the first part, but also upon the availability of the Prague Dependency Treebank (Hajič et al., 2001) for Czech and the tools (parsers and taggers) based upon it. As described previously, we will parse and part-of-speech tag our target languages, based upon the Czech grammar and the set of parts of speech used by the PDT.

We proceed in the following manner:

1. Produce taggers
 - Use a Czech tagger to tag the Bible.
 - Transfer tag information into the target language using the dictionary

- Use the tagged Bible in the target language to produce a tagger for that language
2. Produce parsers (two approaches):
- Build a parser in the target language
 - Parse the Czech Bible
 - Transfer the information across words which are translations and obtain (partially) parsed Bible in the target language
 - Create a parser using the parsed Bible in the target language (along with POS tagger and dictionary)
 - Parse target language as if it were Czech
 - “Translate” a sentence into Czech
 - Parse it by the Czech parser
 - “Translate” the parse tree back into the target language

We then evaluate and compare the resulting tagger and parsers.

5.2 Building a tagger

Part of speech tagging is a relatively easy problem. It is well known that even a relatively small amount of supervised data is sufficient to achieve reasonable quality for many languages.

We first tag the Czech Bible using the hmm tagger which comes with the PDT (Hajič et al., 2001). We discard all information except the basic POS tag, like Noun or Verb.

The most obvious approach to build a tagger, which we will treat as a baseline, is as follows. For every word t on the target side, pick the best translation s on the source side according to our dictionary. Assign The part of speech of s to be the part of speech of t .

This actually is fairly reasonable for many words if our dictionary is good (and it is), but it also makes many mistakes which could be corrected.

Presumably, one of the reasons this approach makes mistakes is that the translations are sometimes bad, yet this algorithm is not using the quality of translations in its decisions. For example, if in a given sentence, *Moses* gets translated as *he*¹, it would be tagged as a pronoun. We know, however, that *he* is a bad translation for *Moses* (because our dictionary is probabilistic), but it so happens that there's no better translation among all the words in the given sentence on the target language side. This tells us two things. First, if we have other translations for *Moses* in different sentences, and most of these are proper nouns, we should trust those ones more than this one, and tag this instance as a proper noun as well. Second, when picking a subset of good translations for training, don't include this sentence.

We can do better for those words (and sentences) which we know to be better than average translations.

5.2.1 Finding good translations (no other data)

We need to have an automatic way to identify which sentences in the target language are highly parallel translations of the source sentences. One way to do so is to use the probability of the target sentence given the source under some model. The problem with this, however, is that we would need some alignment model and not just the word mapping model which we have. Instead, we simply use the idea that the sentences which have unambiguously translatable words are the ones that would have the best translations.

In particular, if a word s in the source language has (according to our word translation model) translations t_i with probabilities $P(t_i|s)$, we can define its entropy:

$$H(s) = - \sum_{i=1}^n P(t_i|s) \log P(t_i|s)$$

and for a source sentence $S = (s_1 \dots s_m)$ we can define the average entropy

$$H(S) = \frac{\sum_{i=1}^m H(s_i)}{m}$$

¹This is fairly common if one of the translations is less literal

Obviously, a sentence with low average entropy is much more likely to have a good translation. This turns out to be true during evaluation, and we can use $-H(S)$ as a scoring function to identify these sentences

5.2.2 Finding good translations (with other data)

On the other hand, if we had generated two different candidate taggers T_1 and T_2 for the target language, we could identify a set of good translations trivially, by defining a scoring function

$$f(S) = \frac{\sum_{i=1}^m \delta(T_1(s_i), T_2(s_i))}{m}$$

where

$$\delta(a, b) = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{if } a \neq b \end{cases}$$

In other words, we would pick sentences that taggers tend to agree on, as likely to be good translations, and a good subset to train the next candidate tagger from. This idea is inspired by co-training approaches (e.g. Yarowsky (1995)).

5.2.3 Training taggers

Let us suppose we have a good subset of sentences tagged, which we will treat as a gold standard. There is an extensive number of tools and approaches that can be trained on this subset and used to tag the full corpus. We pick one that is known to be quite good for POS tagging, called YamCha (Kudo and Matsumoto, 2003; Kudo and Matsumoto, 2001). This tool requires a feature set, and we use two different one at different stages. The smaller feature set contains:

- Whether the word is uppercase, lowercase, or punctuation (for current and two previous words)
- The 1, 2, and 3 letter suffixes (for current and two previous words)
- The word itself (for current and two previous words)

- The tag assigned by the system for two previous words (dynamically)

The idea is that the ending of a word is a good predictor of its part of speech, and capitalized words are usually nouns.

The larger set contains all these features, plus 1, 2, and 3 letter prefixes (for current and two previous words). The idea for this set is that at a later stage, when common words, at least, are already reliably tagged, some related forms of these words may be tagged with the same tag, based on their sharing a lemma with the common word.

5.2.4 The algorithm

We can now use a bootstrapping-like approach to build the tagger:

- Tag the target language bible by the source language tagger, call this stage 0 corpus/tagger
- Use the approach described in section 5.2.1 to choose a high-quality subset of sentences
- Train a target-language tagger on this subset (with stage 0 tags treated as true) with the smaller feature set, call this stage 1
- Use the approach described in section 5.2.2 and stage 0 and 1 taggers to choose a subset of sentences (non-intersecting with the previously chosen ones)
- Train a tagger on this subset (with stage 1 tags) with the smaller feature set, call this stage 2
- Use the approach described in section 5.2.2 and stage 1 and 2 taggers to choose a subset of sentences (non-intersecting with the previously chosen ones)
- Train a tagger on this subset (with stage 2 tags) with the larger feature set, call this stage 3
- If any word tokens for the same word type are tagged by multiple tags, change them to the most common tag for this word type, call this stage 4

Table 5.1: Tagging performance

Stage	Description	Accuracy
0	Only Czech tagger	75%
1	Tagger based on low-entropy subset	80%
2	Tagger based on subset with high agreement for 0 and 1	82%
3	Tagger based on subset with high agreement for 1 and 2 and more feats	83%
4	Tagger based on most common tag for word type (according to 3)	85%

The last step turns out to be useful because for Slavic languages most words have only one part of speech.

5.3 Evaluating a tagger

We have a part of speech tagger and a partial parser for Russian called *AOT* which is freely available online (Nozhov, 2003). We do not, however, have any taggers for the other languages, and we will only evaluate the tagger for Russian². We will treat this parser’s output as the gold standard, and compare our output to it. Where the AOT parser has multiple possible tags, matching either is considered correct, and we skip words for which AOT has no analysis.

In all cases we completely tag the Russian bible. We report the per-word tagging accuracy in Table 5.3.

5.3.1 Error analysis

Since we measure accuracy on per-token basis, a few very common misclassified words make a big difference. It turns out that several pronouns, especially *I* are misclassified as verbs. The reason for that is that an expression which might be rendered as *I am* in English becomes *I* in Russian, and *am* in Czech. This highlights a fundamental problem with our assumptions, so we cannot address this problem within our framework.

Most of the other problems seem to arise from rephrasing, where a word in one part of speech gets aligned with a word (perhaps a synonym or a related word) in a different part of speech.

²There’s no reason to believe that results for the other languages would differ.

5.4 Building a parser

It is not our goal to build a high-quality parser as such, but only to demonstrate how an existing high-quality parser may be adapted to a different language.

One way to use such a parser is, given a sentence in the target language, one may translate it into the source language, parse it, and transfer the information back. Another way is to translate the model that the parser uses into the target language and thus obtain a parser for the target language. We will refer to the former approach as the indirect parser, and to the latter as the direct parser. We try both approaches.

5.4.1 Building an indirect parser

The appeal of an indirect parser is that it requires no knowledge of how such parser operates. We can take any existing parser as a black box. The natural disadvantage is that we have no way of tuning the parser to the specific elements of the target language.

We use the unpublished Charniak parser for Czech as our high-quality parser. For comparison we also use a simple probabilistic context free grammar (PCFG) parser (for an early description of PCFG, see Jelinek et al. (1990), we use the implementation by Klein and Manning (2002)).

Since a parser is fixed, the only variation we deal with is how to transform the input and the output appropriately. We are essentially limited to word-to-word translation because our models are very simple. We do, however, have a choice of either taking one or multiple translations.

If we simply take the most probable translation from the target into the source language as the parser input, and the translation happens to be bad, we may do very poorly. If we take the top N translations, parse each, and recombine the results, we can expect to do better (or at least no worse), but it takes N times longer to parse each sentence.

Figure 5.1: A sample PCFG

S	→	NP VP	1.0
NP	→	N	0.9
NP	→	N NP	0.1
VP	→	V	0.2
VP	→	V NP	0.8
N	→	<i>John</i>	0.3
N	→	<i>Mary</i>	0.3
N	→	<i>Smith</i>	0.4
V	→	<i>loves</i>	0.4
V	→	<i>hates</i>	0.4
V	→	<i>wakes</i>	0.2

5.4.2 Building a direct parser

We may expect that it would be better to modify the model used by a parser to adapt it directly to the target language. It is, however, a hard problem, because the adaptation depends on the kind of model which this parser uses. This may limit us to simpler parsers. In particular, the model used by the Charniak parser is too complex for us to adapt automatically. We are, however, able to adapt the PCFG model.

A PCFG is a modification of a context free grammar which has a probability associated with each production rule. For example, a sample PCFG is given in Figure 5.4.2.

The PCFG model may be described as consisting of two parts: the grammar, which consists of rules containing only non-terminals, and the lexicon, which contains a large number of unary rules for producing terminals. We can make the assumption that the grammar derived from the source language is good enough for the target language. The lexicon, however, is not.

On the other hand, if we have a part-of-speech tagged corpus, we can easily obtain the lexicon from it. The maximum likelihood estimator for the probability of rule $T \rightarrow x$ for the tag T and word x is simply the number of times the word x occurs with tag T divided by the number of times tag T occurs in our corpus. Therefore, we are able to take the grammar trained on labeled source language trees and the lexicon derived from our POS-tagged target language data, and build a direct PCFG

Table 5.2: Parser for Russian

Czech parser	Type of transformation	Accuracy
Charniak	indirect, top 1 translation	54%
Charniak	indirect, top 10 translations	55%
PCFG	indirect, top 10 translations	43%
PCFG	direct	42%

parser for the target language.

5.4.3 Parsing results

We do not have access to a parser for any of the target languages. Therefore, we manually parse several hundred sentences and use these sentences as a test set. The parser produces a list of pairs: for every word in a sentence, the word it depends on (or NONE if the word is the root of the dependency tree). Our evaluation measure is simply the accuracy of this classification.

The results for Russian are given in Table 5.4.3.

These results demonstrate that direct and indirect methods seem to work equally well when both are available, but the original parser’s quality makes a big difference. Also, there is very little advantage in using more than one translation. This is actually good news, since indirect parsers are easy to create and using only one translation makes them very efficient.

5.4.4 Error analysis

It turns out that most of the sentences in the Bible have right-branching parse trees, presumably following original Hebrew order, while normal Russian (and Czech) parse trees are fairly balanced. This means that rule probabilities derived from a normal Czech corpus are wrong for this corpus. For example, the parser learns that two nouns in a row is probably a noun phrase, which is reasonable in an SVO language. However, the bible is written in essentially a VSO word order, which means that subject and object are next to each other, and the parser tends to combine them as a noun phrase.

Therefore, I believe that the actual parser performance on a different (normal) text would be higher.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this work we have demonstrated that, first, it is possible to build a high quality dictionary for multiple related languages, and, second, that having such a dictionary, allows us to create a high quality tagger, and gives us a reasonable parser to work from.

In the first part of the project, we discovered that having parallel text in more than two languages is very helpful and invented a novel method to exploit this fact. We also discovered that having features exploiting lexical and syntactic similarity between the languages helps during dictionary induction.

In the second part, we came up with a bootstrapping-like algorithm to create part of speech taggers for all of our target languages. We also tried several ways to adapt parsers to our target languages and discovered that an efficient approach involves simply translating sentences word-for-word into the source language to be parsed.

Overall, we show that it is possible to build tools for a number of related languages starting with only a parser for one of them and a commonly available multitext - the Bible. In practical terms, we can now create a parser and a tagger for any Slavic language (without being able to speak it) in

about a day, provided only with the Bible in that language.

6.2 Future Work

Let us now discuss some further work that can be done beyond the scope of this dissertation.

One of the objections to the usefulness of our approach is that parsing the Bible is not particularly useful. While certain syntactic relationships will be learned, they might not be applicable to parsing more normal text, mainly because of the large number of out-of-vocabulary items, but also because of stylistic peculiarities of the Bible. To address this problem it would be useful to see the above approach as a bootstrapping step. Having a Bible-based parser, tagger, and several dictionaries would be sufficient to mine the Web for parallel text in any language pair our system supports. Furthermore, even if the Bible is atypical in its vocabulary and style, we will have a grasp on most basic, short, frequent words and syntactic constructions. Longer words, on the other hand, are easier to learn automatically, when a related language with a large vocabulary is available. Such a project forms the most likely next step.

Another idea concerns applications. It would be useful to build a number of parsers for all related languages and then try using them for some task, such as text summarization. One obvious application is to turn around and check if they help with machine translation. Another domain where they could be used is the increasingly multilingual Web.

Furthermore, having a parser for a language goes a long way toward building a machine translation system for it. Having such a system would, conversely, help to build a better parser. This suggests an iterative improvement process, whether automated or semi-automated, that would lead to further improvements.

References

- The Central Intelligence Agency. 2004. The world factbook.
- S. B. Bernshtein. 1961. *Ocherk sravnitelnoi grammatiki slavianskikh iazykov*. Izd-vo Akademii Nauk SSSR, Moscow.
- A. Bhattacharyya. 1943. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.*, 35:99–109.
- P. F. Brown, J. Cocke, V. Della Pietra, S. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 6(2):79–85.
- P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.
- W. J. Entwistle and W. A. Morison. 1949. *Russian and the Slavonic Languages*. Faber and Faber Limited.
- D. Genzel. 2005. Inducing a multilingual dictionary from a parallel multitext in related languages. In *Proceedings of the HLT/EMNLP conference*.
- J. Hajic, J. Hric, and V. Kubon. 2000. Machine translation of very close languages. In *Proceedings of the 6th Applied Natural Language Processing Conference*.
- J. Hajič, E. Hajičová, P. Pajas, J. Panevová, P. Sgall, and B. Vidová-Hladká. 2001. Prague dependency treebank 1.0 (final production label). CDROM CAT: LDC2001T10., ISBN 1-58563-212-0.
- H. Hartley. 1958. Maximum likelihood estimation from incomplete data. *Biometrics*, 14:174–194.
- J. Herkel. 1826. *Elementa universalis linguae Slavicae*. Budapest.
- I. Hosek. 1907. *Grammatik der Neuslavischen Sprache*. Kremsier.
- M. Hucko. 2005. Slovio: simplified Slavic international language. URL: <http://www.slovio.com/>.
- F. Jelinek, J. D. Lafferty, and R. L. Mercer. 1990. Basic methods of probabilistic context free grammars. Research Report RC 16374 (#72684), IBM, Yorktown Heights, New York 10598.
- D. Klein and C. D. Manning. 2002. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, December.
- D. Klein and C. D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*.
- G. Kondrak. 2001. Identifying cognates by phonetic and semantic similarity. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics, Pittsburgh, PA*, pages 103–110.
- T. Kudo and Y. Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics, Pittsburgh, PA*.
- T. Kudo and Y. Matsumoto. 2003. Fast methods for kernel-based text analysis. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*.

- G. Mann and D. Yarowsky. 2001. Multipath translation lexicon induction via bridge languages. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics, Pittsburgh, PA*, pages 151–158.
- I. D. Melamed. 2000. Models of translational equivalence among words. *Computational Linguistics*, 26:221–249, June.
- I. D. Melamed. 2004. Statistical machine translation by parsing. In *Proceedings of the 42nd Meeting of Association for Computational Linguistics*.
- I. M. Nozhov. 2003. *Morfologicheskaya i sintakticheskaya obrabotka teksta (modeli i programmy)*. Ph.D. thesis, Russian State University for the Humanities. Code available at <http://www.aot.ru>.
- F. J. Och and H. Ney. 2000. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 440–447, Hongkong, China, October.
- F. J. Och and H. Ney. 2001. Statistical multi-source translation. In *Proceedings of MT Summit VIII*, pages 253–258.
- C. Pado and M. Lapata. 2005. Cross-linguistic projection of role-semantic information. In *Proceedings of HLT/EMNLP conference*.
- D. Smith and A. Smith. 2004. Bilingual parsing with factored estimation: Using english to parse korean. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.
- C. Xi and R. Hwa. 2005. A backoff model for bootstrapping resources for non-english languages. In *Proceedings of HLT/EMNLP conference*.
- D. Yarowsky, G. Ngai, and R. Wicentowski. 2001. Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of First International Conference on Human Language Technology Research (HLT)*.
- D. Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Meeting of the ACL, Cambridge, MA*, pages 189–196.