Abstract

Title of dissertation: MACHINE TRANSLATION BY PATTERN MATCHING

Adam Lopez, Doctor of Philosophy, 2008

Dissertation directed by: Professor Philip Resnik
Department of Linguistics and
Institute for Advanced Computer Studies

The best systems for machine translation of natural language are based on statistical models learned from data. Conventional representation of a statistical translation model requires substantial offline computation and representation in main memory. Therefore, the principal bottlenecks to the amount of data we can exploit and the complexity of models we can use are available memory and CPU time, and current state of the art already pushes these limits. With data size and model complexity continually increasing, a scalable solution to this problem is central to future improvement.

Callison-Burch et al. (2005) and Zhang and Vogel (2005) proposed a solution that we call *translation by pattern matching*, which we bring to fruition in this dissertation. The training data itself serves as a proxy to the model; rules and parameters are computed on demand. It achieves our desiderata of minimal offline computation and compact representation, but is dependent on fast pattern matching algorithms on text. They demonstrated its application to a common model based on the translation of contiguous substrings, but leave some open problems. Among these is a question: can this approach match the performance of conventional methods despite unavoidable differences that it induces in the model? We show how to answer this question affirmatively.

The main open problem we address is much harder. Many translation models are based on the translation of discontiguous substrings. The best pattern matching algorithm for these models is much too slow, taking several minutes per sentence. We develop new algorithms that reduce empirical computation time by two orders of magnitude for these models, making translation by pattern matching widely applicable. We use these algorithms to build a model that is two orders of magnitude larger than the current state of the art and substantially outperforms a strong competitor in Chinese-English translation. We show that a conventional representation of this model would be impractical. Our experiments shed light on some interesting properties of the underlying model. The dissertation also includes the most comprehensive contemporary survey of statistical machine translation.

MACHINE TRANSLATION BY PATTERN MATCHING

by

Adam David Lopez

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2008

Advisory Committee:
Professor Philip Resnik, Chair/Advisor
Professor David Chiang
Professor Bonnie Dorr
Professor David Fushman
Professor Douglas Oard

A portion of this work is based on: Statistical Machine Translation, in *ACM Computing Surveys*, 40(3), Aug. © Association for Computing Machinery, 2008.
http://doi.acm.org/10.1145/1380584.1380586

For Jennifer.

Acknowledgements

I am deeply indebted to Philip Resnik for guiding me through this process. This dissertation is a testament to his faith in me, which was often greater than my own. I would not have found my own path if he had not given me the freedom to get lost, and direction when I stayed lost for too long. His enthusiasm and encouragement were crucial to my success, and I can only hope that I one day inspire someone else as much as he has inspired me.

I am also grateful for the guidance of my committee—David Chiang, Bonnie Dorr, Doug Oard, and David Fushman. Much of the work described herein was influenced in one way or another by my interactions with David Chiang. In 2005, I asked him whether the suffix array trick would work with hierarchical models, and he told me he didn't think so. It's about the only time I've ever known him to be wrong about anything, and even that took over two years and an enormous amount of work to prove. David also gave me very constructive early feedback that shaped the survey chapter, and it is not hyperbole to say that I learned much of what I know about statistical translation just from reading his elegant code. Bonnie Dorr gave me tremendously detailed and helpful feedback on several drafts of the dissertation, and Doug Oard directly challenged me to broaden its scope and vision. The final result is much stronger than it would have been without them. Finally, David Fushman gamely served on my committee and engaged with me throughout this process even though his own work is very far afield from mine. Together with Philip, they held this dissertation to a high standard. Any errors that remain are my own.

Several people in the Department of Computer Science and UMIACS helped along the way. I am grateful to Bill Gasarch for serving on my proposal committee, but more importantly for his unvarnished advice. Mihai Pop graciously allowed me to audit and actively engage with his bioinformatics seminar, where I found most of the tools that I needed to solve David's challenge. Fritz McCall went above and beyond the call of the duty assisting me through my forays into network programming.

I learned most of what I know about natural language processing from the present and past students, postdoctoral researchers, and visitors who have been my colleagues in the CLIP lab. Many ideas were developed and refined in discussions with Fazil Ayan, Dina Demner-Fushman, Mona Diab, Nizar Habash, Tim Hunter, Okan Kolak, Nitin Madnani, Smaranda Muresan, Christof Monz, Mike Nossal, Michael Subotin, Davic Zajic, and Dan Zeman. I owe special thanks to Chris Dyer for all of the brainstorming, critiquing, and commiserating during the course of writing this dissertation. Most especially, I thank Rebecca Hwa for being my mentor and my friend.

Outside of the lab, I thank Bill Woods for my earliest exposure to NLP research, and I'm enormously grateful to Philipp Koehn—first for offering me a fantastic job that I'm very excited about, and then for waiting on me while I completed this obligation.

I had a lot of fun in grad school. This I owe to the friends I spent time with outside of the lab: Laura Bright, Gene Chipman, Ric Crabbe, Shaun Gittens, Debbie Heisler, Omer Horvitz, Dave Hovemeyer, Doc Hunter, Aram Khalili, Jordan Landes, Jeremy Manson, Jan Neumann, Brian Postow, Reiner Schulz, Jaime Spacco, Kate Swope, and Chadd Williams. Thank you all for keeping me sane.

My family has been supportive of me throughout this endeavor, even though it seemed prolonged and bewildering when beheld from afar. I am thankful to Lisa and Josh Friedman, and my parents Richard and Edith Lopez.

Most importantly, Jennifer Baugher has filled my days, nights, and dreams with love throughout my time here. She has been with me for every moment of triumph and despair, through count-

less adventures and many bottles of wine. I know I could accomplish anything when she is with me. Thank you for everything, for ever.

This research was supported in part by NSA Contract RD-02-5700, ONR MURI Contract FCPO.810548265 and the GALE program of the Defense Advanced Research Projects Agency, Contract No. HR0011-06-2-001. Any opinions, findings, conclusions or recommendations expressed in this work are those of the author and do not necessarily reflect the view of DARPA. Additional support was provided by the EuroMatrix project funded by the European Commission (6th Framework Programme).

Contents

Dedication ii

Acknowledgements iii

Contents vi

List of Tables ix

List of Figures x

1 Introduction 1
- 1.1 Outline of the Dissertation . . . 2
- 1.2 Research Contributions . . . 2

2 A Survey of Statistical Machine Translation 4
- 2.1 Previous Work . . . 4
- 2.2 Background and Context . . . 5
  - 2.2.1 Formal Description . . . 6
- 2.3 Modeling Part I: Translational Equivalence . . . 8
  - 2.3.1 Finite-State Transducer Models . . . 8
    - 2.3.1.1 Word-Based Models . . . 9
    - 2.3.1.2 Phrase-Based Models . . . 10
  - 2.3.2 Synchronous Context-Free Grammar Models . . . 12
    - 2.3.2.1 Bracketing Grammars . . . 15
    - 2.3.2.2 Syntax-Based Translation . . . 16
    - 2.3.2.3 Hierarchical Phrase-Based Translation . . . 17
  - 2.3.3 Other Models of Translational Equivalence . . . 17
    - 2.3.3.1 More Powerful Formalisms . . . 18
    - 2.3.3.2 Syntactic Phrase-Based Models . . . 18
    - 2.3.3.3 Alternative Linguistic Models . . . 18
- 2.4 Modeling Part II: Parameterization . . . 19
  - 2.4.1 Generative Models . . . 20
    - 2.4.1.1 Language Models . . . 21
    - 2.4.1.2 Translation Models . . . 22
  - 2.4.2 Discriminative Models . . . 24
- 2.5 Parameter Estimation . . . 26
  - 2.5.1 Parameter Estimation in Generative Models . . . 26
    - 2.5.1.1 Learning Word Translation Probabilities . . . 26
    - 2.5.1.2 Learning Phrase Translation Probabilities . . . 28
    - 2.5.1.3 Learning Parameters of Generative SCFG Models . . . 28
  - 2.5.2 Interlude: Word Alignment . . . 28
    - 2.5.2.1 Formal Definition . . . 29
    - 2.5.2.2 Asymmetric Models . . . 29
    - 2.5.2.3 Symmetric Alignment Models . . . 31
    - 2.5.2.4 Supervised Learning for Alignment . . . 31
    - 2.5.2.5 Evaluation of Word Alignment . . . 32

2.5.3 Estimation in Log-Linear Models . . . 33
2.5.3.1 Minimum Error-Rate Training . . . 33
2.5.3.2 Purely Discriminative Training . . . 36
2.6 Decoding . . . 36
2.6.1 FST Decoding . . . 37
2.6.1.1 Optimality and Pruning . . . 39
2.6.1.2 Greedy Decoding . . . 39
2.6.2 SCFG Decoding . . . 39
2.6.3 Reranking . . . 41
2.7 Evaluation . . . 41
2.8 Current Directions and Future Research . . . 43
2.9 Conclusions . . . 44

3 Machine Translation by Pattern Matching 46
3.1 Baseline Model . . . 47
3.2 Phrase Tables . . . 48
3.3 Translation by Pattern Matching . . . 52
3.3.1 Pattern Matching and Suffix Arrays . . . 54
3.3.2 Source-Driven Phrase Extraction and Scoring . . . 55
3.4 Bringing Performance up to the State of the Art . . . 59
3.5 Results . . . 60
3.5.1 Baseline System Results . . . 60
3.5.2 Translation by Pattern Matching Results . . . 60
3.5.3 Analysis of Memory Use . . . 61
3.6 Conclusions . . . 62

4 Pattern Matching for Phrases with Gaps 63
4.1 Hierarchical Phrase-Based Translation (Redux) . . . 64
4.2 The Pattern Matching Problem for Hierarchical Phrases . . . 66
4.2.1 Baseline Algorithm . . . 66
4.2.2 Analysis . . . 69
4.3 Solving the Pattern Matching Problem for Hierarchical Phrases . . . 70
4.3.1 Efficient Enumeration . . . 70
4.3.1.1 The Zhang-Vogel Algorithm . . . 70
4.3.1.2 Prefix Trees and Suffix Links . . . 71
4.3.1.3 Special Cases for Phrases with Gaps . . . 73
4.3.1.4 Putting It All Together: Prefix Tree Generation Algorithm . . . 75
4.3.1.5 The Basic QUERY Algorithm . . . 75
4.3.1.6 Precomputation of Inverted Indices for Frequent Subpatterns . . . 82
4.3.2 Faster Pattern Matching for Individual Query Patterns . . . 83
4.3.2.1 Fast Intersection via Double Binary Search . . . 83
4.3.2.2 Precomputation of Collocations . . . 84
4.3.2.3 Putting it all Together: The Root QUERY algorithm . . . 85
4.4 Source-Driven Phrase Extraction . . . 85
4.5 Results . . . 89
4.5.1 Timing Results . . . 90
4.5.2 Translation Quality Results . . . 92
4.6 Conclusion . . . 94

5 Tera-Scale Translation Models 95
5.1 Rapid Prototyping via Pattern Matching . . . 96
5.2 Experiments . . . 98
5.2.1 Relaxing Grammar Length Restrictions . . . 98
5.2.2 Interlude: Hierarchical Phrase-Based Translation and Lexical Reordering 99
5.2.3 Complementary Scaling Axes . . . 101
5.2.3.1 Out of Domain Data . . . 101
5.2.3.2 Discriminatively Trained Word Alignments . . . 102
5.2.3.3 Phrase Extraction Heuristics . . . 102
5.2.3.4 Coherent Translation Units . . . 103
5.2.3.5 Results . . . 104
5.3 A Tera-Scale Translation Model . . . 104
5.4 Conclusions . . . 105

6 Conclusions and Future Work 106
6.1 Future Work . . . 107
6.2 Concurrent Trends and Related Work . . . 108

Bibliography 110

List of Tables

3.1 Baseline system results compared with systems missing one or more features. . . 60

3.2 Effect of different sample sizes on translation speed and translation accuracy. . . 61

4.1 Timing results for different combinations of algorithms (seconds per sentence), not including extraction or decoding time. 91

4.2 Effect of double binary λ parameter on per-sentence query time. 91

4.3 Effect of precomputation on memory use and processing time 92

4.4 Baseline system results compared with systems missing one or more features. . . 93

4.5 Effect of different sampling sizes on per-sentence times for query, extraction, and scoring and translation accuracy. 93

5.1 CPU time needed to generate the baseline model. 97

5.2 Size of representations for the baseline model. 98

5.3 Effect of varying phrase span and phrase length parameters. Baseline values are in bold. 99

5.4 Effect of varying the maximum number of nonterminals and subpatterns. 100

5.5 Sizes of experimental corpora. 102

5.6 Results of scaling modifications and ablation experiments. 104

5.7 Generation times and size of representations for the improved model (cf. Table 5.2). 105

List of Figures

2.1 An example of translationally equivalent sentences. 6

2.2 An alignment of the the sentence in Figure 2.1. 7

2.3 Visualization of IBM Model 4. 9

2.4 Visualization of the finite-state transducer conception of IBM Model 4. 11

2.5 Visualization of the phrase-based model of translation. 12

2.6 Visualization of the finite-state transducer conception of phrase-based translation. 13

2.7 Visualization of CFG and SCFG derivations. 15

2.8 Visualization of a bracketing grammar derivation. 16

2.9 Visualization of inside-outside alignments. 16

2.10 Visualization of hierarchical phrase-based translation. 18

2.11 The noisy channel model of sentence pair generation. 21

2.12 Supervised learning of phrases from word alignments. 29

2.13 Illustration of the MERT line minimization algorithm. 34

2.14 The flow of data, models, and processes commonly involved in the deployment of an SMT system. 36

2.15 Illustration of search in a finite-state decoder 38

2.16 Illustration of SCFG decoding. 40

2.17 Example of partial string matching used for most evaluation methods 42

3.1 Example phrase table. 48

3.2 Prefix tree representation of the phrase table. 49

3.3 Architecture of a simple table-based decoder. 50

3.4 Number of unique source phrases of different lengths and their cumulative effect on estimated phrase table size. 51

3.5 Architecture of a table-based decoder with filtering. 52

3.6 Translation by pattern matching. 53

3.7 Example input sentence and resulting query patterns for phrase-based translation. 55

3.8 Example of a text and its set of suffixes 56

3.9 Suffix array example 57

3.10 Examples of source-driven phrase extraction. 58

3.11 Histogram of source phrase frequencies. 59

4.1 Example input sentence and resulting query pattern for hierarchical phrase-based translation. 65

4.2 Matches in a suffix array fragment for the discontiguous query pattern *him X it*. . 66

4.3 Illustration of baseline pattern matching algorithm for query pattern *him X it*. . . 68

4.4 Cumulative time required for collocation computations. 70

4.5 Size of input sets ($|M_1^k| + |M_{k+1}|$) compared with time required to compute collocation from the two sets using the baseline algorithm (double logscale). 70

4.6 Portion of the prefix tree for tree for our example input (Figure 4.1). 72

4.7 A fragment of the prefix tree augmented with suffix links. 74

4.8 Illustration of several recursive cases in the prefix tree construction algorithm. . . 77

4.9 Relationship between a matching $m_{a\alpha b}$ of pattern $a\alpha b$ and its suffix matching $s(m_{a\alpha b})$. 78

4.10 QUERY examples. 79

4.11 Examples showing the division of $M_{\alpha b}$ into regions by $m_{a\alpha} \in M_{a\alpha}$. 80

4.12 xample showing the relationships of $m_{a\alpha} \in M_{a\alpha}$ to an element $m_{\alpha b}$. 82

4.13 Illustration of the precomputation algorithm. 84

4.14 Examples of hierarchical base phrase extraction. 87

4.15 Examples of extended hierarchical phrase extraction. 88

4.16 Effect of caching on average and per-sentence lookup time and memory use. . . . 92

5.1 Experimental workflow for conventional architecture vs. translation by pattern matching. 96

1 Introduction

Computer Science is no more about computers than astronomy is about telescopes.

–Edsger Dijkstra

The purpose of computing is insight, not numbers.

–Richard Hamming

Machine translation is an old problem. Like many other computational problems it has been largely reinvented in the last fifteen years through the incorporation of machine learning methods. The machine learning approach to machine translation—called *statistical machine translation*—requires considerable data and careful modeling. With available data increasing in size and the best models increasing in complexity, efficient algorithms are central to continued progress. This dissertation presents an algorithmic approach to statistical machine translation that enables scaling to large data and complex models, and fosters the rapid experimentation needed to exploit them.

Most statistical machine translation systems implement a *direct representation* of the model. This entails offline computation of the complete model and storage of this model in an efficient data structure in main memory at runtime. In practice, these requirements restrict the size of training data and the complexity of the model, since both factors can affect model size.

Callison-Burch et al. (2005) and Zhang and Vogel (2005) proposed an alternative that we call *translation by pattern matching*, which we bring to fruition in this dissertation. This approach relies on an indirect representation of the model. We store the training data itself in memory as its proxy. Model parameters are then computed as needed at runtime. This representation is much more compact than the full model and requires substantially less offline computation. It depends crucially on the ability to quickly search for relevant sections of the training data and compute parameters as needed. To perform this search, we need algorithms for fast pattern matching on text, drawn primarily from the literature in bioinformatics and information retrieval. Although Callison-Burch et al. (2005) and Zhang and Vogel (2005) resolve several problems in the application of translation by pattern matching to a standard phrase-based model, they leave behind several open problems. The first is whether their approach can equal the performance of direct representation, a question that we answer affirmatively.

The next open problem we address is much harder. Callison-Burch et al. (2005) and Zhang and Vogel (2005) applied their approach only to phrase-based translation models based on the translation of contiguous strings. However, a variety of translation models are now based on the translation of *discontiguous* phrases. This new wrinkle poses a difficult algorithmic challenge. Exact pattern matching algorithms used for contiguous phrases don't work and the best approximate pattern matching algorithms are much too slow, taking several minutes per sentence. A major contribution of this dissertation is the development of new approximate pattern matching algorithms for translation models based on discontiguous phrases. Our algorithms reduce the empirical computation time by two orders of magnitude, making translation by pattern matching feasible for these models.

With these tools in hand, we address the problem of scaling our translation model. We then explore several scaling axes—some obvious and some novel—and show that we can significantly improve on a state-of-the-art hierarchical phrase-based translation system. Our experiments highlight interesting characteristics of the model and interactions of various components. Finally, we conclusively demonstrate that reproducing our results with a direct representation would be extremely impractical.

1.1 Outline of the Dissertation

To set the stage for our research, Chapter 2 surveys the current state of the art in statistical machine translation. This is the most extensive contemporary survey of the field. It serves as a foundation on which the remaining chapters build.

Chapter 3 introduces translation by pattern matching. We first review previous work that serves as inspiration for our approach. We then identify some limitations of this work and describe how we overcame these, transforming a clever idea into a practical solution. We establish for the first time that translation by pattern matching is a viable replacement for direct representation in a state-of-the-art phrase-based translation model.

In Chapter 4 we apply translation by pattern matching to models based on the translation of discontiguous strings. This requires the invention of new pattern matching algorithms. We describe our algorithms in detail and analyze them theoretically and empirically. Our algorithms allow us to apply translation by pattern matching to hierarchical phrase-based translation, a stronger baseline than the model used in 3. It matches the state of the art achieved by direct representation.

In Chapter 5 we apply translation by pattern matching to the problem of scaling translation models. We describe a number of axes along which we can scale the models and investigate each of these empirically. We then show that our scaling methods improve translation quality over our strong baseline. Our experiments illuminate some interesting qualities of these models and their interaction with aligned bilingual texts. Finally, we demonstrate that our results would be extremely difficult to replicate using a direct representation.

Chapter 6 contains conclusions and future work. We also relate our work to other developments in the field, and describe future research plans.

1.2 Research Contributions

The dissertation contains several important research contributions.

- We describe a complete algorithmic framework for machine translation by pattern matching that enables scaling to large data, richer modeling, and rapid prototyping.
- We identify unanswered questions in the prior work that provides the foundation of our framework, and describe how we answer them to give state-of-the-art performance in standard phrase-based translation models.
- We introduce a novel approximate pattern matching algorithm that extends our framework to a variety of important models based on discontiguous phrases. Our algorithm solves an empirically difficult algorithmic problem. It is two orders of magnitude faster than the previous state of the art, and enables a number of previously difficult or impossible applications.
- We show how our framework can be applied to scale statistical translation models along several axes, leading to improvement in translation accuracy over a state-of-the-art baseline.

- We show that our method can scale to extremely large models, at least two orders of magnitude larger than most contemporary models.
- We include the most comprehensive survey of the field of statistical machine translation to date.

2 A Survey of Statistical Machine Translation

> *It is very tempting to say that a book written in Chinese is simply a book written in English which was coded into the "Chinese code." If we have useful methods for solving almost any cryptographic problem, may it not be that with proper interpretation we already have useful methods for translation?*
>
> –Warren Weaver

We begin with a tutorial overview and survey of statistical machine translation.[1] It is the most comprehensive contemporary survey of this rapidly growing field. Additionally, it introduces many of the concepts and definitions that are used throughout the remainder of the dissertation. The organization is horizontal—we show that the creation of a statistical translation system requires making a number of important decisions, but that many of these decisions are in fact orthogonal. Two systems that are radically different in one aspect may nonetheless share a common basis in several other aspects. We believe that this type of organization is beneficial to continued progress of the field since it encourages cross-pollination of ideas.

The goals of this chapter are to characterize the core ideas of SMT and provide a taxonomy of various approaches. We have tried to make the survey as self-contained as possible. However, SMT draws from many fundamental research areas in computer science, so some knowledge of automata theory, formal languages, search, and data structures will be beneficial. Familiarity with statistical theory and mathematical optimization techniques used in machine learning will also be helpful, but we will focus on the main ideas and intuitions behind the mathematics rather than full mathematical rigor, which is in any case beyond the scope of this work. We will touch briefly on a few linguistic concepts, but for our purposes SMT can be understood in pure computer science terms.

2.1 Previous Work

Knight (1997, 1999b) has written excellent tutorial introductions, but they do not cover many later developments in the field. Knight and Marcu (2005) briefly survey the recent landscape, while Ayan (2005, chapter 2) provides a longer treatment, focusing primarily on word alignment (§2.5.2). At the time of this writing, other materials in preparation include Koehn (2008) and a chapter in a planned future edition of Jurafsky and Martin (2000). For greater coverage of related fundamental research, refer to textbooks on natural language processing (NLP; Jurafsky and Martin, 2000; Manning and Schütze, 1999), artificial intelligence (Russell and Norvig, 2003) machine learning (Mitchell, 1997), or formal language theory (Hopcroft and Ullman, 1979; Sipser, 2005).

[1] This chapter has been accepted to appear in *ACM Computing Surveys* (Lopez, 2008).

2.2 Background and Context

Machine translation (MT) is the automatic translation from one natural language into another using computers. Interest in MT is nearly as old as the electronic computer—popular accounts trace its modern origins to a letter written by Warren Weaver in 1949, only a few years after ENIAC came online.[2] It has since remained a key application in the field of natural language processing (NLP). A good historical overview is given by Hutchins (2007), and a comprehensive general survey is given by Dorr et al. (1999).

Statistical machine translation (SMT) is an approach to MT that is characterized by the use of machine learning methods. In less than two decades, SMT has come to dominate academic MT research, and has gained a share of the commercial MT market. Progress is rapid and the state-of-the-art is a moving target. However, as the field has matured, some common themes have emerged.

SMT treats translation as a machine learning problem. This means that we apply a learning algorithm to a large body of previously translated text, known variously as a *parallel corpus*, *parallel text*, *bitext*, or *multitext*. The learner is then able translate previously unseen sentences. With an SMT toolkit and enough enough parallel text, we can build an MT system for an new language pair within a very short period of time— perhaps as little as a day (Al-Onaizan et al., 1999; Oard and Och, 2003; Oard et al., 2003). For example, Oard and Och (2003) report constructing a Cebuano-to-English SMT system in a matter of weeks. Workshops have shown that translation systems can be built for a wide variety of language pairs within similar time frames (Callison-Burch et al., 2007; Koehn and Monz, 2005, 2006). The accuracy of these systems depends crucially on the quantity, quality, and domain of the data, but there are many tasks for which even poor translation is useful (Church and Hovy, 1993).

Interest in SMT can be attributed to the convergence of several factors.

1. The growth of the Internet has strongly affected two constituencies of translation consumers. The first of these is interested in the *dissemination* of information in multiple languages. Examples are multilingual governments and news agencies and companies operating in the global marketplace. The Internet enables them to easily publish information in multiple languages. Due to this widespread dissemination, SMT researchers now have access to Biblical texts (Resnik et al., 1997), bilingual government text (Koehn, 2005), bilingual news text, and other data mined from the Internet (Resnik and Smith, 2003). These data are the fundamental resource in SMT research. Because they are the product of day-to-day human activities, they are constantly growing. Multilingual governments interested in dissemination, such as the European Union, have increased MT research funding to further their domestic policy interests.

2. The other consumers of translation are those interested in the *assimilation* of information not in their native language. These include intelligence agencies, researchers, and casual Internet users. The Internet has made such information much more readily accessible, and increasing demand from these users helps drive popular interest in MT. The United States government is interested in assimilation, and has increased MT research funding to further its international policy interests.

3. Fast, cheap computing hardware has enabled applications that depend on large data and billions of statistics. Advances in processor speed, random access memory size, secondary storage, and grid computing have all helped to enable SMT.

4. The development of automatic translation metrics—although controversial—has enabled rapid iterative development of MT systems and fostered competition between research groups.

[2] This letter is reproduced as Weaver (1955).

However	,	the	sky	remained	clear	under	the	strong	north	wind	.
虽然	北	风	呼啸	,	但	天空	依然	十分	清澈	。	
Although	*north*	*wind*	*howls*	*,*	*but*	*sky*	*still*	*extremely*	*limpid*	*.*	

Figure 2.1: An example of translationally equivalent sentences. We give an English gloss for each Chinese word.

Objective measurable goals have naturally led to objective measurable progress. The National Institute of Standards has used these metrics since 2002 in a yearly competition at its MT Evaluation conference.[3] Academic workshops coordinate similar evaluations (Callison-Burch et al., 2007; Koehn and Monz, 2005, 2006).

5. Several projects have focused on the development of freely available SMT toolkits (Al-Onaizan et al., 1999; Burbank et al., 2005; Germann et al., 2001; Koehn, 2004a; Koehn et al., 2007; Och and Ney, 2003; Olteanu et al., 2006). Many are open-source. These implementations help lower the barrier for entry into SMT research.

2.2.1 Formal Description

Formally, our task is to take a sequence of tokens in the *source language* with vocabulary V_F, and transform it into a sequence of tokens in the *target language* with vocabulary V_E.[4] We will assume that tokens are words and sequences are sentences. Agglutinative languages such as German and Inuktitut, or languages with no clearly marked word boundaries, such as Chinese, may require special preprocessing. The most important consideration is that all data are preprocessed consistently, since statistical systems are sensitive to discrepancies. There is often no special treatment of morphological variants—for instance, the English words *translate* and *translation* are treated as unrelated, indivisible tokens. Therefore, it is possible for the size of the vocabularies V_E and V_F to reach into the tens or hundreds of thousands, or even millions in the case of morphologically complex languages such as Arabic.

We denote a sequence of J source words as $f_1 f_2 ... f_J$ or $f_1^J \in {V_F}^J$, and a sequence of I target words as $e_1 e_2 ... e_I$ or $e_1^I \in {V_E}^I$. The goal of a translation system, when presented with an input sequence f_1^J, is to find a sequence e_1^I that is *translationally equivalent*.

An example of translationally equivalent sequences is shown in Figure 2.1. An exercise familiar to those who have learned a foreign language is to draw a line between the words in the sentence that are translations of each other. For instance, we can see that Chinese word 北 is translated as the English word *north*, and we could draw a line between them. We say that such words are *aligned*. An example word alignment is shown in Figure 2.2. This illustrates that translational equivalence can be decomposed into a number of smaller equivalence problems at the word level.

Word translation is often ambiguous. For instance, we might reasonably translate 北 as *northern* without loss of meaning. However, it is not uncommon for the different possible translations of a word to have very different meanings. Often, the correct choice will depend on context. Therefore, our system will need some mechanism to choose between several possible options for

[3] These evaluations are described at `http://www.nist.gov/speech/tests/mt`.

[4] We follow the widely used notation of Brown et al. (1993), who use E for English and F for French (or foreign).

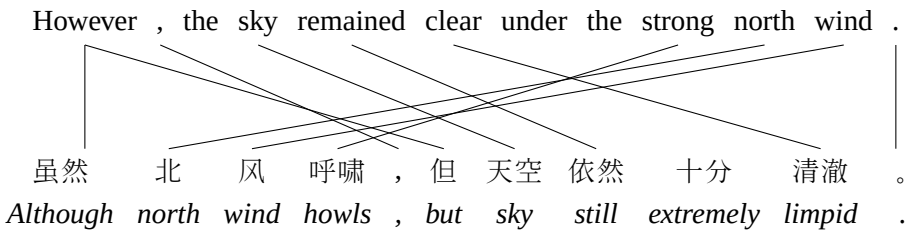


Figure 2.2: An alignment of the the sentence in Figure 2.1.

each translation decision.

In addition to word translation, the other main problem that can be seen from the figure is that words with equivalent meanings do not appear in the same order in both sentences. Therefore, our system will need some mechanism to correctly *reorder* the words. Reordering is typically dependent on the syntactic structure of the target language. For instance, in English sentences, the typical sentence order is *subject-verb-object* (SVO). In Japanese, it is *subject-object-verb* (SOV). As with word translation, reordering decisions often entail resolving some kind of ambiguity.

Translation can therefore be thought of as making a sequence of word translation and reordering decisions. We rewrite the source sentence one decision at a time, until we have replaced it completely with a target sentence. At each decision point, a large number of possible rules may apply. We will need some mechanism to disambiguate these rules. Furthermore, both rules and methods of disambiguation must be learned from our parallel data.

Following these core ideas, there are four problems that we must solve in order to build a functioning SMT system.

1. First, we must describe the series of steps that transform a source sentence into a target sentence. We can think of this as creating a story about how a human translator might perform this task. This story is called a *translational equivalence model*, or more simply a *model*. All of the translational equivalence models that we will consider derive from concepts from automata and language theory. We describe them in §2.3.

2. Next, we want to enable our model to make good choices when faced with a decision to resolve some ambiguity. We need to develop a *parameterization* of the model that will enable us to assign a score to every possible source and target sentence pair that our model might consider. We describe parameterization in §2.4. Taken together, translational equivalence modeling and parameterization are often combined under the rubric of *modeling*.[5]

3. The parameterization defines a set of statistics called *parameters* used to score the model, but we need to associate values to these parameters. This is called *parameter estimation*, and it is based on machine learning methods. We describe parameter estimation in §2.5.

4. Finally, when we are presented with input sentence, we must search for the highest-scoring translation according to our model. This is called *decoding*. We describe decoding in §2.6.

In addition to these four problems, we will discuss the important topic of evaluation in §2.7. The article concludes with notes on current directions in §2.8.

[5] Although these two steps are conflated under this term in the literature, following Brown et al. (1990), for didactic reasons we find it helpful to separate them.

2.3 Modeling Part I: Translational Equivalence

Broadly speaking, a model is simply the set of all rules employed by an MT system to transform a source sentence into a target sentence. In principle these rules may come from anywhere. In most SMT systems they are automatically extracted from a parallel corpus. The extraction process is described in more detail in §2.5.2. In this section, we describe the various types of models.

Most popular models can be described by one of two formalisms: finite-state transducers (FST) or synchronous context-free grammars (SCFG).[6] These formalisms are generalizations of finite-state automata (FSA) and context-free grammar (CFG), respectively. Rather than producing single output strings as in those formalisms, they produce two output strings, and define an alignment between them.[7] Translational equivalence models are important in decoding, where they constrain the search space in which we will attempt to find translations.

The remainder of this section discusses translational equivalence models. FST models are described in §2.3.1, and SCFG models are described in §2.3.2. We briefly touch on other model types in §2.3.3.

2.3.1 Finite-State Transducer Models

Finite-state transducers are extensions of finite-state automata (FSA). Recall that we can define a finite-state automaton (S, L, D) as a set of states S, a set of labels L, and a set of transitions D. Each transition in $D \subseteq \{S \times S \times L\}$ is defined as a pair of states and a label that must be output (or read, depending on the use of the FSA) as we move from the first state to the second. Finite-state methods are widely used in NLP for problems such as automatic speech recognition and part-of-speech tagging.

Finite-state transducers extend FSAs with a second label set. Each transition includes a label from both sets. We can imagine that the transducer operates on an input string and an output string. When a label from the first set is read from the input while traversing the transition, the label for the other is written to the output. A transition labelled with x from set L_1 and y from set L_2 therefore signifies a correspondence between x and y. Additionally, either or both labels may consist of the empty string ε, which indicates that there is no change in that output for that particular transition. We can compose FSTs by making the output of one FST the input to another, giving us a useful modeling tool. For each of the model types that we describe in this section, we will show how it can be implemented with composed FSTs.

We first describe *word-based models* (§2.3.1.1), which introduce many common problems in translation modeling. They are followed by *phrase-based models* (§2.3.1.2).

[6] This distinction may be confusing, since finite state transducers come to us from automata theory and synchronous context-free grammars from formal language theory. Although concepts from each theory have dual representations in the other, it is a historical accident that in the first case, the automata theory concept is generally used, and in the second case, the language theory concept is generally used. We simply follow the prevailing terminology in the literature. However, we note that there is growing interest in *tree transducers*, a class of objects in the automata literature with computational properties similar (but not identical) to those of context-free grammar (Galley et al., 2006; Knight and Graehl, 2005; Marcu et al., 2006).

[7] We can generalize these formalisms even further to handle an arbitrary number of dimensions (Melamed, 2003). This is useful for *multi-source translation*, wherein we have a document already translated in multiple languages, and we would like to translate into another language Och and Ney (2001).

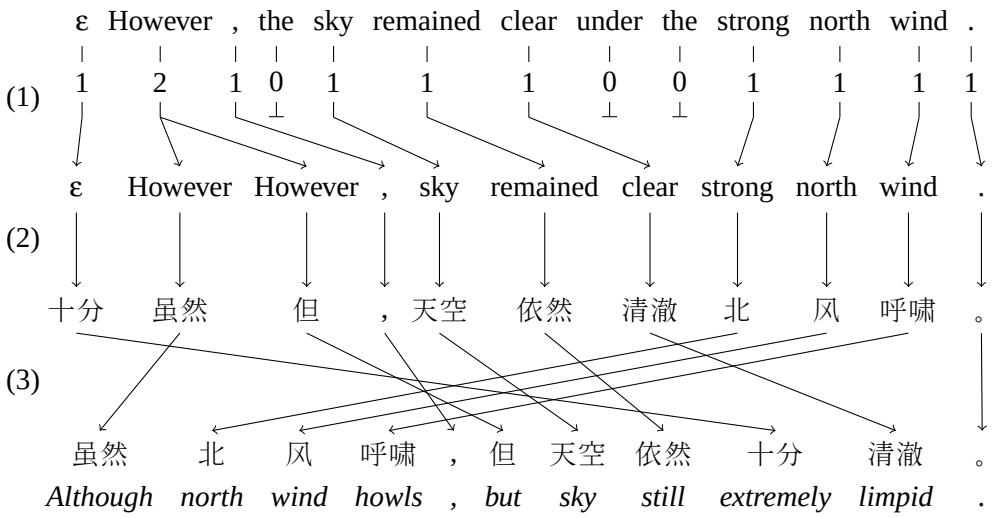


Figure 2.3: Visualization of IBM Model 4. This model of translation takes three steps. (1) Each English word (and the null word) selects a fertility—the number of Chinese words to which it corresponds. (2) Each English word produces a number of Chinese words corresponding to its fertility. Each Chinese word is generated independently. (3) The Chinese words are reordered.

2.3.1.1 Word-Based Models

SMT continues to be influenced by the groundbreaking IBM approach (Berger et al., 1994; Brown et al., 1990, 1993). The IBM Models are *word-based models* and represent the first generation of SMT models. They illustrate many common modeling concepts. We focus on a representative example, IBM Model 4.

For reasons that we will explain in §2.4.1, IBM Model 4 is described as a target-to-source model—that is, a model that produces the source sentence f_1^J from the target sentence e_1^I. We follow this convention in our description.

The model entails three steps (Figure 2.3). Each step corresponds to a single transducer in a composed set (Knight and Al-Onaizan, 1998). The transducers are illustrated in Figure 2.4.

1. Each target word chooses the number of source words that it will generate. We call this number ϕ_i the *fertility* of e_i. One way of thinking about fertility is that when the transducer encounters the word e_i in the input, it outputs ϕ_i copies of e_i in the output. The length J of the source sentence is determined at this step since $J = \sum_{i=0}^{I} \phi_i$. This enables us to define a translational equivalence between source and target sequences of different lengths.

2. Each copy of each source word produces a single target word. This represents the translation of individual words.

3. The translated words are permuted into their final order.

These steps are also applied to a special empty token ε, called the *null word*—or more simply *null*—and denoted e_0. *Null translation* accounts for source words that are dropped in translation, as is often the case with function words.

Note that the IBM Model 4 alignment is asymmetric. Each source word can align to exactly one target word or the null word. However, a target word can link to an arbitrary number of

source words, as defined by its fertility. This will be important when we discuss word alignment (§2.5.2.2).

The reordering step exposes a key difficulty of finite-state transducers for translation. There is no efficient way to represent reordering in a standard finite-state transducer. They are designed to represent relationships between strings with a monotonic alignment —in other words, if an input label at position i is aligned to an output label at position j, then an input label at position $i' > i$ will be aligned to an output label at position $j' > j$. This is fine for problems such as automatic speech recognition, optical character recognition, and part-of-speech tagging, where monotonicity is the natural relationship between two sequences. However, as we have seen, words are typically reordered in real translation.

One solution to this discrepancy is to simply ignore the reordering problem and require monotonic alignments (Banchs et al., 2005; Tillmann et al., 1997; Zens and Ney, 2004). This enables very fast decoding, but generally leads to less accurate translation, particularly for languages with naturally different word orders.

At the other end of the spectrum is full permutation of the J source words. An FST representation of a permuted sequence contains $O(J!)$ paths and $O(2^J)$ states (Och and Ney, 2003). However, search for the best permutation is NP-complete, as Knight (1999a) shows by reduction to the Traveling Salesman Problem.

Most models take a middle ground, using a mostly monotonic approach but allowing any of the k leftmost uncovered words to be translated. The setting $k = 4$ is sometimes called the *IBM constraint*, following Berger et al. (1996a). This method enables local reordering, which accounts for much language variation. However, it still prevents *long-distance reordering*, such as the movement of "north" (北) in our example. Accounting for long-distance reordering without resorting to arbitrary permutation requires additional modeling. For instance, Tillman and Ney (2003) describe targeted long-distance reordering for verb movement in German to English translation, using language-specific heuristics.

As we can see, a dominant theme of translation modeling is the constant tension between expressive modeling of reordering phenomena, and model complexity.

2.3.1.2 Phrase-Based Models

In real translation, it is common for contiguous sequences of words to translate as a unit. For instance, the expression "北 风" is usually translated as "north wind". However, in word-based translation models, a substitution and reordering decision is made separately for each individual word. For instance, our model would have to translate 北 as "north" and 风 as "wind", and then order them monotonically, with no intervening words. Many decisions invite many opportunities for error. This often produces "word salad"— a translation in which many words are correct, but their order is a confusing jumble.

Phrase-based translation addresses this problem (Koehn et al., 2003; Marcu and Wong, 2002; Och and Ney, 2004; Och et al., 1999). In phrase-based models the unit of translation may be any contiguous sequence of words, called a *phrase*.[8] Null translation and fertility are gone. Each source phrase is nonempty and translates to exactly one nonempty target phrase. However, we do not require the phrases to have equal length, so our model can still produce translations of different length. Now we can characterize the substitution of "north wind" with "北 风" as an atomic operation. If in our parallel corpus we have only ever seen "north" and "wind" separately, we can still translate them using one-word phrases.

[8] In this usage, the term "phrase" has no specific linguistic sense.

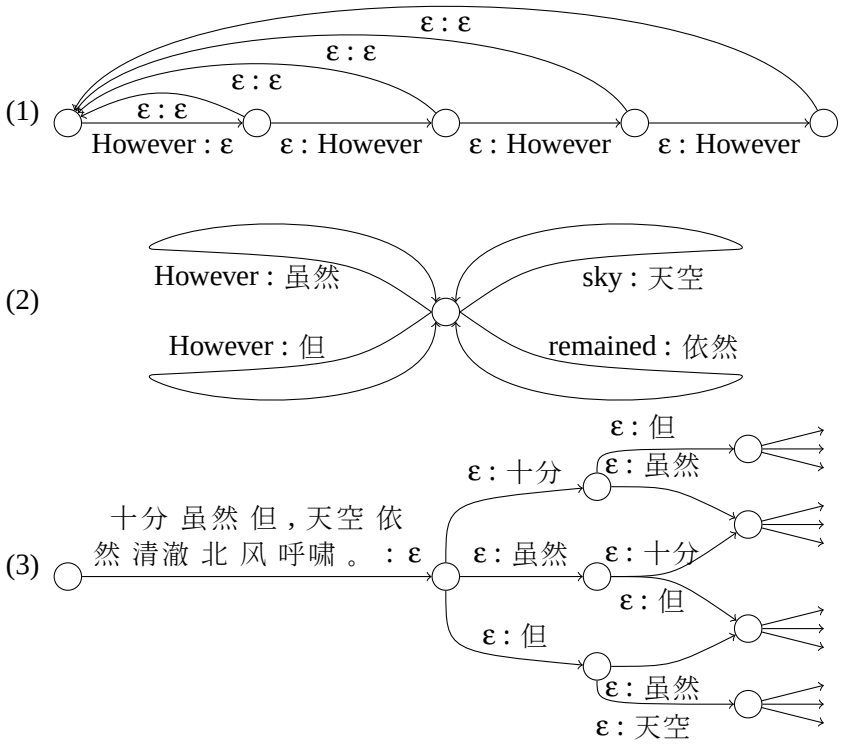


Figure 2.4: Visualization of the finite-state transducer conception of IBM Model 4. We show only a portion of each transducer. Transducer (1) copies the input word to its output a number of times according to its fertility; (2) corresponds to word-to-word translation, and (3) corresponds to reordering. Transducer (3) is the most complex, because it must represent all possible reorderings. We can compose these transducers to represent the process shown in Figure 2.3.

The translation process takes three steps (Figure 2.5).

1. The sentence is first split into phrases.
2. Each phrase is translated.
3. The translated phrases are permuted into their final order. The permutation problem and its solutions are identical to those in word-based translation.

A cascade of transducers implementing this is shown in Figure 2.6. A more detailed implementation is described by Kumar et al. (2006).

The most general form of phrase-based model makes no requirement of individual word alignments (Marcu and Wong, 2002). Explicit internal word alignments are sometimes assumed for the purposes of parameterization (Koehn et al., 2003; Kumar et al., 2006).

A variant of phrase-based models is the *alignment template model* (Och and Ney, 2004; Och et al., 1999). In this model explicit phrase-to-phrase translations are not used. Instead, each phrase is first associated with an alignment template, which is a reordering of the words in the phrase based on word categories rather than specific word identities. The words in the phrase are then translated using word-to-word translation, and the alignment templates are reordered as in phrase-based models.

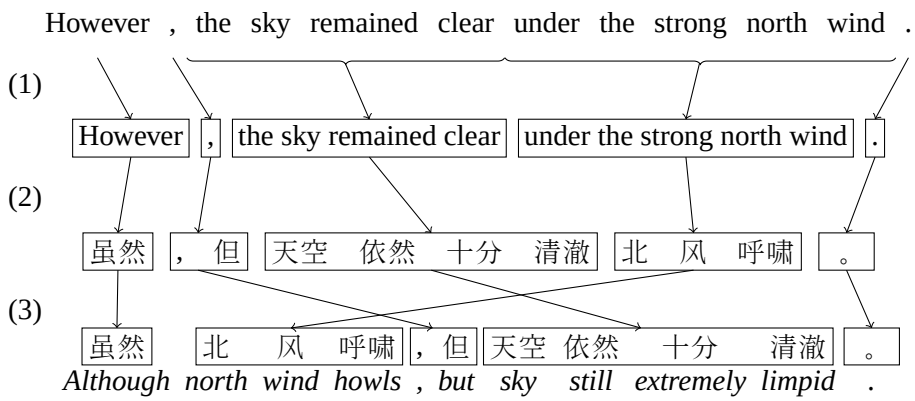


Figure 2.5: Visualization of the phrase-based model of translation. The model involves three steps. (1) The English sentence is segmented into "phrases"—arbitrary contiguous sequences of words. (2) Each phrase is translated. (3) The translated phrases are reordered.

Simard et al. (2005) present another variant of phrase-based translation in which phrases can be discontiguous, and gaps must be filled in with other phrases.

Phrase-based translation is implemented in the Pharaoh toolkit (Koehn, 2004a) and its open-source successor Moses (Koehn et al., 2007).[9] They are widely used in the SMT research community.

Phrase-based models produce better translations than word-based models and they are widely used. They successfully model many local reorderings, and individual passages are often fluent. However, they cannot easily model long-distance reordering without invoking the expense of arbitrary permutation. This sometimes leads to *phrase salad*. It is sometimes argued that linguistic notions of syntax are needed to adequately model these phenomena (Burbank et al., 2005). However, incorporation of syntax into phrase-based models has had mixed results, with some failures (Koehn et al., 2003; Och et al., 2004b) and some successes (Collins et al., 2005; Wang et al., 2007). In general, phrase-based models seem to be a poor fit for syntax, which usually depends on hierarchical models of string generation.

Formal language theory tells us that the set of *regular languages* that can be generated using finite-state automata is a strict subset of the set of *context-free languages* that can be generated using push-down automata. To gain modeling power, we will now look at synchronous formalisms from this theoretically more powerful class.

2.3.2 Synchronous Context-Free Grammar Models

Compared with the regular languages generated by FSAs, *context-free grammars* (CFG) confer a couple of potential benefits that relate to our modeling problem. First, they are closely tied to some linguistic representations of syntax. Second, in their synchronous form, they can easily represent long-distance reordering without the exponential complexity of permutation. However, added modeling power comes with added modeling challenges, and meeting these challenges is currently an area of much active research. Indeed, there is some skepticism over the superiority of synchronous grammar models over finite-state methods. For the remainder of this section, we describe translation models based on *synchronous context-free grammars* (SCFG), and describe

[9] The Moses toolkit is available at `http://www.statmt.org/moses`.

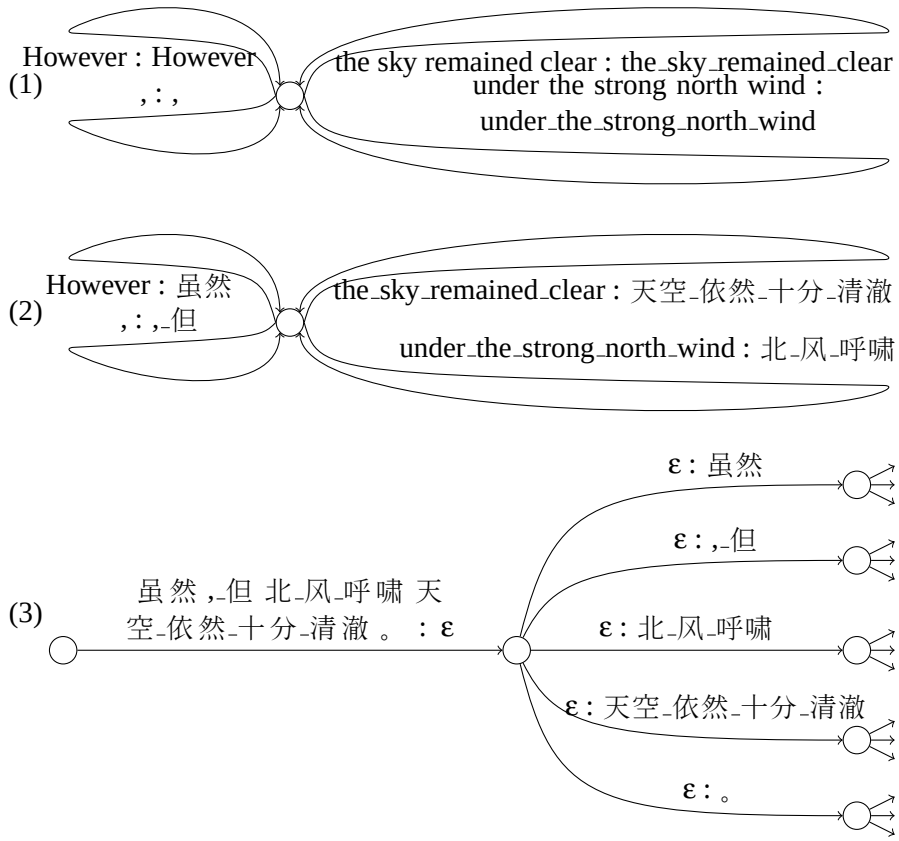


Figure 2.6: Visualization of the finite-state transducer conception of phrase-based translation. We show only a portion of each transducer. Transducer (1) segments the target sentence into phrases; (2) performs word-to-word translation; (3) reorders the phrases in the same way that the reordering transducer of Figure 2.4 reorders words.

some of their modeling challenges.

SCFGs are known in different guises as *syntax-directed translation* (Lewis and Stearns, 1968), *inversion transduction grammar* (Wu, 1995a), *head transducers* (Alshawi et al., 2000), and a number of other names. A formalism that generalizes these is multitext grammar (Melamed, 2003). Chiang (2006) provides a good overview of SCFG and several variants. In the following discussion, we will use the notation of SCFG, which is relatively easy to understand.

SCFG is a generalization of CFG to the case of two output strings. Recall that a CFG (N,T,D) consists of a set of non-terminal symbols N, terminal symbols T, and productions $D = \{N \rightarrow \{N^* \times T^*\}\}$. Each production describes the in-place replacement of the productions *right-hand side* nonterminal with the *left-hand side* sequence of terminals and nonterminals. We begin by writing a special *root* non-terminal symbol to the output. This symbol is rewritten using a rule $d \in D$. Rewriting of non-terminal symbols continues recursively until the output contains only terminal symbols. CFG is popular in natural language parsing, where terminal symbols are words (i.e. $T = E$) and non-terminal symbols represent syntactic categories. Consider the following fragment of a CFG grammar.[10]

[10]The nonterminal categories are borrowed from those used in an widely used syntactic annotation of several thousand sentences from the Wall Street Journal (Marcus et al., 1993). They represent English syntactic categories such as

$$\text{NP} \longrightarrow \text{DT NPB} \quad \text{(C1)}$$
$$\text{NPB} \longrightarrow \text{JJ NPB} \quad \text{(C2)}$$
$$\text{NPB} \longrightarrow \text{NP} \quad \text{(C3)}$$
$$\text{DT} \longrightarrow \text{the} \quad \text{(C4)}$$
$$\text{JJ} \longrightarrow \text{strong} \quad \text{(C5)}$$
$$\text{JJ} \longrightarrow \text{north} \quad \text{(C6)}$$
$$\text{NN} \longrightarrow \text{wind} \quad \text{(C7)}$$

In this grammar, the syntactic category NP can be rewritten as "the strong north wind" via a series of productions, as shown in Figure 2.7.

In SCFG, the grammar specifies *two* output strings for each production. It defines a correspondence between strings via the use of *co-indexed* nonterminals. Consider a fragment of SCFG grammar.

$$\text{NP} \longrightarrow \text{DT}_{\boxed{1}}\text{NPB}_{\boxed{2}} \;/\; \text{DT}_{\boxed{1}}\text{NPB}_{\boxed{2}} \quad \text{(S1)}$$
$$\text{NPB} \longrightarrow \text{JJ}_{\boxed{1}}\text{NN}_{\boxed{2}} \;/\; \text{JJ}_{\boxed{1}}\text{NN}_{\boxed{2}} \quad \text{(S2)}$$
$$\text{NPB} \longrightarrow \text{NPB}_{\boxed{1}}\text{JJ}_{\boxed{2}} \;/\; \text{JJ}_{\boxed{2}}\text{NPB}_{\boxed{1}} \quad \text{(S3)}$$
$$\text{DT} \longrightarrow \text{the} \;/\; \varepsilon \quad \text{(S4)}$$
$$\text{JJ} \longrightarrow \text{strong} \;/\; 呼啸 \quad \text{(S5)}$$
$$\text{JJ} \longrightarrow \text{north} \;/\; 北 \quad \text{(S6)}$$
$$\text{NN} \longrightarrow \text{wind} \;/\; 风 \quad \text{(S7)}$$

The two outputs are separated by a slash (/). The boxed numbers are co-indexes. When two nonterminals share the same co-index, they are aligned. We can think of SCFG as generating isomorphic trees, in which the non-terminal nodes of each tree are aligned. One tree can be transformed into another by rotating its non-terminal nodes, as if it were an Alexander Calder mobile (Figure 2.7). Note that if we ignore the source string dimension, the rules in this SCFG correspond to the rules that appear in our CFG example. Also note that we can infer an alignment between source and target words based on the alignment of their parent nonterminals. In this way SCFG defines a mapping between both strings and trees, and has a number of uses depending on the relationship that we are interested in (Melamed, 2004a; Wu, 1995a).

Normal forms and complexity analysis for various flavors of SCFG are presented in Aho and Ullman (1969) and Melamed (2003). The number of possible reorderings is quite large. Church and Patil (1982) shows that the number of binary-branching trees that can generate a string is related to a combinatorial function, the *Catalan number*. Zens and Ney (2003) shows that the number of reorderings in a binary SCFG are related to another combinatorial function, the *Shröder number*. However, due to recursive sharing of subtrees among many derivations, we can parse SCFG in polynomial time with dynamic programming algorithms (Melamed, 2003). This makes SCFG models of translational equivalence less computationally expensive than full permutation, even though they can model long-distance reordering (Wu, 1996).

Numerous different approaches to SMT can be expressed in the SCFG formalism. In the following sections we will illustrate three applications of SCFG that are representative of their use

noun phrase (NP), adjective (JJ), preposition (IN), determiner (DT), and noun (NN).

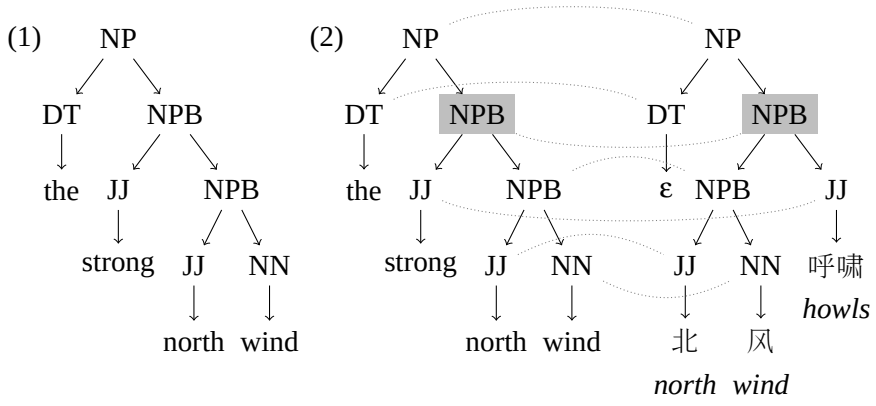


Figure 2.7: Visualization of CFG and SCFG derivations. Derivation happens in exactly the same way in CFG (1) and SCFG (2). Each nonterminal symbol is replaced by the contents of the right-hand-side of a rule whose left-hand-side matches the symbol. The difference in SCFG is that we specify two outputs rather than one. Each of the non-terminal nodes in one output is linked to exactly one node in the other; the only difference between the outputs is the order in which these nodes appear. Therefore, the trees are isomorphic. Although terminal nodes are not linked, we can infer a word alignment between words that are generated by the same non-terminal node. In this illustration, the only reordering production is highlighted. Note that if we ignore the Chinese dimension of the output, the SCFG derivation in the English dimension is exactly the same as in (1).

in SMT. We will consider *bracketing grammars* used to constrain reordering (§2.3.2.1), *syntax-based translation* that exploits linguistic syntax (§2.3.2.2), and *hierarchical phrase-based translation* that combines the insights of phrase-based models with formal syntactic models (§2.3.2.3).

2.3.2.1 Bracketing Grammars

One reason to use SCFG is efficient expression of reordering. In FST models, long-distance reordering is difficult to model. The most permissive approach—arbitrary permutation—is exponential in sentence length. In contrast, SCFG can represent long-distance reordering while remaining polynomial in sentence length. This motivates the use of bracketing grammars. They represent all possible reorderings consistent with a binary bracketing of the input string (Wu, 1996). Bracketing grammars use a single undifferentiated nonterminal symbol and three rules.

$$\mathrm{X} \longrightarrow \mathrm{X}_{\boxed{1}}\mathrm{X}_{\boxed{2}} \;/\; \mathrm{X}_{\boxed{1}}\mathrm{X}_{\boxed{2}} \tag{B1}$$

$$\mathrm{X} \longrightarrow \mathrm{X}_{\boxed{1}}\mathrm{X}_{\boxed{2}} \;/\; \mathrm{X}_{\boxed{2}}\mathrm{X}_{\boxed{1}} \tag{B2}$$

$$\mathrm{X} \longrightarrow e \;/\; f \tag{B3}$$

In Rule B1 we define symbols $e \in V_E \cup \varepsilon$ and $f \in V_F \cup \varepsilon$. Instances of this rule model word-to-word alignments.

The polynomial runtime of bracketing grammar comes at a cost. In contrast with permutation, the bracketing grammar cannot represent certain reorderings. A canonical example is the so-called inside-outside alignment (Figure 2.9). It was originally argued that this reordering does

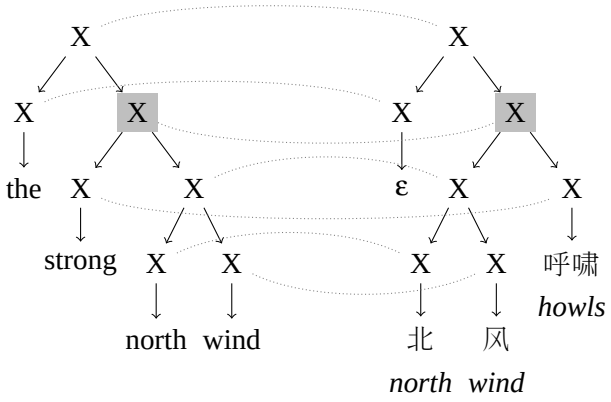


Figure 2.8: Visualization of a bracketing grammar derivation. Rules that involve reordering are highlighted, and the order of the target language sentence is illustrated beneath the syntax tree.

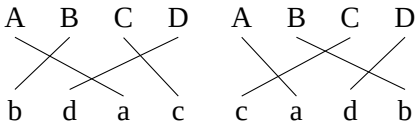


Figure 2.9: Visualization of inside-outside alignments that are not possible using bracketing transduction grammar. Due to the interleaving words in these configurations, we cannot construct a binary-branching SCFG that is isomorphic for these strings, although a SCFG with four nonterminals can produce them. These are the smallest impossible configurations in this grammar; as the number of words in each string increases, so does the number of impossible configurations.

not occur in real translation (Wu, 1995b). However, Wellington et al. (2006b) provided counterexamples from real data. On the other hand, Zens and Ney (2003) show empirically that bracketing grammars can model more of the reorderings observed in real data than a word-based model using the IBM constraint.

One way to increase the reordering power of an SCFG is to increase the number of coindexed nonterminal symbols in each rule. In general, any alignment can be expressed in a grammar with enough coindexed nonterminals (Aho and Ullman, 1969). The tradeoff is increased complexity. In particular, upper bound complexity is exponential in the maximum number of coindexed nonterminals. As this number increases, the reordering power and the complexity of the grammar approach those of permutation.

A *lexicalized* bracketing grammar, in which nonterminal symbols are annotated with words, is described in Zhang and Gildea (2005). A related formalism is the head transduction grammar (Alshawi et al., 2000). Xiong et al. (2006) adapted bracketing grammars to phrase translation.

2.3.2.2 Syntax-Based Translation

In addition to long-distance reordering, a perceived benefit of SCFG is that it allows us to easily incorporate knowledge based on natural language syntax. This follows from developments in syntactic modeling for ASR (Chelba and Jelinek, 1998).

Often, we will have meaningful linguistic grammars only for one language.[11] Monolingual syntax resembles our example fragment CFG (Rules C1-C8). To use this monolingual syntax in an SMT model, we construct an SCFG where productions mirror the known syntax. In the other language, we allow arbitrary reorderings of these symbols (Wu and Wong, 1998; Yamada and Knight, 2001). The objective of such a SCFG is to keep linguistic phrases intact, a property that is often (but not universally) observed in real translation (Fox, 2002; Wellington et al., 2006b). Collectively, models that use linguistic syntax are called *syntax-based models*. The example derivation from Figure 2.7 is an illustration of this.

The hypothesis investigated by syntax-based translation was that reorderings would respect linguistic syntax in translation. Empirical evidence only partially supports this. Fox (2002) shows that reordering tends to respect the boundaries of syntactic phrases, but also describes some systematic exceptions. Additional discrepancies are described by Wellington et al. (2006b). The rigidity of full isomorphism at the level of SCFG productions harms syntax-based translation. These difficulties have prompted investigation into more powerful formalisms for syntactic translation (Galley et al., 2004, 2006; Knight and Graehl, 2005).

2.3.2.3 Hierarchical Phrase-Based Translation

The SCFG models that we have described share a key weakness of word-based FST-models: they enable only word-to-word translation, which requires a reordering decision for each word. We know that phrase-based models solve this. Ideally, we would like to benefit from the insights behind both hierarchical models and phrase-based models. This is accomplished in hierarchical phrase-based translation (Chiang, 2005, 2007; Chiang et al., 2005).

In this grammar, no linguistic syntax is required. A single undifferentiated non-terminal X is used in the main productions, and a maximum of two nonterminals are permitted in the right-hand size of any rule, just as in bracketing grammar. However, unlike a bracketing grammar, the right-hand side may also contain a number of terminal symbols in both languages. This corresponds to the basic insight of phrase-based translation, in that each rule can represent a mapping between sequences of words. Essentially, the rules represent phrases that may be reordered recursively. Consider the following grammar fragment.

$$X \longrightarrow \text{However , } X_{\boxed{1}} X_{\boxed{2}} \text{ . } / \text{虽然 } X_{\boxed{2}} \text{ , 但 } X_{\boxed{1}} \text{ 。} \quad \text{(H1)}$$

$$X \longrightarrow \text{under the strong north wind } / \text{北 风 呼啸} \quad \text{(H2)}$$

$$X \longrightarrow \text{the sky remained clear } / \text{天空 依然 十分 清澈} \quad \text{(H3)}$$

In this grammar, recursivity is captured in Rule H1. A derivation is illustrated in Figure 2.10.

Chiang (2005, 2007) showed that this model was competitive with a standard phrase-based models, outperforming Pharaoh on standard test sets.

2.3.3 Other Models of Translational Equivalence

FST and SCFG models represent a good cross-section of popular models. However, a number of models do not fit these characterizations.

[11] Usually this is the target language (Wu and Wong, 1998; Yamada and Knight, 2002). This imbalance reflects the fact that many of the translation systems reported in the literature are designed to translate into well-studied languages, such as English, for which we already have high-quality syntactic parsers.

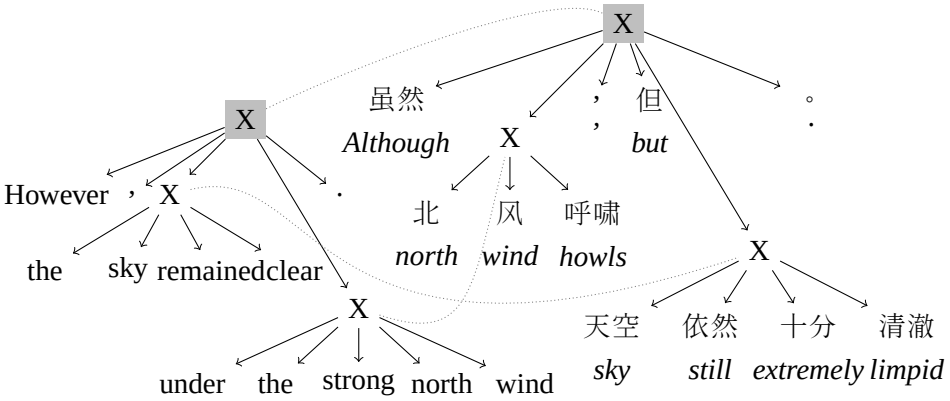


Figure 2.10: Visualization of hierarchical phrase-based translation.

2.3.3.1 More Powerful Formalisms

Moving up the hierarchy of formal languages, there are synchronous models based on language formalisms more powerful than context-free languages. A good example is tree-adjoining grammar (Joshi and Schabes, 1997), which we can generalize to synchronous tree-adjoining grammar (Shieber and Schabes, 1990). Formally, tree-adjoining grammars are part of a large class of formalisms known as linear context-free rewriting systems (Joshi et al., 1991; Vijay-Shanker et al., 1987). These formalisms can parse a restricted subset of context-sensitive languages in polynomial time. Much of the work in this area is currently theoretical. However, *generalized multitext grammar* (Melamed et al., 2004), which is equivalent to a linear context-free rewriting system, is the basis of an SMT system (Burbank et al., 2005).

2.3.3.2 Syntactic Phrase-Based Models

Much active research aims to combine the advantages of hierarchical reordering, syntax, and phrases (Galley et al., 2006; Marcu et al., 2006; Quirk et al., 2005). Most of these models employ syntactic reordering models more complex than those that can be described with SCFG. The formal description for several of these models is based on *tree transducers*, which describe operations on tree fragments rather than strings. SCFG translation can be modeled with tree transducers, although in general they are strictly more powerful than SCFG. Systems described using tree transducers are increasingly common, though many of these are equivalent to SCFG (Galley et al., 2006; Graehl and Knight, 2004; Marcu et al., 2006). For a good introduction to tree transducers, refer to Knight and Graehl (2004).

2.3.3.3 Alternative Linguistic Models

A wide variety of linguistic theories are computationally equivalent to CFG, and these can be used as the basis for translation using SCFG. Head transducers may be seen as a form of synchronous *dependency grammar* (Alshawi et al., 2000). In dependency grammar, the nodes of the rooted tree which describes the sentence structure are also the words of the sentence. It is possible to derive transformations that will convert many dependency grammars to context-free grammars, and vice versa (Collins et al., 1999). Therefore, we can construct SCFGs that correspond to dependency grammar (Melamed, 2003). Dependency grammar translation models

are described by Gildea (2004) and Quirk et al. (2005).

2.4 Modeling Part II: Parameterization

Translational equivalence models allow us to enumerate possible structural relationships between pairs of strings. However, even within the constraints of a strict model, the ambiguity of natural language results in a very large number of possible target sentences for any input source sentence. Our translation system needs a mechanism to choose between them.

This mechanism comes from our second topic in modeling: *parameterization*. We design a function that allows us to assign a real-valued score to any pair of source and target sentences. The general forms of these models are similar to those in other machine learning problems. There are a vast number of approaches; we will only briefly describe the most common ones here. For more detail, the reader is referred to a general text on machine learning, such as Mitchell (1997).

In typical machine learning problems, we are given an input $y \in Y$, and the goal is to find the best output $x \in X$. Note that x and y may be multidimensional. We introduce a function $f : X \times Y \rightarrow \mathbb{R}$ that maps input and output pairs to a real-valued score that is used to rank possible outputs. We introduce the *random variables* $\mathbf{x}$ and $\mathbf{y}$ which range over the sets X and Y, respectively. Let $x \in X$ and $y \in Y$ be specific values drawn from these sets. The model may be probabilistic, meaning that we constrain it in one of two ways. In a *joint model*, denoted $P(\mathbf{x}, \mathbf{y})$, we introduce two constraints.

$$\sum_{(x,y) \in \{X \times Y\}} P(\mathbf{x} = x, \mathbf{y} = y) = 1$$
$$\forall_{(x,y) \in \{X \times Y\}} P(\mathbf{x} = x, \mathbf{y} = y) \in [0, 1]$$

The value $P(\mathbf{x} = x, \mathbf{y} = y)$ is the *joint probability* of the assignments $\mathbf{x} = x$ and $\mathbf{y} = y$ occurring, out of all possible combinations of assignments to these variables. We will often abbreviate this as $P(x, y)$.

In a *conditional model*, denoted $P(\mathbf{x}|\mathbf{y})$, we introduce the following constraints.

$$\forall_{y \in Y} \sum_{x \in X} P(\mathbf{x} = x | \mathbf{y} = y) = 1$$
$$\forall_{(x,y) \in \{X \times Y\}} P(\mathbf{x} = x | \mathbf{y} = y) \in [0, 1]$$

The conditional probability $P(\mathbf{x} = x|\mathbf{y} = y)$, abbreviated $P(x|y)$, is simply the probability of the assignment $\mathbf{x} = x$, given that the assignment $\mathbf{y} = y$ is fixed. In this case, we assume that knowledge of the value assigned to $\mathbf{y}$ will help us determine the assignment to $\mathbf{x}$.

These constraints represent the *distribution* of finite probability mass across all combinations of assignments to $\mathbf{x}$ and $\mathbf{y}$. In many machine learning problems, it is not unusual for the input set Y to be complex. Often, the set X of possible outputs is a small, finite set of *labels* or *classes* and our goal is simply to find the best *classification* of the input. This is not the case in SMT, where our input $\mathbf{f}$ ranges over ${V_F}^*$ and our output $\mathbf{e}$ ranges over ${V_E}^*$. We will usually expand our definition of the output to include the decisions made by our translational equivalence model defining the relationship between $\mathbf{f} = f_1^J$ and $\mathbf{e} = e_1^I$. We denote this structure using the variable $\mathbf{d} = d_1^M \subset D$. Recall that D is a set of transitions in the case of FST models (§2.3.1) and a set of grammar productions in the case of SCFG models (§2.3.2); in other words, D is simply a set of rules. In the SMT problem, we are given an input f_1^J and we are interested in finding multidimensional "class" (e_1^I, d_1^M) drawn from a domain that is exponential in the size of the input. This

problem is known as *structured classification* or *structured prediction* (Taskar, 2004). Note that the set d_1^M of derivations exactly defines e_1^I for a given input f_1^J, and we could simply denote the label using d_1^M; we use (e_1^I, d_1^M) to remind us of our ultimate goal. For notational purposes we also define a predicate $Y(e_1^I, d_1^M)$ that is true when the derivation d_1^M yields e_1^I, and false otherwise.

The mathematical function that we are truly interested in is $P(\mathbf{e}|\mathbf{f})$. This function ignores the derivation of the output. However, these models usually define multiple structures that can relate the same pair (e_1^I, f_1^J). The value of $P(\mathbf{e}|\mathbf{f})$ is therefore obtained by summing the probabilities of all derivations that yield $\mathbf{e}$.

$$P(\mathbf{e}|\mathbf{f}) = \sum_{\mathbf{d}:Y(\mathbf{d},\mathbf{e})} P(\mathbf{e},\mathbf{d}|\mathbf{f}) \tag{2.1}$$

Unfortunately, computation of this sum involves exponential complexity in both FST (Brown et al., 1993) and SCFG models (Melamed, 2004a). Therefore we will use the simpler function $P(\mathbf{e},\mathbf{d}|\mathbf{f})$ for classification. We can view the classification problem as one in which the decoder produces candidate labels according to our translational equivalence model, and the parameterization of the model determines the rank of candidates. Usually these tasks are integrated, but not necessarily (see §2.6.3).

Although the function $P(\mathbf{e},\mathbf{d}|\mathbf{f})$ ranges over discrete sets, these sets are very large or even infinite. This poses both practical and theoretical problems. There is no efficient way to enumerate the function. Furthermore, we will never have enough training data to reliably learn what its values should be. The goal of mathematical modeling, then, is to *parameterize* the function in such a way that we can efficiently and reliably learn it. There are a number of ways to accomplish this. We describe *generative models* in §2.4.1 and *discriminative models* in §2.4.2.

2.4.1 Generative Models

We begin our discussion of generative models by introducing some statistical machinery. Later in the section, we will illustrate this with an example.

One method of manipulating probabilistic models is through use of the chain rule.

$$P(\mathbf{x},\mathbf{y}) = P(\mathbf{x}|\mathbf{y})P(\mathbf{y}) \tag{2.2}$$

Generative models decompose $P(\mathbf{x}|\mathbf{y})$ using Bayes' rule, which we derive using the chain rule and a bit of algebra as shown in Equation 2.3.

$$P(\mathbf{x}|\mathbf{y}) = \frac{P(\mathbf{x},\mathbf{y})}{P(\mathbf{y})} = \frac{P(\mathbf{y}|\mathbf{x})P(\mathbf{x})}{P(\mathbf{y})} \tag{2.3}$$

Applying Bayes' rule to our structured classification problem, we arrive at the following decomposition:

$$P(\mathbf{e},\mathbf{d}|\mathbf{f}) = \frac{P(\mathbf{f},\mathbf{d}|\mathbf{e})P(\mathbf{e})}{P(\mathbf{f})} \tag{2.4}$$

In decoding we can ignore the denominator $P(\mathbf{f})$ because it is constant for any input f_1^J. Therefore, we do not need to consider this model any further and we can focus on $P(\mathbf{f},\mathbf{d}|\mathbf{e})$ and $P(\mathbf{e})$. The decomposition was inspired by its successful use in automatic speech recognition (Brown et al., 1990).

In SMT we call $P(\mathbf{e})$ the *language model* and $P(\mathbf{f},\mathbf{d}|\mathbf{e})$ the *translation model*. Note that while our objective is to discover e_1^I given f_1^J, we actually model the reverse. This originated

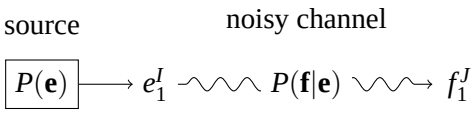


Figure 2.11: The noisy channel model of sentence pair generation. The source model $P(\mathbf{e})$ produces e_1^I, which is then transformed by the channel model $P(\mathbf{f}|\mathbf{e})$ to produce f_1^J. If we are given only f_1^J, we can try to deduce an e_1^I using our knowledge of both models.

with the IBM system, and it is for this reason that IBM Model 4 is described as a translation from e_1^I to f_1^J as we saw in §2.3.1.1. The advantage of this over modeling $P(\mathbf{e},\mathbf{d}|\mathbf{f})$ directly is that we can apply two independent models to the disambiguation of $\mathbf{e}$ (Brown et al., 1990). This is beneficial because our estimates for each model are errorful. By applying them together we hope to counterbalance their errors.

In fact, we can think of the language model $P(\mathbf{e})$ as a stochastic model that generates target language sentences, and the translation model $P(\mathbf{f},\mathbf{d}|\mathbf{e})$ as a second stochastic process that "corrupts" the target language to produce source language sentences. This idea was suggested by Weaver.

> One naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: "This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode." (Weaver, 1955)

Weaver makes analogy to information theoretic work on signal transmission over a physical medium, called the *noisy channel* problem. It is illustrated in Figure 2.11. Following Weaver, the process to recover e_1^I is called *decoding*.

2.4.1.1 Language Models

In language modeling, our goal is to find a tractable representation of the function $P(e_1^I)$. Generative models use probabilistic tools for this. One of these is chain rule, which allows us to write the following.

$$P(e_1^I) = \prod_{i=1}^{I} P(e_i|e_1^{i-1}) \tag{2.5}$$

This equation tells us that the conditional probability of the sentence e_1^I is simply the product of many small probabilities, each of which corresponds to a single word.

Equation 2.5 helps to simplify our problem, but not completely. For instance, the distribution $P(e_I|e_1^{I-1})$ assigned to the last word of the sentence contains nearly as many terms as $P(e_1^I)$ itself. In order to simplify the model even further we introduce the idea of *conditional independence*. When we say that a variable $\mathbf{x}$ is conditionally independent of $\mathbf{y}$, we mean that $P(\mathbf{x}|\mathbf{y}) = P(\mathbf{x})$. In other words, conditional independence means that knowing the value of $\mathbf{y}$ does not affect the probability distribution of $\mathbf{x}$. By making independence assumptions about our data, we can drop enough terms from our functions that they become tractable. The obvious danger, of course, is that if we make too many or wrong independence assumptions, our resulting probability

distributions will be incorrect. This is a constant danger in SMT modeling. In general, nearly any independence assumption will be unrealistic; our hope is that they are approximately close enough to that it will not harm our results. Making conditional independence assumptions is a matter of art and pragmatism.

In language modeling, the simplest assumption we can make is that the probability of word e_i is conditionally independent of all but the $n-1$ preceding words e_{i-n}^{i-1}. We call e_{i-n}^{i} an *n-gram* and the language model based on this independence assumption is an *n-gram language model*. We assume without loss of generality that the first word e_1 is preceded by $n-1$ distinguished start symbols not in V_E.

We will use the notation $P_\delta(\mathbf{x}|\mathbf{y})$ to represent the distribution $P(\mathbf{x}|\mathbf{y})$ that has been rewritten to include only elements of $\mathbf{y}$ on which $\mathbf{x}$ is conditionally dependent. Using this notation, we can rewrite Equation 2.5 as an *n*-gram model.

$$P(e_1^I) = \prod_{j=1}^{I} P_\delta(e_i|e_1^{i-1}) = \prod_{j=1}^{I} P(e_i|e_{i-n}^{i-1}) \tag{2.6}$$

Most language models take this form, inherited from SMT's roots in speech recognition. The discussion of *n*-grams barely scratches the surface of language modeling, which is a full topic in its own right, well beyond the scope of this paper. For a good overview, refer to a text on speech recognition, such as Jelinek (1998).

Language modeling has not received much special attention in the SMT community, which has preferred to focus on the more specialized translation models. However, it continues to be an active area of research, especially in the speech recognition community. Most SMT systems simply borrow models that were popularized for speech. However, there is little doubt that better language modeling leads to better translation (Brants et al., 2007; Eck et al., 2004; Kirchhoff and Yang, 2005; Och, 2005; Zhang et al., 2006b). A few syntax-based models are constructed to take advantage of syntactic language models based on context-free grammars (Charniak et al., 2003; Marcu et al., 2006; Wu and Wong, 1998).

2.4.1.2 Translation Models

Taking advantage of the statistical machinery that we have introduced, we now turn to the *translation model* $P(f_1^J, d_1^M|e_1^I)$. As we have seen, we can use the chain rule to decompose this into a series of smaller models.

$$P(f_1^J, d_1^M|e_1^I) = \prod_{j=1}^{J} P(f_j|f_1^{j-1}, d_1^M, e_1^I) \times \prod_{m=1}^{M} P(d_m|d_1^{m-1}, e_1^I) \tag{2.7}$$

This equation tells us that the conditional probability of the pair (f_1^J, d_1^M) with respect to e_1^I is simply the product of many small probabilities, each of which corresponds to a single action taken by our translational equivalence model. Thus, in our FST model, there is a probability for each transition in each transducer; in our SCFG model, there is a probability for each production. Using our notation for conditional independence, we can now rewrite Equation 2.7.

$$P(f_1^J, d_1^M|e_1^I) = \prod_{j=1}^{J} P_\delta(f_j|f_1^{j-1}, d_1^M, e_1^I) \times \prod_{m=1}^{M} P_\delta(d_m|d_1^{m-1}, e_1^I) \tag{2.8}$$

Assuming that we have made strong enough independence assumptions, each distribution on the right side of this equation contains a sufficiently small number of terms that we can actually learn them. At runtime, we will simply look up the values associated with the terms and use them to compute the function. Each of these values is a *parameter* of our model. In other words, a parameter is an element of the smallest distribution that we represent in our models—a distribution that we do not represent as a function of other distributions. We will use $p(x,y)$ or $p(x|y)$ to denote parameters. We already saw a parameter in the section on language modeling: $p(e_i|e_{i-n}^{i-1})$ was a parameter.

We do not have space to describe all of the parameterizations that have been proposed for even the small selection of translational equivalence models we described in §2.3. However, we will use a slightly simplified version of IBM Model 4 as an example to illustrate parameterization of generative models. Recall the steps of this model.

1. Each target word e_i selects a fertility ϕ_i and copies itself ϕ_i times.
2. Each copy of each target word is translated to a single source word.
3. The source words are reordered into their final positions.

Let us now take a look a parameterization of this model. We know that we want to assign a probability to each step taken by the model. Therefore, we will need a *fertility probability*, a *word translation probability*, and some probability to control the reordering, which we call a *distortion probability*.

Fertility is simple. We can make it conditionally dependent on the word identity. We define the fertility probability for word e_i by the parameter $p(\phi_i|e_i)$. We can think of this as the probability that a human translator would choose ϕ_i source words to translate the target word e_i.

Word translation is equally simple if we limit ourselves to the identities of the words being translated. Let $\tau_{i,k} \in V_F$ be the translation of the kth copy of e_i. The word translation probability is then $p(\tau_{i,k}|e_i)$. This is quite intuitive. We can think of it as the probability that a translator, when presented with word e_i, will choose to translate it using word $\tau_{i,k}$.

Let T denote the set of all random variables $T = \{\tau_{i,k} : 0 \leq i \leq I, 0 \leq k \leq \phi_i\}$ representing target word translations.

Modeling the reordering step is a bit more complicated. We will use two parameters to control this. The first parameter controls the placement of the *first* word generated by e_i, $\tau_{i,1}$. Let random variable $\pi_{i,1} \in [1,J]$ denote its final position. IBM Model 4 models this according to the distribution $p(\pi_{i,1} - \odot_{i-1}|C_E(e_{i-1}), C_F(\tau_{i,1}))$. Here, $\pi_{i,1} \in [1,M]$ represents the final absolute location of the translated word and $\odot_{i-1} = \frac{1}{k}\lceil\sum_{k=1}^{\phi_{i-1}} \pi_{i-1,k}\rceil$ represents the average location of all translations of e_{i-1}. In other words, we make its position dependent on the positions of the previously translated word. The functions $C_E : E \rightarrow [1,K]$ and $C_F : F \rightarrow [1,K]$ partition the vocabularies V_E and V_F onto suitably small sets of K classes to avoid the sparseness that would arise from conditioning on the words themselves (Brown et al., 1992; Och, 1999).

Just as we condition the positioning of the first translated word $\tau_{i,1}$ on the position of the translations of the adjacent target word, we condition the positioning of $\tau_{i,k}$ on $\pi_{i,k-1}$. Let this be controlled by the distribution $p(\pi_{i,k} - \pi_{i,k-1}|C(\tau_{i,k}))$. We use Π to denote the set of all random variables $\Pi = \{\pi_{i,k} : 0 \leq i \leq I, 0 \leq k \leq \phi_i\}$ representing word positions.

We can now show the complete parameterization of the IBM Model 4.[12]

[12] For simplicity, we leave out the parameterization of null translation.

$$
\begin{aligned}
P(\mathbf{f}|\mathbf{e}) = \sum_{T,\Pi} & \prod_{i=0}^{I} p(\phi_i|e_i)\times & \text{fertility} \\
& \prod_{i=0}^{I}\prod_{k=1}^{\phi_i} p(\tau_{i,k}|e_i)\times & \text{translation} \\
& \prod_{i=0}^{I} p(\pi_{i,1} - \odot_{i-1}|C_E(e_{i-1}), C_F(\tau_{i,1}))\times & \text{distortion for } \tau_{i,1} \\
& \prod_{i=0}^{I}\prod_{k=2}^{\phi_i} p(\pi_{i,k} - \pi_{i,k-1}|C(\tau_{i,k}))\times & \text{distortion for } \tau_{i,k}, k > 1
\end{aligned}
$$

As we can see from this discussion, the parameterization of generative models is closely tied to the form of the underlying translational equivalence model. However, many of these parameters have obvious analogues in other models. For instance, phrase-based models are parameterized using *phrase translation probabilities*, which apply to pairs of phrases, and are analogous to word translation probabilities (Koehn et al., 2003; Marcu and Wong, 2002). Numerous other parameterizations of distortion have been proposed for FST models (Al-Onaizan and Papineni, 2006; DeNero and Klein, 2007; Lopez and Resnik, 2005; Och and Ney, 2000; Toutanova et al., 2002; Vogel et al., 1996). The parameterization of SCFG models follows a similar pattern of diversity.

2.4.2 Discriminative Models

Generative models are useful in decoding (§2.6) because they correspond so closely to the translational equivalence models that define the search space. However, there are tradeoffs. As we have seen, to make them both theoretically well-founded and tractable, we must make very strong independence assumptions. This means that the information that we can bring to bear at each individual decision point is very limited. For instance, we are generally limited to translation between small numbers of words in each sentence, although we expect in principle that knowledge of all words in a source sentence may help us translate any particular word. Using generative models, there is no tractable mechanism to represent this.

We can bring additional context into modeling by moving from generative to *discriminative* models. In SMT, a popular form for this is *log-linear* modeling (Berger et al., 1996b; Och and Ney, 2002).[13] The introduction of log-linear models to SMT follows from their increasing use in NLP (Berger et al., 1996b; Ratnaparkhi, 1998; Smith, 2006), and reflects general trends in machine learning.

Log-linear models define a relationship between a set of K fixed *features* $h_1^K(\mathbf{e},\mathbf{d},\mathbf{f})$ of the data and the function $P(\mathbf{e},\mathbf{d}|\mathbf{f})$ that we are interested in. A feature can be any function $h : E^* \times D^* \times F^* \longrightarrow [0,\infty)$, that maps every pair of input and output strings to a non-negative value. An example of a feature might be the number of times a particular word pair (e_i, f_j) appears in the data (Och and Ney, 2002); the number of phrases in a segmentation of e_1^I (Koehn, 2004c); or the logarithm of the probability defined by a generative model (or even one of its distributions) from the previous section. Most features used in SMT to date have taken the latter form. Log-linear models take the form of Equation 2.9.

$$
P(\mathbf{e},\mathbf{d}|\mathbf{f}) = \frac{\exp\sum_{k=1}^{K}\lambda_k h_k(\mathbf{e},\mathbf{d},\mathbf{f})}{\sum_{\mathbf{e}',\mathbf{d}':Y(\mathbf{e}',\mathbf{d}')}\exp\sum_{k=1}^{K}\lambda_k h_k(\mathbf{e}',\mathbf{d}',\mathbf{f})} \tag{2.9}
$$

[13] In the more general literature this is often simply called a *linear model*.

The daunting normalization factor in the denominator is required only to make the function a well-formed probability. Fortunately, we can ignore it during decoding because it constant for any given f_1^J. Its computation may or may not be required during parameter estimation, depending on the algorithm.

The log-linear model defined in Equation 2.9 has K parameters, λ_1^K.[14] These are called *feature weights* or *model scaling factors*. They determine the contribution of a feature to the overall value of $P(\mathbf{e},\mathbf{d}|\mathbf{f})$. Ideally, each parameter would indicate the pairwise correspondence between the feature and the output probability. A positive value λ_k should indicate that the feature $h_1^K(\mathbf{e},\mathbf{d},\mathbf{f})$ correlates with $P(\mathbf{e},\mathbf{d}|\mathbf{f})$; a negative value should indicate an inverse correlation; and a value near zero should indicates that the feature is not a useful predictor of $P(\mathbf{e},\mathbf{d}|\mathbf{f})$. However, in practice, if two features are highly correlated with each other, an estimator might distribute the weight in any fashion between them. If a feature is uncorrelated with the output, then the estimator might assign an arbitrary weight to it. This complicates parameter estimation and motivates the task of *feature selection*, which seeks to identify the smallest set of the most predictive features. Feature selection is a very open problem in SMT.

Note that Equation 2.4 corresponds to a special case of Equation 2.9 when the following conditions hold.

$$
\begin{aligned}
K &= 2 \\
\lambda_1 &= \lambda_2 = 1 \\
h_1(\mathbf{e},\mathbf{f}) &= \log P(\mathbf{f},\mathbf{d}|\mathbf{e}) \\
h_2(\mathbf{e},\mathbf{f}) &= \log P(\mathbf{e})
\end{aligned}
$$

Log-linear models *discriminate* between different possible values e_1^I when presented with a particular f_1^J. In contrast with generative models, there is no requirement that we assign a single probability to every element of data. We may assign multiple probabilities to an element or none at all. In fact, the values that we assign are not required to be well-formed probabilities at all— the normalization factor in Equation 2.9 takes care of this for us. In particular, we are not required to define probabilities for our input data as we do in generative models. Because we are freed from the constraints of Bayes' rule, features can be overlapping—we could, for instance, use several of the generative models discussed previously, even though each of them will have a different explanation of each word in the target language. In principle, any other model of $P(\mathbf{e})$, $P(\mathbf{f}|\mathbf{e})$, $P(\mathbf{e}|\mathbf{f})$, or $P(\mathbf{e},\mathbf{f})$, or combination thereof, can be a feature.[15] A popular technique is to simply take the summed logarithm of all instances of a particular distribution in an underlying generative model. This yields a small number of features, usually less than ten. Another important feature of most models is the *word count* or *word penalty* feature, which is the number of words in the target sentence. Its weight therefore controls target sentence length.[16]

Discriminative modeling is powerful because it frees us from the generative modeling requirement that each term must conform to an event in our translational equivalence model, which is often chosen for computational reasons rather than for its ability to distinguish between good

[14] Confusingly, in the general literature this is sometimes called a *parameter-free* model. It is also known as a *distribution-free* model, which can be understood from the fact that the normalization is not strictly required.

[15] A justification for using log-linear models in this way was that a system based on $P(\mathbf{e}) \cdot P(\mathbf{e},\mathbf{d}|\mathbf{f})$ worked nearly as well as $P(\mathbf{e}) \cdot P(\mathbf{f},\mathbf{d}|\mathbf{e})$ in empirical studies, even though the former cannot be theoretically motivated using Bayes' rule (Och and Ney, 2002; Och et al., 1999). Beginning with (Koehn, 2004a), many systems use both (Chiang, 2007; Marcu et al., 2006; Simard et al., 2005). It is not clear if this is beneficial (Lopez and Resnik, 2006).

[16] Lopez and Resnik (2006) found this feature to be quite important. This is partly due to the use of evaluation metrics that are very sensitive to length §2.7.

translations. This allows us to define arbitrary features that may help to improve translation. The primary art in discriminative modeling is defining useful features. However, with some exceptions this area has not been fully explored (Liang et al., 2006a; Marcu et al., 2006; Och et al., 2004a,b; Venugopal et al., 2007). Many models use little more than a word penalty feature and a small set of generative features that can be traced directly to the IBM Models.

2.5 Parameter Estimation

Once we have defined $P(\mathbf{e}, \mathbf{d}|\mathbf{f})$, we need to assign values to its parameters, the actual values that are used to compute it. We call this *parameter estimation*. In SMT, we use a parallel corpus as input to a machine learning algorithm in order to learn the parameter values. Broadly speaking, we can say that SMT relies on *supervised learning*, because we are given samples of input/output pairs.

Most SMT systems use a log-linear model of $P(\mathbf{e}, \mathbf{d}|\mathbf{f})$ that incorporates generative models as feature functions. Before we can learn the parameters of the log-linear model, we must fix values of the feature functions, including any generative models used as features. This means that we must first estimate any underlying generative models independently, and then separately estimate the parameters of the log-linear models. An method for iteratively estimating both is presented by Fraser and Marcu (2006).

We describe parameter estimation for generative models in §2.5.1. We will then discuss the important concept of word alignment in §2.5.2. Finally, we describe parameter estimation for log-linear models in §2.5.3.

2.5.1 Parameter Estimation in Generative Models

An example parameter from our generative models is the translation probability $p(\text{风}|\text{wind})$. Our model says that we will use this value whenever we translate the word "风" as "wind". To estimate this parameter, we turn to our parallel corpus.

One way to capitalize on our data comes to us from statistical estimation theory. We assume that the parallel corpus was produced by our model using the unknown, true parameter values. Our goal, then, is to *estimate* those values so that our estimates are as close as possible to the true ones.

If we denote our training data as $C \subset \{E^* \times F^*\}$, the complete set of parameters as Θ, and the probability (or likelihood) of C under parameter set Θ as $P_\Theta(C)$, then our goal is to choose Θ that satisfies Equation 2.10.

$$\Theta = \underset{\hat{\Theta}}{\operatorname{argmax}} P_{\hat{\Theta}}(C). \tag{2.10}$$

Parameter estimation, then, is equivalent to finding the maximum of a function (the *objective function*) —in this case, the *likelihood* function $P_\Theta(C)$. We call this *maximum likelihood estimation* (MLE). The parameter set Θ that satisfies the maximization problem in Equation 2.10 is the *maximum likelihood estimate*. This is not the only possible objective function, but it is the one that is typically used for generative models. We will discuss a different objective function in §2.5.3.1.

2.5.1.1 Learning Word Translation Probabilities

Recall that in our generative models, each probability is tied to a single decision taken by the model. MLE is easy when we can observe all of these decisions. Consider a very simple model, the coin-flip model. In this model, we have a coin that comes up heads with probability $p(h)$. We

do not know $p(h)$ and we would like to guess what it is. If we have access to the coin itself, we can flip it a number of times and see how many times each side comes up. Suppose that we flip the coin a number of times, and we count the number of times it comes up heads, which we denote $\#(h)$. The total number of flips is the sum of the number of heads and tails, $\#(h+t)$. Most people know intuitively that the value for $p(h)$ should be $\#(h)/\#(h+t)$. In fact, we can show analytically that this relative frequency estimate corresponds to the MLE. The accuracy of MLE depends crucially on the number of examples—we can see that if we flip the coin only once, then the only possible outcomes are $p(h) = 1$ and $p(h) = 0$, either of which is likely to be far from the true value of the parameter. This issue has not received much attention in SMT, although Foster et al. (2006) show that methods to *smooth* poorly estimated probabilities can improve performance.[17][18]

Now suppose that we wish to estimate the parameters of a word-based generative translation model. If we had access to an alignment—such as the one depicted in Figure 2.2—for every sentence in our corpus, then it would be easy to count the fertility, substitution, and distortion outcomes for each word, and we could estimate our parameters as easily as we did in the coin-flipping model. For instance, if we saw the word "wind" $\#(\text{wind})$ times, and it was aligned to the word "风" $\#(a(\text{风},\text{wind}))$ times, then we would compute $p(\text{风}|\text{wind}) = \#(a(\text{风},\text{wind}))/\#(\text{wind})$.

It is not that easy, however. The data that we collect from the real world contains only sentence pairs, not alignments.[19] So, while we can see that "wind" and "风" both *cooccur* in many of these sentence pairs, we cannot see how many times they are actually aligned to each other. We can make some estimates based only on cooccurrence—for instance, we will estimate $p(f|e) = 0$ for words f and e that never cooccur in our training data. How can we estimate the probability for words that *do* cooccur?

One solution to this problem is to automatically generate an alignment, and then to use this alignment as our training data for maximum likelihood estimation. We will describe word alignment methods in §2.5.2. Alternatively, we need a method to estimate our parameters that will work even when we cannot explicitly count all of the decisions that we are interested in. Since we will not be able to directly observe the outcome of each decision made by our model, we can view the learning of the associated parameters as a form of *unsupervised learning*. A method that is commonly used to solve this problem in SMT is the Expectation-Maximization (EM) algorithm (Dempster et al., 1977). It works by substituting observed counts of events with *expected counts*. Although we cannot actually observe the number of times that "wind" aligns to "风", we can compute the expected number of times that this happens if we have some initial value for Θ, which we will call Θ_0. Let Θ_0 be random, uniform, or initialized from some simpler model. We can then compute the expected count of the event $E(a(\text{风},\text{wind}))$ in any particular sentence pair $(\mathbf{e},\mathbf{f})$ as follows.

$$E(a(\text{风},\text{wind})) = \frac{P_{\Theta_0}(a(\text{风},\text{wind}), f_1^J|e_1^I)}{P_{\Theta_0}(f_1^J|e_1^I)}$$

In other words, we compute all possible alignments between f_1^J and e_1^I, and their probabilities under Θ_0. We then sum the probability of those alignments that contain the decision that we are interested in, and divide this by the summed probability of all possible alignments. This gives a

[17] A discussion of smoothing is highly relevant, as it is to most NLP problems, but well beyond the scope of this paper. Chen and Goodman (1998), Manning and Schütze (1999), and Jurafsky and Martin (2000) and are good starting points.

[18] In fact, the unsmoothed values used by most phrase-based models are so imprecise that they can be stored in four bits without loss of performance (Federico and Bertoldi, 2006; Och, 2005).

[19] In fact, we have even glossed over the conversion of raw text data to sentence pairs, which is not an entirely trivial problem (Gale and Church, 1993; Smith, 2002).

fractional expected probability that the event occurred. If we apply this method to all parameters over the entire corpus, we can produce a new estimate, Θ_1. Under certain conditions this offers the minimal guarantee that $P_{\Theta_1}(C) > P_{\Theta_0}(C)$. Thus, it works by *hill-climbing*, or constantly improving its estimate of Θ. We can define EM according to the following recursion, for our observed training data C and unknown alignment data D.

$$\Theta_i = \underset{\hat{\Theta}}{\operatorname{argmax}} P_{\hat{\Theta}}(C, E_{\Theta_{i-1}}(D)) \tag{2.11}$$

The EM algorithm does not, in general—and in particular for most SMT models— guarantee convergence to a globally optimal value for Θ. In fact, it depends crucially on a good initial estimate Θ_0 to avoid a poor local maximum. A method for generating this estimate for the IBM Models is sketched in §2.5.2. Despite the various difficulties in using EM, it has been applied to a variety of other NLP problems (Lari and Young, 1990; Merialdo, 1994).

A full discussion of the EM algorithm is beyond the scope of this paper. For a more detailed overview, refer to Dempster et al. (1977). For a full account of the analytical solution to the EM algorithm in the case the IBM Models, refer to Brown et al. (1993).

2.5.1.2 Learning Phrase Translation Probabilities

In order to train the parameters of a phrase-based model we must have access to a *phrase-to-phrase alignment*. EM for phrase-based models involves many approximations and tradeoffs (Birch et al., 2006; DeNero et al., 2006; Marcu and Wong, 2002). A pragmatic solution is to generate a word alignment (§2.5.2), and then count all phrases that are consistent with the alignment (Koehn et al., 2003; Och et al., 1999). We then compute the MLE using the hypothesized phrases as our observed events. We say that a bilingual phrase is consistent with a word alignment if no word inside the phrase is aligned to any word outside the phrase—in other words, the phrase contains the transitive closure of some set of nodes in the bipartite alignment graph. This is illustrated in Figure 2.12.

2.5.1.3 Learning Parameters of Generative SCFG Models

As we have seen, SCFG models usually contain a word translation probability, which can be learned using EM under the specific model (Wu, 1996; Yamada and Knight, 2001), or using word-translation probabilities learned from other alignments (§2.5.2) and a supervised learning method. Melamed (2004a) presents a series of steps whereby parallel corpora and parsing statistics can be coordinated to learn the parameters of an arbitrary SCFG, using MLE (including EM) where necessary. Hierarchical phrase-based grammars can be learned using a supervised method similar to the one used for finite-state phrase-based models (Chiang, 2005, 2007).

2.5.2 Interlude: Word Alignment

We need a method for word alignment as a precursor to most of our learning methods in generative models. Word alignment is a microcosm of translation: we need a model, parameter estimation, and a search algorithm. The word alignment task can be viewed as a warm-up for decoding, since it is more constrained—in word alignment, we need only find a correspondence between sequences, whereas in decoding we will be required to find both the correspondence and the target sequence.

Over the past decade, a number of additional uses have been proposed for word alignment, including the automatic acquisition of bilingual dictionaries (Melamed, 1996) which can be used

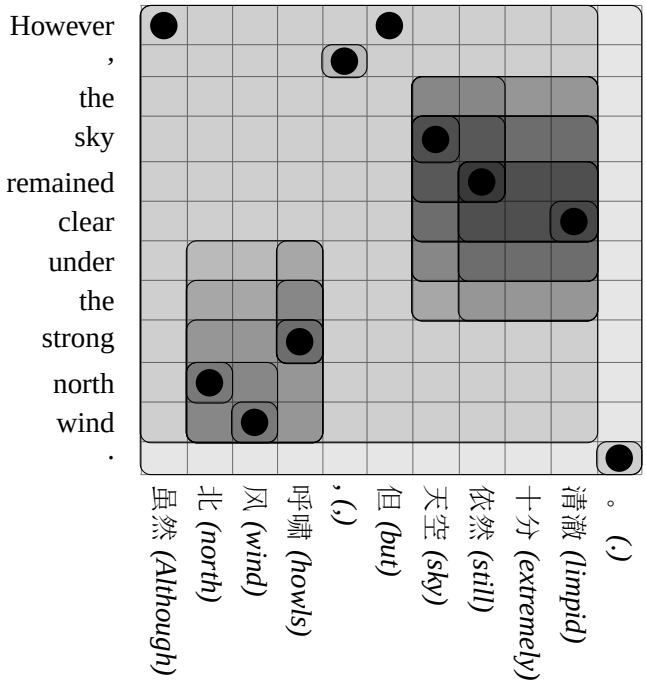


Figure 2.12: Supervised learning of phrases from word alignments. Here, we view each sentence pair on a grid. Word alignment is indicated by the presence of dark circles in the grid point corresponding to the word pair. The rectangles outline bilingual phrase pairs that are consistent with this word alignment.

in cross-language information retrieval (Wang, 2005); and cross-lingual syntactic learning (Hwa et al., 2005; Smith and Smith, 2004; Yarowsky et al., 2001). For this reason, word alignment has become a topic of significant study in its own right. The remainder of this section provides a brief overview of the word alignment task. We will return to the more general topic of parameter estimation in §2.5.3.

2.5.2.1 Formal Definition

Formally, we say that the objective of the word alignment task is to discover the word-to-word correspondences in a sentence pair (e_1^I, f_1^J). The alignment A of this pair is simply a set of these correspondences. We say that $A \subset [1,I] \times [1,J]$. If $(i,j) \in A$, then word e_i is aligned to word f_j. Models for word alignment depend on the way in which they decompose this problem.

2.5.2.2 Asymmetric Models

Recall that in our word-based translational equivalence model (§2.3.1.1) an asymmetry exists in the alignment between e_1^I and f_1^J. In particular, each source word f_j corresponds to one and only one target word (or null). The target words are unconstrained, and each can link to an arbitrary number of words (even zero), defined by its fertility. When we are generating an initial alignment to train this model, we must observe the same constraints.

In fact, we can exploit this asymmetry to produce an efficient alignment algorithm by mod-

eling the alignment directly. To do this we introduce the alignment variable $\mathbf{a}$ to which we must assign a value a_1^J. In this representation each element a_j is a value in the range $\{0,1,...,I\}$. The value of a_j represents the position of the target word e_{a_j} to which f_j corresponds. By making the very strong assumption that each variable a_j is independent, we arrive at Equation 2.12, which tells us how to find the optimal alignment $\mathbf{a}$.

$$\mathbf{a} = \underset{a_1^J}{\operatorname{argmax}} \prod_{j=1}^{J} p(a_j) \cdot p(f_j|e_{a_j}) \tag{2.12}$$

This is the form of IBM Model 1 (Brown et al., 1993). If we make the additional simplifying assumption that the distribution $p(a_j)$ is uniform, the only parameters that are required in order to compute the optimal alignment are the word translation parameters $p(f_j|e_i)$. Note that our independence assumptions reduce the model to a set of J independent decisions each with $I+1$ possible outcomes. In this simple form, the space of all possible alignments can be compactly represented, and the EM search is guaranteed to converge to a single solution (Brown et al., 1993). Although this convergence will guarantee an optimal value for $P_\Theta(C)$, this optimal value may not produce the best alignments in practice, because maximizing likelihood does not necessarily guarantee a reduction in error. This is particularly true if the model makes too many independence assumptions, as Model 1 does. Moore (2004) proposes an alternative method of Model 1 parameter estimation that produces better results in practice.

Note that we can compute $P(\mathbf{f}|\mathbf{e})$ as a sum over Model 1 alignments as follows:

$$P(\mathbf{f}|\mathbf{e}) = \prod_{j=1}^{J} \sum_{a_j=0}^{I} p(a_j) \cdot p(f_j|e_{a_j}) \tag{2.13}$$

Thus Model 1 is a translation model, although it will not produce very good translations on its own (Knight, 1999a). However, it is useful as a feature function in log-linear models, most likely because it computes a correspondence between all source and target words (Lopez and Resnik, 2006; Och et al., 2004b).

We obtain better alignments when we move to a first-order dependency between the alignment variables, as in Equation 2.14 (Vogel et al., 1996).

$$\mathbf{a} = \underset{a_1^J}{\operatorname{argmax}} \prod_{j=1}^{J} p(a_j|a_{j-1}) \cdot p(f_j|e_{a_j}) \tag{2.14}$$

Equation 2.14 is in the basic form of a Hidden Markov Model (HMM). HMMs have been applied to numerous problems in NLP, such as part-of-speech tagging (Merialdo, 1994). A key benefit of HMMs is that standard algorithms for EM parameter estimation (Baum, 1972) and maximization (Viterbi, 1967) are widely known. HMMs have been the subject of several studies in word alignment (DeNero and Klein, 2007; Lopez and Resnik, 2005; Och and Ney, 2000; Toutanova et al., 2002). In general, they are very accurate, significantly outperforming IBM Models 1, 2, and 3 in detailed empirical studies (Och and Ney, 2000, 2003). HMMs are a common form of a *sequence model* that assigns a label to each element of a sequence. In the case of alignment, the sequence is the source sentence and the labels are the target words to which each source word corresponds. HMMs are generative models. A discriminative relative, the *conditional random field* has also been used for alignment (Blunsom and Cohn, 2006).

Finally, we can use IBM Model 4 itself to perform alignment. Search is done by first generating a good alignment with a simpler model, and then modifying it using hill-climbing techniques in conjunction with the IBM Model 4 parameters (Brown et al., 1993; Och and Ney,

2003). The translation parameters can also be imported from a simpler model; this makes IBM Model 4 highly dependent on the models used to bootstrap it (Lopez and Resnik, 2005; Och and Ney, 2003). Och and Ney (2003) note that the likely reason for the good performance of Model 4 is the first-order reordering dependence in the distortion parameters, and proposes combining it with the HMM, which has a complementary first-order dependence. This is accomplished by using both models as features in a log-linear framework.

The IBM Models for alignment are implemented in the open-source toolkit GIZA and its successor GIZA++ (Al-Onaizan et al., 1999; Och and Ney, 2003).[20] They are widely used in the SMT research community for a variety of purposes, including parameter learning for other models (Koehn et al., 2003; Och et al., 1999; Smith and Smith, 2004; Yarowsky and Ngai, 2001). Various improvements to Model 4 have been proposed (Dejean et al., 2003; Fraser and Marcu, 2006, 2007b).

2.5.2.3 Symmetric Alignment Models

The alignment models that we have described so far are asymmetric, following IBM Model 4. This is a necessity if we plan to train a translation model with a corresponding asymmetry. However, many models are symmetric. We would like symmetric alignments as well.

One approach to symmetric alignment is to align our training corpus twice using an asymmetric method, applying the asymmetry to each side in turn. We symmetrize by combining these two alignments. This is done via set union, set intersection, or a number of heuristic methods, which usually begin with the intersection and proceed by iteratively adding links from the union (Koehn et al., 2003; Och et al., 1999). Matusov et al. (2004) present a symmetric word alignment method based on linear combination of complementary asymmetric word alignment probabilities. Ayan and Dorr (2006a) investigate the effect of various symmetrization heuristics on the performance of phrase-based translation.

An alternative is to simply use an alignment algorithm that explicitly generates symmetric alignments. In this case, the alignment task corresponds to solving $I \cdot J$ binary decision problems: one for each potential link in the set A. The complexity of this space depends on any constraints we put on the links. With no constraints, the problem reduces to a set of binary decision problems and is tractable under a wide variety of models and learning algorithms (Ayan and Dorr, 2006b; Ayan et al., 2005a,b; Liang et al., 2006b). A common constraint is to require that each word in either sentence be linked exactly once, or to null (Melamed, 2000). This constraint produces an exponential space of allowable alignments because decisions are not independent of each other. A solution to this is to use a greedy search algorithm called competitive linking (Melamed, 2000). A number of cooccurrence-based correlation metrics have been used to score each link in this algorithm (Cherry and Lin, 2003; Gale and Church, 1991; Melamed, 2000; Moore, 2005b).

Fraser and Marcu (2007b) extended the IBM Models to produce symmetric many-to-many alignments that can be viewed as phrase alignments.

2.5.2.4 Supervised Learning for Alignment

Although the alignment learning methods that we have described so far depend on unsupervised learning of the alignment model parameters, it is possible to learn alignment models using supervised learning. Callison-Burch et al. (2004) construct an experiment showing that alignment with the IBM Models could be significantly improved with supervised learning. However, a primary limitation of supervised learning for alignment is that the number of sentences that have been

[20] GIZA++ is available from `http://www.fjoch.com/GIZA++.html`.

aligned by human annotators is nearly always several orders of magnitude smaller than the number of unannotated sentences. Supervised learning algorithms must learn from a few hundred or thousand annotated sentences. Contrast with unsupervised learning, where we typically have access to hundreds of thousands or millions of sentences. Therefore supervised learning of alignments is highly dependent on models which are sufficiently general, with a compact set of parameters. The solution to this is to use discriminative models with rich feature sets that do not depend heavily (or at all) on the specific identities of the words being aligned. In particular, it is unrealistic to expect such models to learn weights for word-to-word features, since we not have enough training data to populate the tables. However, we can use probabilities learned using unsupervised methods as features in a discriminative model.

Numerous discriminative alignment models have been proposed, based on a wide variety of machine learning methods. Most of these methods depend on supervised training, and depend on the availability of a small set of manually produced alignments. Learning methods include transformation-based learning (Ayan et al., 2005b), neural networks (Ayan et al., 2005a), maximum margin estimation (Taskar et al., 2005), perceptron learning (Moore, 2005a), and log-linear models (Fraser and Marcu, 2006; Ittycheriah and Roukos, 2005). When annotated alignments are available, these methods outperform unsupervised methods according to common alignment metrics, and sometimes in downstream translation results.

2.5.2.5 Evaluation of Word Alignment

The ultimate measure of word alignment is in its contribution to parameter estimation of our translation models. If one alignment method produces a better translation system than another, we might conclude that it is more accurate overall.

Word alignment is used for tasks other than SMT parameter estimation, so other task-based evaluations might be applicable. Although this is preferable, it is common to evaluate word alignment intrinsically, by comparison with alignments prepared by human annotators. Most of these test sets contain a few hundred sentences. They are available in several languages (Melamed, 1998; Mihalcea and Pedersen, 2003; Och and Ney, 2000). Ideally, each sentence is aligned by multiple annotators and the results are combined in some way. In much of the reported literature, the annotations contain two sets of links. The *sure* set S contains links about which all annotators agreed. The *probable* set P is a superset of S that additionally contains links about which annotators disagreed or expressed uncertainty about, such as "idiomatic expressions, free translations, and missing function words" (Och and Ney, 2000). However, Fraser and Marcu (2007a) found that the use of probable links reduced the ability of alignment metrics to predict translation accuracy. They recommend an annotation style that does not contain them (Melamed, 1998).

Given the set of hypothesized alignment links A, we compute the *precision* $|A \cap P|/|A|$ corresponding to the fraction of accurate links in the hypothesized alignment, and the *recall* $|A \cap S|/|S|$ corresponding to the fraction of "true" links were discovered by the alignment algorithm.[21] A widely used metric that combines these statistics is the *alignment error rate* (AER) given in Equation 2.15 (Och and Ney, 2000).

$$AER = 1 - \frac{|S \cap A| + |P \cap A|}{|S| + |A|} \tag{2.15}$$

A similar metric was proposed by Ahrenberg et al. (2000).

Although intrinsic evaluation of word alignments is popular, the exact relationship between alignment evaluations and SMT performance is not entirely clear. Several studies report poor

[21] Precision and recall metrics are common in NLP evaluation.

correlation between alignment performance and MT performance (Callison-Burch et al., 2004; Ittycheriah and Roukos, 2005; Koehn et al., 2003), and a number of researchers have investigated the relationship directly (Ayan and Dorr, 2006a; Fraser and Marcu, 2007a; Lopez and Resnik, 2006). In particular, Fraser and Marcu (2007a) advocate *unbalanced* F-measure as a better predictor of SMT performance than AER. The F-measure is given in Equation 2.16.

$$F = \frac{|S \cap A|}{\alpha|S| + (1-\alpha)|A|} \tag{2.16}$$

The parameter α is used to move balance towards either precision or recall. Fraser and Marcu (2007a) show that α can be tuned to more reliably predict translation accuracy.

2.5.3 Estimation in Log-Linear Models

We now return to the subject of estimating translation model parameters. One we have estimated the parameters of all of our generative models, we can turn our attention to the estimation of the log-linear feature weights λ_1^K (§2.4.2). This is usually done on a training set separate from the one used to learn underlying generative model probabilities.

As in generative models, the maximum likelihood objective (Equation 2.10) can be used to train the feature weights. A nice property of log-linear models is the availability convenient objective function obtained via the *maximum entropy principle* (Berger et al., 1996b).[22] It corresponds to the maximum likelihood objective function and has a single optimum point, which we can find using an iterative search method called *generalized iterative scaling* (Darroch and Ratcliff, 1972). The training of log-linear SMT models is a supervised learning problem, since we are given inputs and the corresponding best output, and all features are known. Unfortunately, the normalization factor represented by the denominator of Equation 2.9 must be computed for the MLE, and this is expensive to compute even in the supervised case because it involves a sum over all possible translations. Och and Ney (2002) show that the N-best output of the previous parameter setting can be used to approximate this sum.

2.5.3.1 Minimum Error-Rate Training

Automatic evaluation metrics for MT have become widespread (§2.7). They facilitate a different method of parameter estimation: *minimum error-rate training* (MERT; Och, 2003). In MERT, we assume that the best model is the one that produces the smallest overall error with respect to a given error function. Unfortunately, determining the amount of error in a translation is not a well-defined problem with an objective answer, and numerous error metrics have been proposed. However, Och (2003) shows empirically that we achieve best results for any particular error function when we use that function in our objective function under MERT. This suggests that we can improve the accuracy of our SMT systems simply by devising an error function that more closely corresponds to human judgements of translation error, or with some task-based notion of accuracy. Ideally, this means that SMT researchers can focus on the question of what makes a good translation, instead of what makes a good translation model (a task fraught with many orthogonal considerations). With MERT, better evaluation functions should lead directly to better translation.

Formally, we say that if we are given an error function $E(\hat{\mathbf{e}}, \mathbf{e})$ defining the amount of error in some hypothesized translation $\hat{\mathbf{e}}$ with respect to a known good actual translation $\mathbf{e}$, then the objective function is:[23]

[22] For this reason, log-linear models are often called maximum entropy models in the NLP literature.

[23] As we will see in §2.7, we sometimes have access to multiple good translations of $\mathbf{f}$. It is straightforward to modify

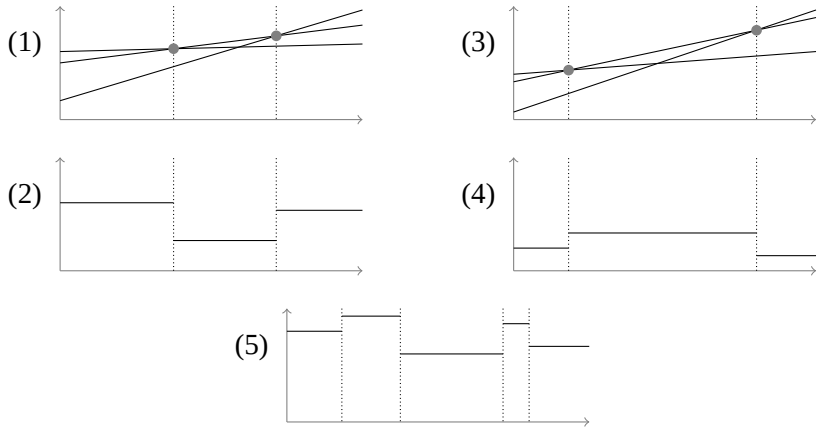


Figure 2.13: Illustration of the MERT line minimization algorithm for optimizing a single parameter. (1) For each candidate translation $\hat{\mathbf{e}}$, compute $P_{\lambda_k}(\hat{\mathbf{e}}|\mathbf{f})$ as a function of λ_k, and find the intervals at which the optimal candidate changes. (2) Using these intervals, compute $E_{\lambda_k}(\text{argmax}_{\hat{\mathbf{e}}} P(\hat{\mathbf{e}}|\mathbf{f}), \mathbf{e})$ as a function of λ_k. (3) and (4) Repeat this procedure for each sentence. (5) Add the single-sentence error functions (2) and (4) to compute the aggregate error function for both input sentences. To optimize, we simply walk along all intervals of the aggregate function until we determine the minimum.

$$\lambda_1^K = \underset{\hat{\lambda}_1^K}{\text{argmin}} \sum_{(\mathbf{e},\mathbf{f})\in C} E(\underset{\hat{\mathbf{e}}}{\text{argmax}} P_{\hat{\lambda}_1^K}(\hat{\mathbf{e}}|\mathbf{f}), \mathbf{e}) \tag{2.17}$$

The optimization contains an argmax operator, which precludes calculation of a gradient. Although there is no way to find a guaranteed optimal solution under these circumstances, we can find a good solution using the method sketched in Och (2003), which we describe in greater detail here due to its widespread use. Pseudocode appears in Algorithm 1.

The MERT algorithm works by iteratively generating random values for λ_1^K, which it then tries to improve by minimizing each parameter λ_k in turn while holding the others constant. At the end of this optimization step, the optimized λ_1^K yielding the greatest error reduction is used as input to the next iteration.

The single-parameter line minimization algorithm at the core of MERT is illustrated in Figure 2.13. It is based on the observation that if we hold all but one parameter λ_k constant, then $P(\mathbf{e}|\mathbf{f})$ for any given pair $\mathbf{e}$ and $\mathbf{f}$ is $P(\mathbf{e}|\mathbf{f}) = \lambda_k h_k(\mathbf{e},\mathbf{f}) + (\sum_{k'\neq k} \lambda_{k',m} h_{k'}(\mathbf{e},\mathbf{f}))$. Note that the second term of the sum is constant, making the function linear in λ_k. Using the intersections of these lines for all candidate translations in a decoder's N-best list for a single input sentence, the algorithm exhaustively computes a representation of the piecewise linear function $E(\text{argmax}_{\hat{\mathbf{e}}} P(\hat{\mathbf{e}}|\mathbf{f}), \mathbf{e}))$. Assuming that our error function is additive, we simply sum over all input $(\mathbf{e},\mathbf{f}) \in C$ to compute the complete function that we are trying to minimize.[24] We then select the midpoint in the interval which minimizes the function.

our functions to accommodate this and it has no impact on the mathematics.

[24] Sometimes this function is not additive, as is the case with the commonly used BLEU score (Papineni et al., 2002). Usually, however, the function is computed in terms of aggregate values over the training set which are additive. If this is the case, we simply keep track of all of the additive values which are used to compute the error function over each interval, and then perform the computation once all intervals are known.

Algorithm 1 Minimum Error Rate Training

Input initial estimate $\lambda_{1,0}^K$ ▷ Uniform or random
Input training corpus C
$\lambda_1^K = \lambda_{1,0}^K$
$E_{best} = \sum_{(\mathbf{e},\mathbf{f})\in C} E(\text{argmax}_{\hat{\mathbf{e}}} P_{\lambda_1^K}(\hat{\mathbf{e}}|\mathbf{f}), \mathbf{e})$ ▷ Decode
repeat
  Generate M random estimates $\lambda_{1,1}^K, ..., \lambda_{1,M}^K$ ▷ To avoid poor local maximum
  for $m = \{0, 1, ..., M\}$ **do**
    for $k = \{1, 2, ..., K\}$ **do**
      $\lambda'_{k,m} = \text{LINE-MINIMIZE}(k, \lambda_{1,m}^K, C)$
      $E_{k,m} = \sum_{(\mathbf{e},\mathbf{f})\in C} E(\text{argmax}_{\hat{\mathbf{e}}} P_{\lambda_{1,m}^{k-1}\lambda'_{k,m}\lambda_{k+1,m}^K}(\hat{\mathbf{e}}|\mathbf{f}), \mathbf{e})$ ▷ N-best list
      if $E_{k,m} < E_{best}$ **then**
        $\lambda_1^K = \lambda_{1,m}^{k-1}\lambda'_{k,m}\lambda_{k+1,m}^K$
        $E_{best} = E_{k,m}$
  $\lambda_{1,0}^K = \lambda_1^K$
  $E_{best} = \sum_{(\mathbf{e},\mathbf{f})\in C} E(\text{argmax}_{\hat{\mathbf{e}}} P_{\lambda_1^K}(\hat{\mathbf{e}}|\mathbf{f}), \mathbf{e})$ ▷ Decode
until no change in λ_1^K
return $\lambda_{1,0}^K$

function LINE-MINIMIZE(k, λ_1^K, C)
  $E_{\lambda_k}(C) = 0$
  for all $(\mathbf{e},\mathbf{f}) \in C$ **do**
    for all $\hat{\mathbf{e}} \in$ DECODER-N-BEST($\mathbf{f}$) **do**
      $m_{\hat{\mathbf{e}}} = h_k(\hat{\mathbf{e}}, \mathbf{f})$ ▷ slope of $P_{\lambda_k}(\hat{\mathbf{e}}, \mathbf{f})$
      $b_{\hat{\mathbf{e}}} = \sum_{k'=1}^{k-1} \lambda_{k'} \cdot h_{k'}(\hat{\mathbf{e}}, \mathbf{f}) + \sum_{k'=k+1}^{K} \lambda_{k'} \cdot h_{k'}(\hat{\mathbf{e}}, \mathbf{f})$ ▷ intercept of $P_{\lambda_k}(\hat{\mathbf{e}}, \mathbf{f})$
    $i = 0$
    $\Delta[i] = -\infty$ ▷ left interval boundary
    $e[i] = \text{argmin}_{\hat{e}}\, m_{\hat{\mathbf{e}}}$ ▷ equivalent to $\text{argmax}_{\hat{\mathbf{e}}} \lim_{\lambda_k \to -\infty} P(\hat{\mathbf{e}}, \mathbf{f})$
    repeat
      $i = i + 1$
      $\Delta[i] = \min_{\hat{\mathbf{e}}}$ X-INTERSECT$(m_{\hat{e}}, m_{e[i-1]}, b_{\hat{e}}, b_{e[i-1]}) > \Delta[i-1]$
      $e[i] = \text{argmin}_{\hat{\mathbf{e}}}$ X-INTERSECT$(m_{\hat{e}}, m_{e[i-1]}, b_{\hat{e}}, b_{e[i-1]}) > \Delta[i-1]$
    until No more intersection points found
    $\Delta_{i+1} = \infty$
    $E_{\lambda_k}(\text{argmax}_{\hat{\mathbf{e}}} P(\hat{\mathbf{e}}|\mathbf{f}), \mathbf{e}) = \{\lambda_k \to E(\hat{\mathbf{e}}, \mathbf{e}) : \hat{\mathbf{e}} = e[i], \Delta[i] \le \lambda_k \le \Delta_{i+1}\}$
    $E_{\lambda_k}(C) += E_{\lambda_k}(\text{argmax}_{\hat{\mathbf{e}}} P(\hat{\mathbf{e}}|\mathbf{f}), \mathbf{e})$
  return $\lambda_k = \text{argmin} E_{\lambda_k}(C)$

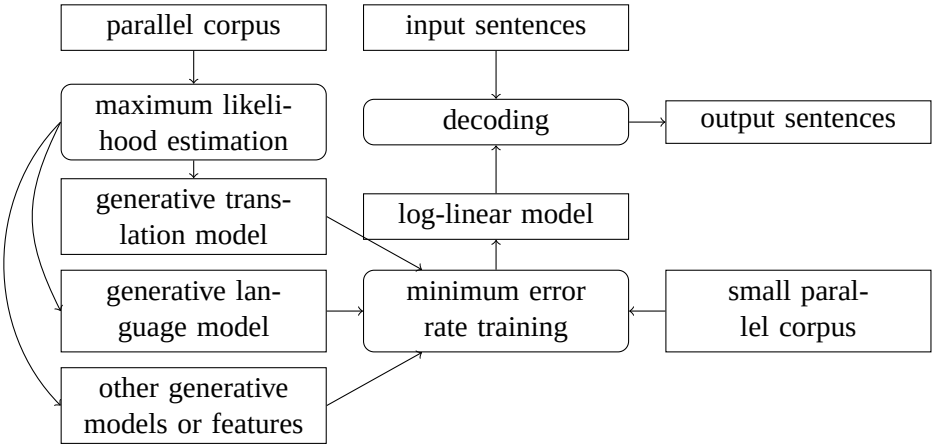


Figure 2.14: The flow of data, models, and processes commonly involved in the deployment of an SMT system.

2.5.3.2 Purely Discriminative Training

Most current state-of-the-art SMT systems use log-linear models with a small number of generative submodels and use MERT in order to optimize whatever error function is chosen for evaluation. An overview of the architecture used in these systems is shown in Figure 2.14. This approach is not *purely* discriminative; it uses generative model estimates as input to a discriminative learner that optimizes a small number of feature weights. In pure discriminative learning, features are usually binary or integral. For instance, we might define a word pair feature $h(e, f)$ as follows:

$$h(e, f) = \begin{cases} 1 \text{ if the input contains } f \text{ and the output contains } e \\ 0 \text{ otherwise} \end{cases}$$

Under this definition, the weight given to this feature by the combined generative and discriminative training procedure outlined above is $\lambda \log p(f|e)$. However, as we have noted, $p(f|e)$ is estimated to maximize likelihood, not translation performance. We might instead wish to assign a weight to this feature that is estimated to directly optimize translation performance. This is the goal of pure discriminative learning, which can be accomplished by a number of different algorithms. Examples include the perceptron algorithm (Liang et al., 2006a), large margin learning (Tillmann and Zhang, 2006; Watanabe et al., 2007), decision tree learning (Wellington et al., 2006a), and transductive learning (Ueffing et al., 2007). Pure discriminative learning is promising, but there are still a number of significant obstacles to overcome, most notably the ability to scale to the very large datasets and billions of parameters required for SMT. The present approaches are quite slow compared to generative model estimation and MERT.

2.6 Decoding

Now that we have a model and estimates for all of our parameters, we can translate new input sentences. This is called decoding. In principle, decoding corresponds solving the maximization problem in Equation 2.18.

$$\mathbf{e} = \underset{(\hat{\mathbf{e}}:Y(\hat{\mathbf{e}},\mathbf{d}))}{\operatorname{argmax}} P(\hat{\mathbf{e}},\mathbf{d}|\mathbf{f}) \tag{2.18}$$

We call this the *decision rule*. Equation 2.18 is not the only possible decision rule, although it is by far the most common. Alternative decision rules are presented in Kumar and Byrne (2004) and Venugopal et al. (2005).

This is a difficult optimization. Recall that $P(\mathbf{e},\mathbf{d}|\mathbf{f})$ ranges over $\{E^* \times D^* \times F^*\}$. Even though $\mathbf{f}$ is fixed, and even though the number of possible outputs $(\mathbf{e},\mathbf{d})$ is finite due to the constraints of our translational equivalence model, there is still a very large number of them to consider in order to maximize the function. Therefore, a primary objective of decoding is to search this space as efficiently as possible.

There are two types of decoders, corresponding to our two broad types of translational equivalence models: FST and SCFG.

2.6.1 FST Decoding

Nearly all approaches to finite-state decoding follow a general framework described by Wang and Waibel (1997) and (Koehn, 2004a). It is a generalization of speech recognition algorithms originating in information theory Jelinek (1969).

In this algorithm, search proceeds through a directed acyclic graph of states representing partial or completed translation hypotheses, which are constructed from left-to-right in the target language word order. An example graph is depicted in Figure 2.15. Each state consists of the following elements.

1. A coverage set $C \subseteq \{1,2,...,J\}$ enumerates the positions of the source string f_1^J that have been translated.

2. If using an n-gram language model, the $n-1$ most recently generated target words are kept for computing the n-gram language model component of the probability. These words and the subset C constitute the state's signature.

3. The cost h of our partial hypothesis is computed as the combination of model costs associated with the hypothesis. This will be fairly straightforward for any generative model based on the underlying translational equivalence model, since we will be reconstructing the events that occur in that model, and we can simply apply the associated probabilities. It may or may not be difficult for a discriminative model, depending on the specific feature functions.

4. The estimated cost g of completing the partial hypothesis is computed heuristically. Because this computation must be done quickly, we usually use only the single-best word-to-word (or phrase-to-phrase) costs in this heuristic function (Koehn, 2004a).

Hypotheses in this space are extended by adding one or more source word indices to the coverage set and appending one or more target words to the hypothesis string to produce a new state. This corresponds to the translation of the newly covered source words by the newly generated target words. We apply model probabilities accordingly to update the partial cost h. We implement model-specific extension operators to apply this algorithm to IBM Model 4 (Germann et al., 2004; Tillman and Ney, 2003), phrase-based models (Koehn, 2004a), or any number of other finite-state translation models (Nießen et al., 1998; Wang and Waibel, 1997).

In order to organize the search space, hypotheses may be stored in one or more priority queues, usually corresponding to either the cardinality $|C|$ of the coverage set, or to the coverage sets themselves (Tillman and Ney, 2003).[25] This is done to ensure that comparisons between

[25]This priority queue is often called a *stack* in literature, and the algorithm that we describe is called *stack decoding*, although its central object is technically not a stack. The terminology dates back to Jelinek (1969).

(1)

虽然	北	风	呼啸	，	但	天空	依然	十分	清澈	。
Although	*north*	*wind*	*howls*	*,*	*but*	*sky*	*still*	*extremely*	*limpid*	*.*
1	2	3	4	5	6	7	8	9	10	11

(2)

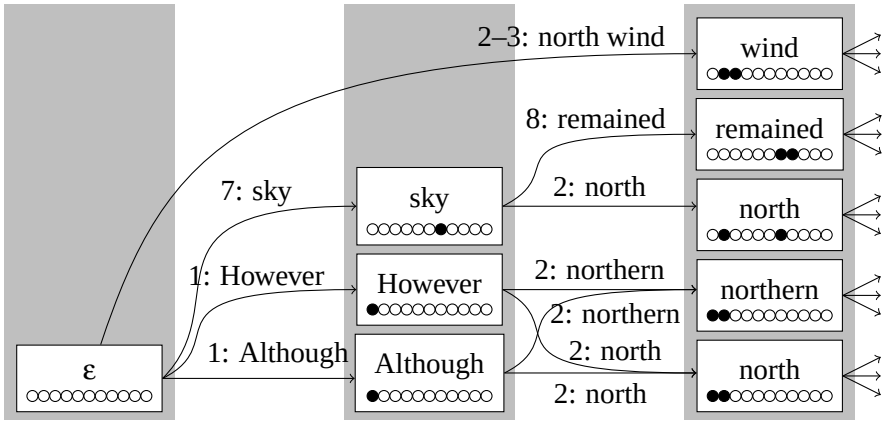


Figure 2.15: Illustration of search in a finite-state decoder. The input sentence (1) generates a large search graph, partially illustrated in (2). In this illustration, each arrow represents extension of a hypothesis by appending the words on the arrow. To recover the best translation, we traverse the highest scoring path. In each state, we show the coverage set and most recently generated target word, which is needed for computation of a bigram language model. Note that states can only be combined if the coverage set and the most recently produced words match. Items with the same number of covered words are stored in the same stack.

hypotheses—used for sorting and pruning purposes within each priority queue— are done on hypotheses of relatively equal depth in the search space. (Wang and Waibel, 1997) If we were to compare hypotheses of unequal length, our heuristic functions, which favor shorter hypotheses, will cause more complete hypotheses to be pruned from the priority queue prior to full evaluation.

Each hypothesis contains a backpointer to the hypothesis that generated it. If two hypotheses have matching signatures, only the higher-scoring hypothesis is kept (Koehn, 2004a; Och et al., 2001). This is a risk-free optimization because the set of all extensions to these two hypotheses will be the same; therefore the higher-scoring partial hypothesis is guaranteed to generate a higher-scoring completed hypothesis.

The search space defines a finite-state word lattice, in which we can find the score of any particular hypothesis by traversing the lattice (Koehn, 2004a; Ueffing et al., 2002). We can use standard finite-state methods for finding the best path (or paths) through this lattice. It is possible to directly implement such decoders as a cascade of weighted finite-state transducers (Knight and Al-Onaizan, 1998; Kumar et al., 2006). These transducers will differ from the ones we describe in §2.3.1. However, the decoding algorithm we have described does, in principle, reverse the set of transductions represented by those models; we can see, for instance, that it reconstructs the English sentence in the order that it was fed into the transducer, at each step consuming the source words that were created by transductions over the associated target word or words.

A variant algorithm allows the target language hypothesis to be extended to either left or right (Watanabe and Sumita, 2002).

2.6.1.1 Optimality and Pruning

Using A^* heuristics, we can solve the optimization in Equation 2.18 exactly (Och et al., 2001). Germann et al. (2004) illustrate how we can also do this by converting the problem to a linear integer programming problem and using standard tools to solve it. Due to the large number of translations for each word or phrase, even with limited reordering this can be very slow. Fortunately, optimal search is not strictly necessary, because there are many good translations of any sentence. If many of these receive high probability under our model, then we may safely permit a certain amount of *search error*. Search error occurs when the decoder does not choose the globally highest-scoring hypothesis according to the model, but rather some other high-scoring hypothesis that can be found more quickly. We can optimize for speed by *pruning* the search graph (Koehn, 2004a; Tillman and Ney, 2003). In *threshold pruning*, any hypothesis with a probability less than t times the probability of the best estimate in the same priority queue is removed. In *histogram pruning*, only the n best hypotheses are kept in any priority queue. Search with pruning is sometimes called *beam search*, where t or n is the size of the *beam*. With a well-tuned beam size, we gain large speedups with very little loss of accuracy (Germann et al., 2004; Koehn, 2004a; Tillman and Ney, 2003; Zens and Ney, 2004).

2.6.1.2 Greedy Decoding

An alternative to standard finite-state decoding is greedy decoding (Germann, 2003; Germann et al., 2004; Marcu and Wong, 2002). In greedy decoding, we generate an initial hypothesis by substituting each source word with the highest-probability target word, using the original target word order. This gives us a complete word-for-word gloss of the source sentence. We then use hill-climbing heuristics in an attempt to find higher-scoring hypotheses by considering neighboring translations produced by changing the order or translation of one or two words at a time, and choosing the highest-scoring neighbor. This new hypothesis becomes the starting point for the next iteration of the algorithm. The algorithm terminates when no higher-scoring hypothesis can be found. With some optimizations, it algorithm runs in time nearly linear in target sentence length (Germann, 2003). A tradeoff of is that the search error rate is much higher than stack decoding (Germann et al., 2004).

2.6.2 SCFG Decoding

Decoding with SCFG models is equivalent to CFG parsing (Melamed, 2004a). The goal is to infer the highest-scoring tree that generates the input sentence using the source side of the grammar, and then read off the tree in target order. Most practical SCFG decoders are straightforward extensions of dynamic programming algorithms for parsing monolingual context-free grammars (Chiang, 2007; Marcu et al., 2006; Venugopal et al., 2007; Wu and Wong, 1998; Yamada and Knight, 2002; Zens and Ney, 2003). A benefit of this is that the standard algorithms and optimizations that have been developed for CFG parsing can be applied to SMT (Melamed, 2004a).

SCFG decoding works by attempting to cover larger and larger *spans* of the input sentence. A span is simply a contiguous sequence of words. States in the search space consist of a span, a nonterminal symbol which covers the span, and any language model information needed to combine spans (Chiang, 2007; Melamed, 2004b; Zhang et al., 2006a). In order to construct larger spans, we find SCFG productions whose right-hand sides match a sequence of nonterminals that we have already inferred to cover a set of smaller, adjacent spans. Once we have constructed the full source language parse, we produce output using an in-order traversal based on target language ordering of the tree. This is illustrated in Figure 2.16.

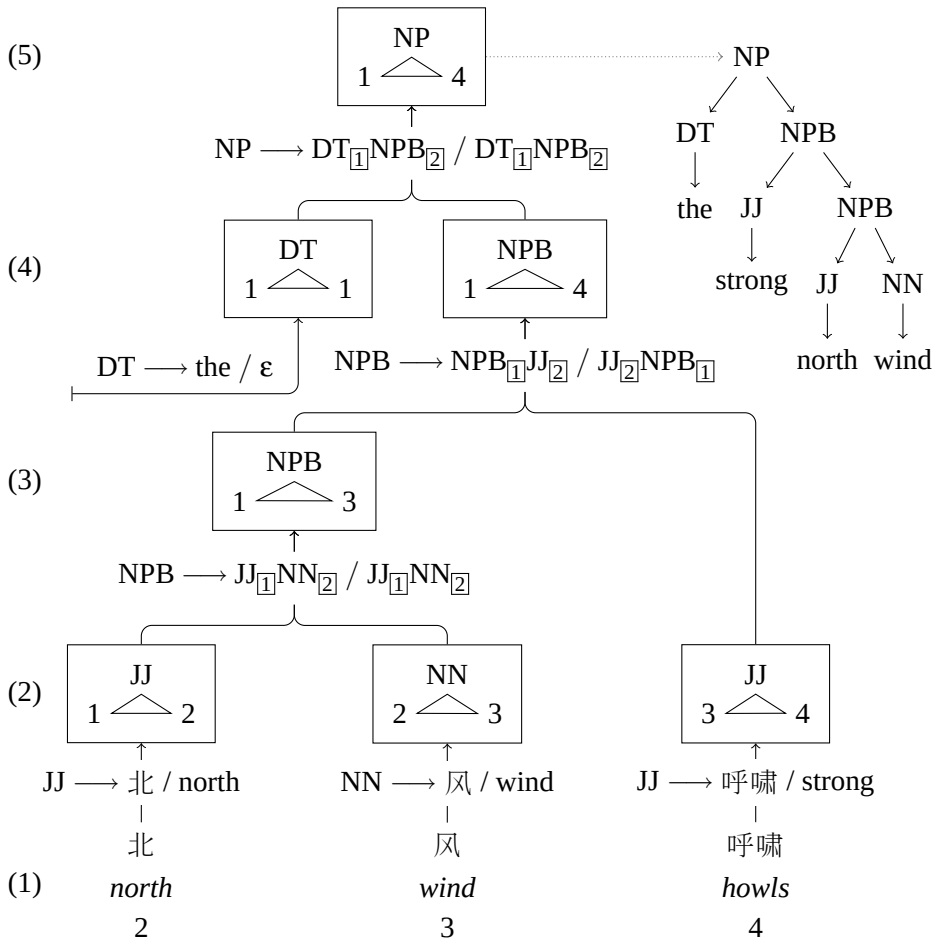


Figure 2.16: Illustration of SCFG decoding. (1) Scan each source word and associate it with a span. (2) Apply SCFG rules that match the target spans. (3) Recursively infer larger spans from smaller spans. (4) Optionally infer any target language words with no matching span in the source language. (5) Read off the tree in target-language order.

There are $O(J^2)$ possible spans in the source sentence, and they can be computed in polynomial time. It is easy to see from this that SCFG decoding is, *in principle*, much less computationally expensive than FST decoding with full reordering. However, most practical FST decoders allow only a limited amount of reordering, and in practice they are often much faster than SCFG decoders.[26] Search optimization for these models is therefore an active area of research.

Chiang (2007) describes a optimization called *cube pruning* that prevents excessive combination of hypotheses in adjacent subspans. Zhang et al. (2006a) describe a method for *binarizing* rules containing more than two nonterminals, which helps reduce grammar constants for parsing and simplifies *n*-gram language model integration. Venugopal et al. (2007) present a method based on delayed language model integration, in which the parse graph is first constructed quickly with simplified language model statistics, and then expanded in a second pass using a full language

[26]It is possible to apply reordering constraints of FST models to SCFG models. Chiang (2005, 2007) restricts hierarchical reordering to spans that are shorter than ten words. Spans longer than this are required to be monotonic orderings of smaller hierarchical phrases. This prevents some long-distance reordering.

model, following only the most promising paths. A number of other optimizations have also been investigated (Huang and Chiang, 2005, 2007)

2.6.3 Reranking

Even if there are no search errors and we produce the translation that exactly optimizes our decision rule, the translations produced by our decoder may not be the actual best translations according to human judgement. It is possible that the search space explored by the decoder contained a better translation, and our decoder assigned a lower score for this hypothesis because its estimate of $P(\mathbf{e}, \mathbf{d}|\mathbf{f})$ was incorrect. This is called *model error*.

One approach to reducing model error is *reranking* or *rescoring*. In reranking, we first run our decoder, and rather than merely returning the highest-scoring translation, we return N highest-scoring translations for some value N. These translations are then input to an alternative model with access to more feature functions than may be efficiently computed in our decoder, or which are otherwise difficult to incorporate. Hopefully, this alternative model can give us more accurate scores than the one used in decoding.

Reranking approaches to SMT are described in Och et al. (2004b) and Shen et al. (2004). Och et al. (2004b) show using oracle studies on decoder n-best lists that large gains in accuracy are possible with rescoring, although so far these are unrealized.

2.7 Evaluation

How can we know if the output of our SMT system is any good? Many methods have been proposed to evaluate MT output. Hovy et al. (2002) attribute to Yorick Wilks the remark that "more has been written about MT evaluation over the past 50 years than about MT itself". In the discussion that follows, we will narrowly focus on methods that have figured prominently in the evaluation of statistical systems.

Traditionally accepted measures of MT evaluation have required examination of MT system output by human judges, who rank the *adequacy* of the translation in conveying the source language meaning and the *fluency* of expression in the target language (White et al., 1994). More ideal than this are measures which determine how well some human task can be performed when the human subject is provided with machine translated text. If possible, we would optimize for task-related metrics directly. Unfortunately, human metrics require time and money. This usually rules out their use in iterative system development, where we will need to perform regular evaluation to determine if changes are beneficial to performance. The next best thing is to develop automatic metrics that closely correlate with human evaluations. The closer that these metrics are to the real objective, the better our performance on that objective will be after we apply discriminative training (§2.5.3).

A common element of automatic metrics is their use of a set of test sentences for which we already have human translations, called *reference translations*. They can come from a parallel corpus, although we must take care to use a separate set of sentences from the set we used for training. The intuition behind metrics based on reference sentences is that MT must be good if it closely resembles human translation of the same sentence (Papineni et al., 2002). These metrics are based on partial string matching between the output and the reference translations, as illustrated in Figure 2.17. However, the use of a single reference may bias the evaluation towards a particular translation style. In order to mitigate against this and reflect the diversity of possible good translations, we may use multiple references. This requires the use of human translators to produce the additional references, but it is a one-time cost.

Although the northern wind shrieked across the sky , but was still very clear .

However , the sky remained clear under the strong north wind .
Although a north wind was howling , the sky remained clear and blue .
The sky was still crystal clear , though the north wind was howling .
Despite the strong northerly winds , the sky remains very clear .

Figure 2.17: Example of partial string matching used for most evaluation methods. Here we show a single output hypothesis compared with four reference translations. Sequences of words in the hypothesis that match sequences in any of the reference translations are highlighted. Likewise, sequences of words in each reference that are found in the hypothesis are highlighted. Most evaluation metrics are based on functions of counts of these matches.

One metric for evaluation is the well-known Levenshtein or edit distance, which is borrowed from ASR evaluation, where it is known as the *word error rate* (WER) (Och et al., 1999). The WER sums the number of insertions, deletions, and substitutions required to transform an output sentence into the reference sentence. Unfortunately, this metric is less appropriate for MT than ASR, because it does not recognize word reorderings. A word that is translated correctly but in the wrong location will be penalized as a deletion (in the output location) and an insertion (in the correct location). This problem motivates the use of *position-independent word error rate* (PER), which is similar to WER but does not penalize reorderings, because it regards the output and reference sentences as unordered bags of words rather than totally ordered strings (Och et al., 1999).

The most widely used metric is the *bilingual evaluation understudy* (BLEU; Papineni et al., 2002). BLEU considers not only single word matches between the output and the reference sentence, but also n-gram matches, up to some maximum n. This allows it to reward sentences where local word order is closer to the local word order in the reference. BLEU is a *precision*-oriented metric; that is, it considers the number of n-gram matches as a fraction of the number of total n-grams in the output sentence. Let $\#(g)$ be the count of an n-gram g in a particular hypothesis sentence $\hat{e}$, and $\#_{clip}(g)$ be the maximum number of times that g appears in any corresponding reference sentence. We can compute the n-gram precision p_n for a set of hypothesis translations H.

$$p_n = \frac{\sum_{\hat{e} \in H} \sum_{g \in ngrams(\hat{e})} \#_{clip}(g)}{\sum_{\hat{e} \in H} \sum_{g' \in ngrams(\hat{e})} \#(g')}$$

To get a better idea of the accuracy, we combine multiple n-gram precisions, up to some maximum n, by taking the geometric average $\sum_n \log p_n$. This biases the metric towards translations with fewer words, because denominator contains the total number of hypothesis n-grams. To correct this defect, the metric includes a *brevity penalty*, which penalizes output sentences that are much shorter than the reference. It compares the overall number of words h of the entire hypothesis set with *effective reference length* r, created by summing the lengths of the closest reference sentences to each candidate sentence.[27] This gives us BLEU.

[27] The NIST evaluation uses an alternative definition of effective reference length, always choosing the shortest reference.

$$BP = \begin{cases} 1 & \text{if } h > r \\ e^{(1-r/h)} & \text{otherwise} \end{cases}$$

$$BLEU = BP \cdot \exp\left(\sum_n \log p_n\right)$$

Automatic evaluation is an active research area. A number of other metrics based on word matching include *precision* and *recall* (Melamed et al., 2003), and length of the longest common subsequence (Lin and Och, 2004). METEOR enhances token matching with weighted matching based on morphological or semantic similarity (Banerjee and Lavie, 2005). Translation edit rate (TER; Snover et al., 2006) computes an edit distance between hypotheses and human-corrected versions of those hypotheses. The intuition is that it corresponds to "the amount of work needed to correct the translations." It is an fully automatic approximation to human TER (hTER), a true task-based metric which measures the amount of work done by human post-editors.

It is important to note when interpreting metrics such as BLEU that they can be used to rank systems relative to each other, but the scores are generally uninterpretable as absolute measures of correctness. A key element of most research in this area is the identification of metrics that correlate with human rankings of systems in controlled studies (Callison-Burch et al., 2007; Papineni et al., 2002). Since this correlation is important, a natural line of research involves the use of machine learning to optimize metrics for correlation (Albrecht and Hwa, 2007; Kulesza and Shieber, 2004; Lita et al., 2005; Liu and Gildea, 2007; Russo-Lassner et al., 2005).

It is not always clear when a difference in scores between two systems represents a significant difference in their output. Koehn (2004b) describes a method to compute statistical confidence intervals for most automatic metrics using bootstrap resampling.

BLEU has been highly influential in SMT research. It is extensively used SMT literature, and it has been used as the basis for a number of comparative evaluations (Callison-Burch et al., 2007; Doddington, 2002; Koehn and Monz, 2005, 2006). It is commonly used in the objective function for minimum error-rate training (Och, 2003).

Use of BLEU is controversial. Turian et al. (2003) and Callison-Burch et al. (2006b) provide counterexamples to its claimed correlation with human judgement. Other problems have been illustrated by construction (Callison-Burch et al., 2006b). Despite controversy, automatic evaluation has had a profound impact on progress in SMT research, and it is likely to continue.

With the proliferation of available metrics, it is not always clear which one to use. Practical considerations such as comparison with previous benchmarks encourages continued use of BLEU, despite criticism. The use of discriminative training depends on computationally simple metrics, including BLEU, METEOR, and TER. Correlation with human judgement is also a desirable characteristic. For a good contemporary evaluation of several metrics in this regard across several language pairs, refer to Callison-Burch et al. (2007).

2.8 Current Directions and Future Research

There are many common elements in the best systems, although there is also growing diversity. Most can be characterized as follows: phrase-based models (in either the FST or SCFG framework); log-linear models with a small set of generative features; and discriminative training. The success of these methods is seen in academic workshops (Callison-Burch et al., 2007; Koehn and Monz, 2005, 2006) and the yearly NIST evaluations.

All of these methods were popularized very quickly after their initial introduction. SMT has made swift progress and there is great optimism for future success. Nonetheless, there are

many hurdles and open questions in the field.

Most of the community evaluations in SMT focus the translation of news and government texts. There is very little work on open-domain translation, particularly for informal genres—which describes much of the information found on the Internet, and for which translation is in demand. Although it is possible to mine data from the Web (Resnik and Smith, 2003), this resource is underutilized. Since statistical methods are sensitive to both domain differences and noise, the move to informal text and Internet data will present many interesting challenges.

Application of SMT to language pairs with very little parallel text presents an interesting challenge. Bannard and Callison-Burch (2005) and Callison-Burch et al. (2006a) describe a novel method for solving this problem by learning paraphrases of the source language using a parallel text in a third language, and applying these paraphrases to generate sentences that can be translated by an impoverished SMT system.

Another understudied problem is the translation of English into other languages. In the United States, research focuses almost exclusively on translation from other languages into English. This is dictated by government funding, but has the effect of obscuring deficiencies in the current approaches. For instance, it is easier to map morphologically rich languages such as German and Arabic onto a relatively morphologically simple language such as English. This can be seen as a movement from a higher-dimensional to a lower dimensional space, where some loss of meaning and nuance is harmless. Translation in the other direction requires much more attention to this issue (Goldwater and McClosky, 2005; Minkov et al., 2007; Nießen and Ney, 2004; Schafer and Drabek, 2005). Koehn and Hoang (2007) and Koehn et al. (2007) describe *factored models*, a framework for modeling with morphology and other annotations.

Evaluation of MT systems will continue to be a focus, since discriminative training illustrates the importance of metrics that correspond to human judgement. However, most popular metrics provide very little insight into the typical errors made by any particular system, as they only produce a single aggregate score over an entire test set. They are especially useless for identifying sentence-level errors since they provide only an aggregate measure of accuracy. For this reason, the relative merits and drawbacks of different models with respect to different types of translation error are not well understood. Error analysis techniques have not been substantially explored, although it has recently been identified as an important task (Och, 2005). A few techniques for error analysis (Chiang et al., 2005; DeNeefe et al., 2005; Popovic et al., 2006) and confidence estimation (Ueffing and Ney, 2005) have been investigated, but in general this area remains underexplored.

The fundamental issues in SMT will remain a focus of all future research. Refinements to modeling techniques and parameter estimation methods will no doubt continue. New developments in machine learning will increasingly be applied to machine translation, although additional work is needed to scale them up to data sizes commonly used in SMT. There is also increasing interest in the incorporation of linguistic knowledge into models and parameter estimation. As we described, syntactic modeling is an area of active research. There are also some steps toward semantic modeling (Carpuat and Wu, 2005, 2007; Chan et al., 2007).

2.9 Conclusions

This chapter has presented a comprehensive tutorial overview of statistical machine translation. To cover a wide variety of approaches, some parts of the discussion have been left abstract. In particular, we have ignored the practical details of efficient implementation of these models. However, increasingly large knowledge sources and the increasingly complex models that exploit

them place growing pressure on these algorithms to scale efficiently. The remainder of this dissertation will describe an innovative solution to this problem, allowing current models to scale far beyond the current state of the art.

3 Machine Translation by Pattern Matching

Calvin: *You can't just turn on creativity like a faucet. You have to be in the right mood.*

Hobbes: *What mood is that?*

Calvin: *Last-minute panic.*

–Bill Watterson

Statistical MT models typically contain many millions of parameters. So far in our discussion, the implications of this have been abstract. We now turn our attention to the implementation of translation models. Our main concern will be efficient representation of and access to model parameters.

The number of model parameters depends on training data size and model complexity. Trends toward larger data and more articulated models place increasing pressure on implementations. If we rely on direct representation of a model trained on large data, it is already quite easy to create a translation system that exhausts the main memory of the commodity hardware on which most research systems are developed.

Callison-Burch et al. (2005) and Zhang and Vogel (2005) introduced an alternative that we call *translation by pattern matching*. It relies on indirect representation of the translation model, using the training data itself as its proxy. This representation is compact. More importantly, it is independent of model size, enabling us to scale to arbitrarily large models (Chapter 5). Rather than enumerate and compute all parameters of the model offline, translation by pattern matching works by efficiently searching for relevant sections of the training data at runtime, and extracting and computing the needed parameters from these sections. This efficient search is based on algorithms for pattern matching.

Although Callison-Burch et al. (2005) and Zhang and Vogel (2005) lay the foundation for pattern matching, there are some open problems. They did not demonstrate that they could match—let alone exceed—the performance of a common baseline system. In fact, a few elements of their approach appear to be incompatible with other methods used in the state of the art. Therefore, it is unclear whether translation by pattern matching is even a viable replacement for direct representation. In this chapter, we answer this question affirmatively.

This chapter is organized as follows. We first describe a standard baseline model (§3.1). We describe direct representation of the model and illustrate its limitations (§3.2). We introduce translation by pattern matching, a alternative to direct representation that generalizes previous work (§3.3). We resolve some open problems in the previous work (§3.4), and show that translation by pattern matching produces equivalent results to a direct representation (§3.5).

3.1 Baseline Model

In §2.4.2, we loosely described some common model features. Here, we describe the exact features of the widely used phrase-based system Pharaoh (Koehn, 2004a).[1] They have been widely replicated in several other related models (see, e.g. Chiang, 2005, 2007; Simard et al., 2005). There are eight features.

1. We take a logarithm of the joint probability of all target-to-source phrase translation probabilities used in the translation.

$$\log \prod_{(\tilde{e},\tilde{f})\in D} p(\tilde{f}|\tilde{e}) \tag{3.1}$$

This feature is roughly equivalent to the translation model in a Bayesian framework (§2.4.1.2).

2. We also take the logarithm of the equivalent source-to-target phrase translation probabilities.

$$\log \prod_{(\tilde{e},\tilde{f})\in D} p(\tilde{e}|\tilde{f}) \tag{3.2}$$

This feature is difficult to justify from a Bayesian perspective. However, Och et al. (1999) found that it could be used interchangeably with the target-to-source probabilities. In the Pharaoh baseline system, both features are used together. This combination is widely reproduced in other models.

3. We use the logarithm of a target-to-source lexical weighting feature.

$$\log \prod_{i=1}^{I} \left[\sum_{j:(i,j)\in A(D)} 1 \right]^{-1} \sum_{j:(i,j)\in A(D)} p(f_i|e_j) \tag{3.3}$$

This feature computes a word-to-word translation probability over aligned words in each phrase pair. It was introduced by Koehn et al. (2003), and it has been suggested by Foster et al. (2006) that it acts as a type of smoothing distribution for the target-to-source phrase translation probabilities, which are estimated from sparse data.

4. We use the logarithm of the source-to-target lexical weighting, following the same logic as for the target-to-source probabilities.

$$\log \prod_{j=1}^{J} \left[\sum_{i:(i,j)\in A(D)} 1 \right]^{-1} \sum_{i:(i,j)\in A(D)} p(e_j|f_i) \tag{3.4}$$

5. We use the logarithm of a trigram language model.

6. We use a *distortion count* feature. To compute this feature, we simply count the number of intervening words between source phrases that are translated consecutively in the target sentence (Koehn et al., 2003; Marcu and Wong, 2002).

7. We use a *phrase count* feature that counts the number of phrase pairs used.

8. We use a *word count* feature that counts the number of words in the target sentence.[2] It enables the model to control the average length of translation output, which is important to the BLEU evaluation criterion (Papineni et al., 2002, see also §2.7).

[1] Moses (Koehn et al., 2007), the successor to Pharaoh, uses nearly identical features. The main difference is that Moses uses a lexicalized distortion model (Koehn et al., 2005; Tillman, 2004). We examine the significance of this difference in §5.2.2.

[2] In the literature, these last two features are sometimes called the *phrase penalty* and *word penalty*, respectively.

$\tilde{f}$	$\tilde{e}$	$\log p(\tilde{f}\|\tilde{e})$	$\log lex(\tilde{f}\|\tilde{e})$	$\log p(\tilde{e}\|\tilde{f})$	$\log lex(\tilde{e}\|\tilde{f})$
北	north	0.88	0.53	0.69	0.43
	northbound	0.02	0.31	0.00	0.09
	northern side	0.33	0.15	0.00	0.00
北 风	north wind	0.33	0.16	0.16	0.05
北 约	nato	0.79	0.19	0.82	0.10
	by nato	0.43	0.19	0.00	0.10
依然	remained strong	0.07	0.01	0.00	0.00
	is still very	0.03	0.01	0.08	0.03
依然 充满	is filled with	0.05	0.00	0.25	0.00
十分	is still clear	1.00	0.00	0.11	0.00
	is still very	0.05	0.00	0.22	0.01

Figure 3.1: Example phrase table. It represents each source phrase along with each possible translation and the associated parameter values

We can easily eliminate the three count features (6–8) from further discussion. Each is a monotonic function of the derivation with no probabilistic parameters. This leaves us with the phrase translation, lexical weighting, and language model features, all requiring many parameters. We can further delimit the five probabilistic models into two groups: those that depend only on the phrase pair, and those that depend on additional context.

Of these features, only the language model probabilities depend on context outside the phrase pair. Although efficient representation of language models is important and highly relevant to our goal of scaling, large-scale language modeling is a separate body of research with applications beyond machine translation, including speech recognition, document classification, optical character recognition, and many more (Rosenfeld, 2000). For the remainder of this dissertation, we will focus on the four translation model features that depend only on the phrase pairs.[3]

3.2 Phrase Tables

In a phrase-based system, the set of translation rules is simply the complete set of bilingual phrase pairs (§2.3). Because the phrase translation and lexical weighting probabilities are dependent only on the rule identities, it is convenient to store them directly with the rules. All that we then require is efficient access to any translation rules containing a source phrase of the input sentence, which in turn gives us both its target phrases and the necessary parameters. The data structure containing these rules and parameters is called the *phrase table*. Abstractly, we can think of it as a table containing the source side and target side of each rule and all of their associated probabilities (Figure 3.1).

The phrase table can be implemented as a *prefix tree* (or *trie*; de la Briandias, 1959; Fredken, 1960) using the source side of each rule as a key. Formally, a prefix tree is an unminimized deterministic finite-state automaton recognizing all of the patterns in a set. Each node in the tree uniquely represents a prefix of one or more patterns. This prefix is identical to the concatenation of edge labels along the path from the root to the corresponding node.

In the prefix tree implementation of a phrase table, the set of patterns is simply the set of

[3] In §6.2 we briefly discuss related work in language model scaling. We simply note here that some of those techniques are similar to our techniques for translation models (e.g. Zhang et al., 2006b).

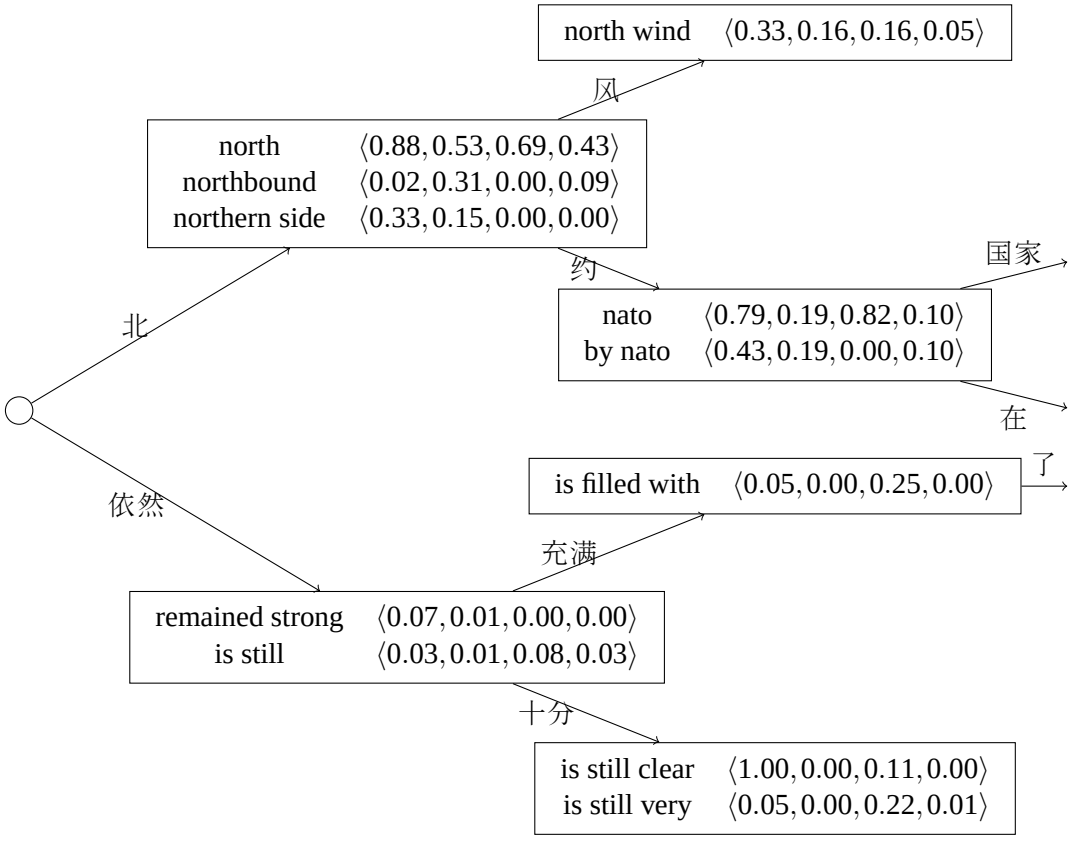


Figure 3.2: Prefix tree representation of the phrase table in Figure 3.1. Each unique source phrase is represented by a single node of the tree, which is found by traversing the path from the root that is labelled with the phrase. The node associated with a source phrase stores all of its possible target phrases, and a vector of scores for the phrase pair.

all source phrases in the model (Figure 3.2). This exploits the fact that in most cases, the prefix of a valid source phrase is itself a valid source phrase. To find an m-length source phrase in the tree, we traverse the m edges that spell out the phrase. Target phrases and associated scores are stored at the node.

This direct representation enables very fast lookup. A sentence of length J contains J^2 possible source phrases and lookup for a length m source phrase starting at the root requires $O(m)$ time. However, if we begin the search at the node representing the phrase's prefix, we need only traverse a single edge, which reduces lookup to constant $O(1)$ time. The upper bound on lookup time for all rules is therefore $O(J^2)$. This is a loose upper bound, since many phrases will not be found and we can terminate lookup as soon as any prefix of the phrase is not found. Since the constant factors in these lookup times are very small, the overall effect is that lookup takes only a very small fraction of the overall decoding time.

Fast lookup comes at a price. The space consumption of the prefix tree can be very high. Consider extraction of rules from a single training sentence. If each substring of the sentence is a valid source phrase, then we would extract up to J^2 source phrases from a sentence with J words. Since each source phrase corresponds to a prefix tree node, the prefix tree representation requires much more space than the sentence that produced it. This is not the only source of redundancy, as

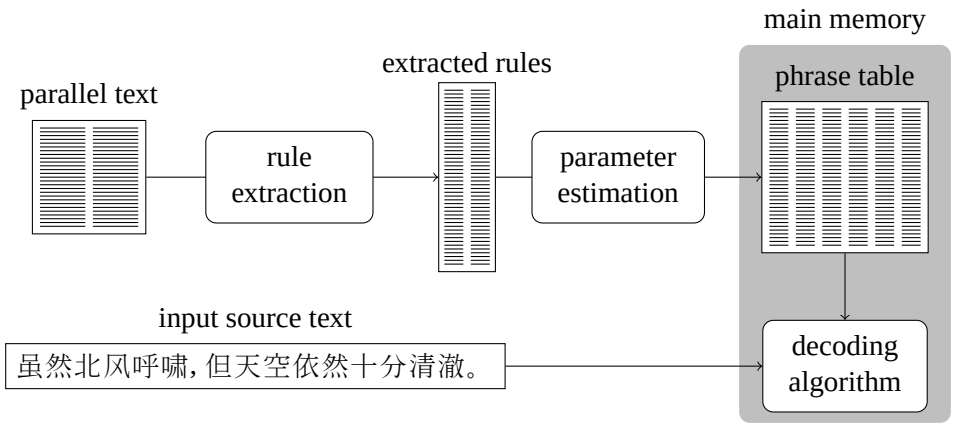


Figure 3.3: Architecture of a simple table-based decoder. This architecture requires that the entire model fit into main memory.

we can see from Figure 3.2. Target phrases at each node are represented independently, although the target phrases of a source phrase and its prefix are highly correlated and overlapping.

To make matters concrete, we estimated the size of a phrase table extracted from the data used in our experiments (see §3.5, below). The data contains over 27 million words of Chinese in over one million sentences. Currently, this is among the largest in-domain training corpora for newswire translation tasks. From this it is easy to compute counts and sizes of all unique source phrases. We did not compute the size of a complete model based on arbitrary length phrases, for reasons that will become apparent. Instead, we estimate the size of such a phrase table using the following assumptions.

1. Each unique source phrase has exactly one translation.
2. All data types require four bytes of storage.
3. A node *sans* data requires twelve bytes of storage: four for a pointer to the node from its parent node, four for the edge label, and four for a pointer to the variable-length data contained in the node.[4]
4. Each source phrase translates to a target phrase of the same length. Therefore, the data stored at the node representing an m-length source phrase requires $4m+20$ bytes: $4m+4$ bytes for the null-terminated target phrase and 16 bytes for the four scores associated with the rule.

Assumption 1 is conservative, particularly for frequent short phrases which often have hundreds or thousands of translations. The other assumptions assume a compact implementation similar to one described by Zens and Ney (2007).[5] Actual phrase table sizes also depend on phrase extraction heuristics, which we don't address.[6] Nonetheless, our estimate seems reasonable for illustrative purposes. Figure 3.4 shows the number of unique m-length source phrases and

[4] We assume a integerized representation of words.

[5] In fact, its compactness entails a slight tradeoff in speed, since a binary search is needed to find an outgoing edge at each node.

[6] We will examine these heuristics empirically in §5.2.3.3.

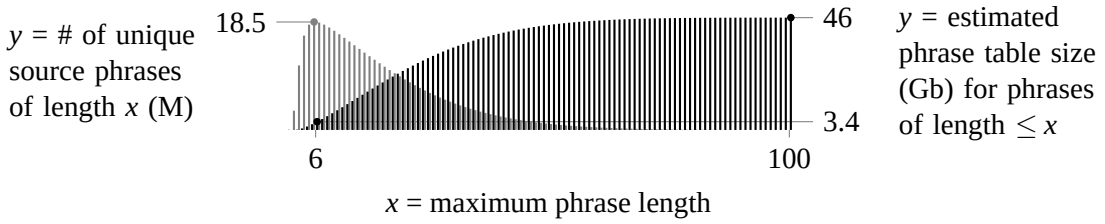


Figure 3.4: Number of unique source phrases of different lengths and their cumulative effect on estimated phrase table size.

their cumulative impact on phrase table size. We estimate that a complete model extracted from our data would require 46 gigabytes of space. This is large enough to be impractical for offline computation, let alone storage in main memory.[7]

Obviously, a tradeoff is required. We need to reduce the number of unique phrases in the model. To do this we either need to reduce the amount of training data, or reduce the number of phrases that are extracted from the training data.

Figure 3.4 suggests an easy implementation of the latter option. We can reduce the number of phrases by limiting the length of phrases rather than allowing any arbitrary-length substring to be a phrase. This is the prevailing strategy in most systems. Obviously the phrase length limit should be low, since the estimate shows that phrases of even a few words can consume gigabytes of storage. Limits from anywhere between two and seven words are typically used (Ayan and Dorr, 2006a; Koehn et al., 2003; Zens and Ney, 2007). As suggested by this range, there is some debate over the best cutoff, a matter which we will examine empirically in Chapter 5. From a practical perspective, it is acceptable to remove longer phrases from the model, since it is very unlikely that they will ever be encountered. Although it is rare for test sentences to match a long phrase, on those occasions we miss the opportunity to fully exploit the training data for what is likely to be a very good translation.

If our training data grows large enough, setting a maximum phrase length might not be enough to prevent our model from outgrowing available memory. Phrase table filtering is a popular solution to this. It is used in batch translation, a common scenario occurring in optimization or in translation of benchmark data such as those used in the NIST evaluations. After the model is computed, source phrases that don't appear in the test set are removed along with their translations and parameters, and only parameters needed to translate the test data are loaded into memory (Figure 3.5). Obviously, this method is limited to cases where we know the test data in advance, such as translation of benchmark data for evaluation purposes. However, it does allow the system to translate with a somewhat larger model than can reasonably be stored in main memory. To alleviate test set dependency, we can also filter on other criteria (Johnson et al., 2007).

Figure 3.5 illustrates an inefficiency of filtering. Although we never require the parameters of the complete model at runtime, our parameter estimation step still computes all of them. We spend additional time removing many of them from the model. For the large corpora used in contemporary systems, these steps take many hours.

Zens and Ney (2007) relax the dependence on main memory. They store their model in an

[7] A 46 gigabyte model might not seem unreasonable within a few years. However, we will show in Chapter 5 that a combination of larger training data and more complex models can generate representations that are at least three orders of magnitude larger than this. Considering the pace of corpus acquisition and model development, we don't expect hardware capacity to catch up with potential model sizes at any time in the foreseeable future.

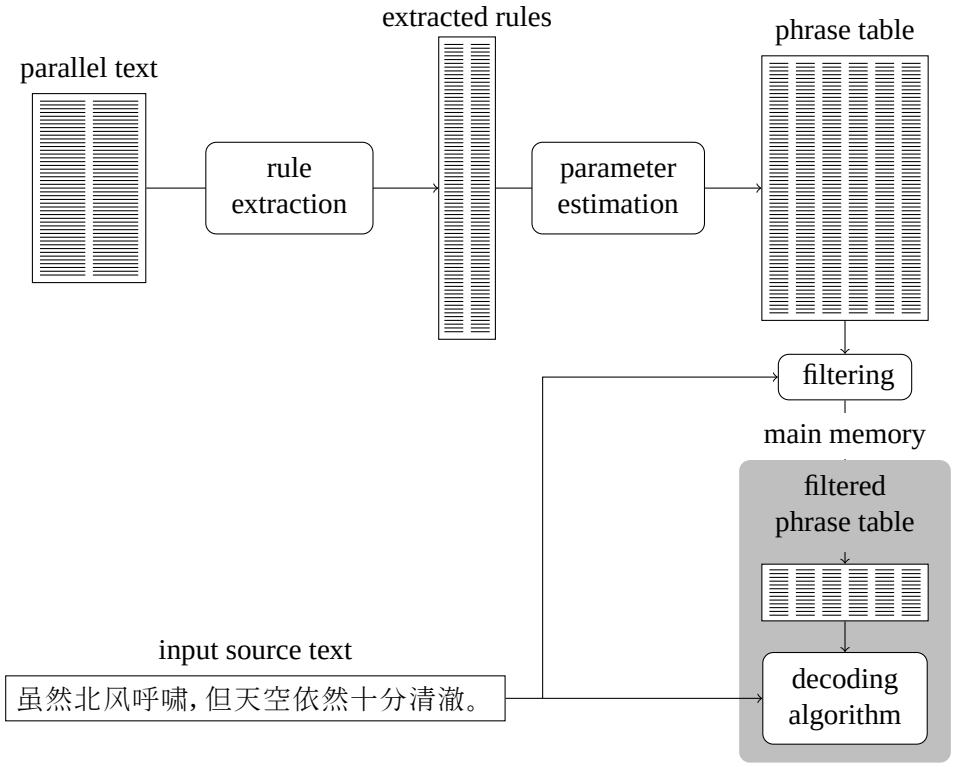


Figure 3.5: Architecture of a table-based decoder with filtering. Filtering is necessary when the model becomes to big to fit into memory (cf. Figure 3.3).

external prefix tree. Portions of the tree that are needed at runtime are paged in from disk. This allows the model to scale somewhat beyond the limits of memory. However, they still must make tradeoffs in order to compute their model offline. They impose a strict maximum phrase length.

We now describe an architecture that requires neither offline computation of a full model nor limitation to maximum phrase length.

3.3 Translation by Pattern Matching

An alternative to direct representation comes from work in *example-based translation*, sometimes called *memory-based translation* (Nagao, 1984; Sato and Nagao, 1990; Somers, 2003). Like statistical MT, example-based MT is a data-driven approach to translation. However, the two approaches draw on largely separate research traditions. As we saw in Chapter 2, a unifying principle in statistical MT is optimization. It draws largely on methods from machine learning. In contrast, example-based translation draws from a number of different disciplines. These include statistics, but example-based translation is typically not implemented as an exercise in optimization. Wu (2005) argues that its distinguishing characteristic is treatment of the training data as a runtime library and translation by analogy, making it similar to case-based reasoning. In fact, practitioners of example-based MT are not agreed on a precise characterization (Hutchins, 2005), and it is sometimes argued that it subsumes statistical MT (Somers, 2003).

Definitions aside, a clear characteristic of example-based translation is its view of the training corpus as a database of examples. To translate a sentence, the system searches for matching

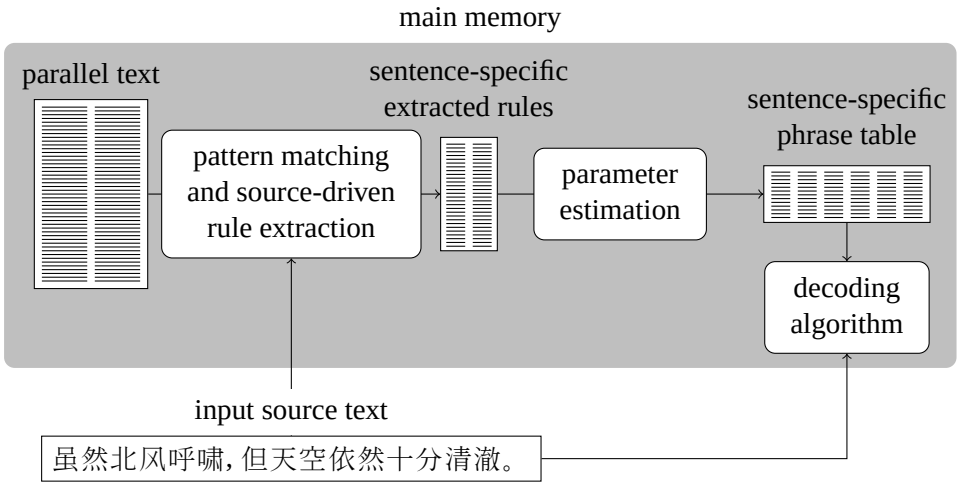


Figure 3.6: Translation by pattern matching. In this architecture, the complete model is no longer bound by what we can fit in main memory, or even what we can efficiently compute offline.

fragments of text in this database at runtime. Subsequent steps are system-specific. However, we are mainly interested in the idea of efficient search over a training corpus. For this, example-based systems use data structures for efficient efficient pattern matching (Brown, 2004).

Callison-Burch et al. (2005) and Zhang and Vogel (2005) independently applied pattern matching to phrase-based statistical MT. They employ a strategy in which the corpus itself is stored in main memory, just as in example-based translation. To decode a new sentence, the system employs pattern matching to search for each candidate source phrase in the source language corpus. Matching phrases are extracted along with their aligned target phrases and scored. The resulting sentence-specific phrase table is then used in the decoding algorithm, and subsequently freed from memory. In this indirect representation, the training data itself serves as a proxy for the complete model, and is queried for model parameters only as needed. We call this translation by pattern matching. It is illustrated in Figure 3.6.

Translation by pattern matching has several potential advantages.

- It is possible to scale to large corpora without the tradeoffs required by direct representation.
- There is no need to filter the model for a specific test set. This makes translation by pattern matching suitable for online settings where arbitrary input is expected.
- Where applicable, arbitrarily long phrases can be used, since we generate and discard phrase pairs as necessary and do not need to keep them in memory all at once.
- Because there is no need to extract, score, and store a complete model, it becomes much easier to experiment with grammar parameters and features. This is illustrated in Chapter 5.

These are all potentially valuable benefits. However, while Callison-Burch et al. (2005) and Zhang and Vogel (2005) lay the foundation for translation by pattern matching, they leave some unanswered questions.

- It is difficult to include the target-to-source translation feature (Equation 3.1). To compute the probability $p(\tilde{f}|\tilde{e})$ we need a count of all source phrases in the corpus that align to the

target phrase $\tilde{e}$. This implies that after searching for and extracting rules for a source phrase, we must then search for and extract rules for all of its possible target phrases. As we will see, the initial search and extraction for the source phrase is expensive. An additional search for target phrases would be onerous, so we sacrifice the feature for efficiency. However, as we saw in Chapter 2, this feature originated in the earliest Bayesian approaches to statistical MT (§2.4.1), and has since been considered indispensable. It is not clear how a system will fare without it.

- The models of Callison-Burch et al. (2005) and Zhang and Vogel (2005) were not optimized for translation accuracy using minimum error rate training (Och, 2003, see §2.5.3.1). In fact, as we will show, an optimization technique used by their approach has an undesirable interaction with the MERT algorithm (Och, 2003, §2.5.3.1) used for optimization. Therefore, paradoxically, although they can easily exploit very large corpora, neither paper reports state-of-the-art results on benchmark data.[8] It is therefore unclear whether translation by pattern matching is a viable approach for state-of-the-art results, or merely a clever algorithmic curiosity.

- The algorithms could only be applied to phrase-based models based on contiguous phrases. In Chapter 2, we described an increasingly complex progression of models. Several of these, such as the model of Chiang (2005, 2007), allow translation of discontiguous phrases. The pressures we described above for standard phrase-based models are much more acute for these models. However, the method of Callison-Burch et al. (2005) and Zhang and Vogel (2005) does not work for discontiguous phrases.

In this chapter and the next, we solve these problems. In order to gain a deeper understanding of them, we first review the suffix array data structure used to implement translation by pattern matching (§3.3.1). We also describe source-driven rule extraction (§3.3.2).

3.3.1 Pattern Matching and Suffix Arrays

A fundamental task in pattern matching is *exact pattern matching* on strings. We are given a *query pattern* w and a *text* T, and our goal is to find all occurrences of w in T. Exact pattern matching has a vast array of applications and many algorithms have been developed. Gusfield (1997) gives an excellent introduction, with a focus on algorithms used in biological sequence analysis.

In phrase-based translation, we are given an input sentence of length J. Each of its J^2 substrings is a possible source phrase. We need to query the source side of the training bitext for each of these substrings (Figure 3.7). If we have a fast enough exact pattern matching algorithm for single query patterns, we can implement our framework by enumerating all substrings of the input and searching the training text for each of them.

To implement fast lookup on the text, we use an indexing data structure called a *suffix array* (Manber and Myers, 1993). It represents all *suffixes* of the text in lexicographical order. Formally, the ith suffix of text T is the substring beginning at position i and continuing to the end of T

[8] Callison-Burch et al. (2005) evaluate on a unique test set, making their results difficult to situate in the literature. Zhang and Vogel (2005) evaluate on the NIST 2002 Chinese-English task achieving a BLEU score of 17.6. A comparable phrase-based system achieves a score of 34.9 on the same task (DeNeefe et al., 2007). The latter score is typical of the best models on this data. Under most circumstances, we caution that it is misleading to directly compare self-reported BLEU scores. Differences in implementation, tokenization, and capitalization can lead to differences of several BLEU points. However, due to the extreme difference in this case, we can confidently state these are extremely poor results.

Input Sentence: *it persuades him and it disheartens him*

Query Patterns: *it, persuades, him, and, disheartens, it persuades, persuades him, him and, and it, disheartens him, it persuades him, persuades him and, him and it, and it disheartens, it disheartens him, it persuades him and, persuades him and it, him and it disheartens, and it disheartens him, it persuades him and it, persuades him and it disheartens, him and it disheartens him, it persuades him and it disheartens, persuades him and it disheartens him, it persuades him and it disheartens him*

Figure 3.7: Example input sentence and resulting query patterns for phrase-based translation. For clarity, all of our pattern matching examples are in English, though in practice our source text is Chinese.

(Figure 3.8). This suffix is uniquely identified by the index i of its first word. The suffix array SA_T of T is a permutation on the set of suffix identifiers $[0, |T|-1]$ corresponding to the lexicographical order of the suffixes (Figure 3.9).[9]

Suffix arrays enable fast exact pattern matching. Every substring of T is the prefix of a suffix of T. Because SA_T represents the suffixes in lexicographical order, we can find all occurrences of a substring w with binary search. Every occurrence of w will correspond to exactly one suffix of T, and they will all be found within a contiguous range of SA_T. This range can be identified with a pair of binary searches on the suffix array. Specifically, a length-m substring can be found in $O(m+\log|T|)$ time (Manber and Myers, 1993).[10]

3.3.2 Source-Driven Phrase Extraction and Scoring

Once we have found the occurrences of a source phrase, we need to extract its translations. If we were computing a direct representation of the model, we would simply extract all viable phrase pairs from the sentence in which the source phrase occurs. However, since we only need the translation of the source phrase we are interested in, this is inefficient. Our goal is to extract only the translation of the specific source phrase that we have found. We call this *source-driven phrase extraction*.

Given a source phrase, its target phrase will be the minimal target span containing all words that are aligned to at least one of its words. To find this span, we simply find the minimal and maximal target word indices of all words aligned to any word in the source phrase.

We are not quite done. Recall that none of the words in a valid phrase pair can be aligned to words outside the pair (§2.5.1.2). In order to extract the phrase pair, we must check to see whether this condition is satisfied. To do this, we invert the previous step and find the minimal source span that is aligned to the target span. If this source span does not match the original source phrase, then the target phrase is aligned to words outside of the source phrase and extraction fails. Otherwise, we extract the target phrase. Under a *loose* heuristic (Ayan and Dorr, 2006a), we can also extract target phrases containing any unaligned target words immediately adjacent to the target span. Phrase extraction is illustrated in Figure 3.10.

[9] Actually, any total ordering on the tokens can be used. In our implementation we use the natural order on a integerized representation of words.

[10] This result requires a bit of algorithmic subtlety that we ignore here. In fact, this simple algorithm is not the most efficient solution. Abouelhoda et al. (2004) show that lookup can be done in optimal $O(m)$ time using some auxiliary data structures. However, for our purposes $O(m+\log|T|)$ is reasonable. The latter term, which we can think of as a corpus-specific constant, is fairly mild.

it	makes	him	and	it	mars	him	,	it	sets	him	on	and	it	takes	him	off	.	#
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

0 it makes him and it mars him , it sets him on and it takes him off . #
1 makes him and it mars him , it sets him on and it takes him off . #
2 him and it mars him , it sets him on and it takes him off . #
3 and it mars him , it sets him on and it takes him off . #
4 it mars him , it sets him on and it takes him off . #
5 mars him , it sets him on and it takes him off . #
6 him , it sets him on and it takes him off . #
7 , it sets him on and it takes him off . #
8 it sets him on and it takes him off . #
9 sets him on and it takes him off . #
10 him on and it takes him off . #
11 on and it takes him off . #
12 and it takes him off . #
13 it takes him off . #
14 takes him off . #
15 him off . #
16 off . #
17 . #
18 #

Figure 3.8: Example of a text and its set of suffixes. Note that each suffix can be uniquely identified by its starting position in the text. In keeping with a common convention of the pattern matching literature, the text ends with a special symbol (#) that is distinct from every other symbol in the alphabet.

it	makes	him	and	it	mars	him	,	it	sets	him	on	and	it	takes	him	off	.	#
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

3	and it mars him , it sets him on and it takes him off . #
12	and it takes him off . #
2	him and it mars him , it sets him on and it takes him off . #
15	him off . #
10	him on and it takes him off . #
6	him , it sets him on and it takes him off . #
0	it makes him and it mars him , it sets him on and it takes him off . #
4	it mars him , it sets him on and it takes him off . #
8	it sets him on and it takes him off . #
13	it takes him off . #
1	makes him and it mars him , it sets him on and it takes him off . #
5	mars him , it sets him on and it takes him off . #
16	off . #
11	on and it takes him off . #
9	sets him on and it takes him off . #
14	takes him off . #
7	, it sets him on and it takes him off . #
17	. #
18	#

Figure 3.9: Suffix array for the example text of Figure 3.8. We also show the result of a query for the pattern *him*. Note that each occurrence is the prefix of a suffix of the corpus, that there is a one-to-one correspondence between the occurrences and the suffixes, and that all of the suffixes occur in a contiguous stretch of the array, meaning that we can find them using binary search.

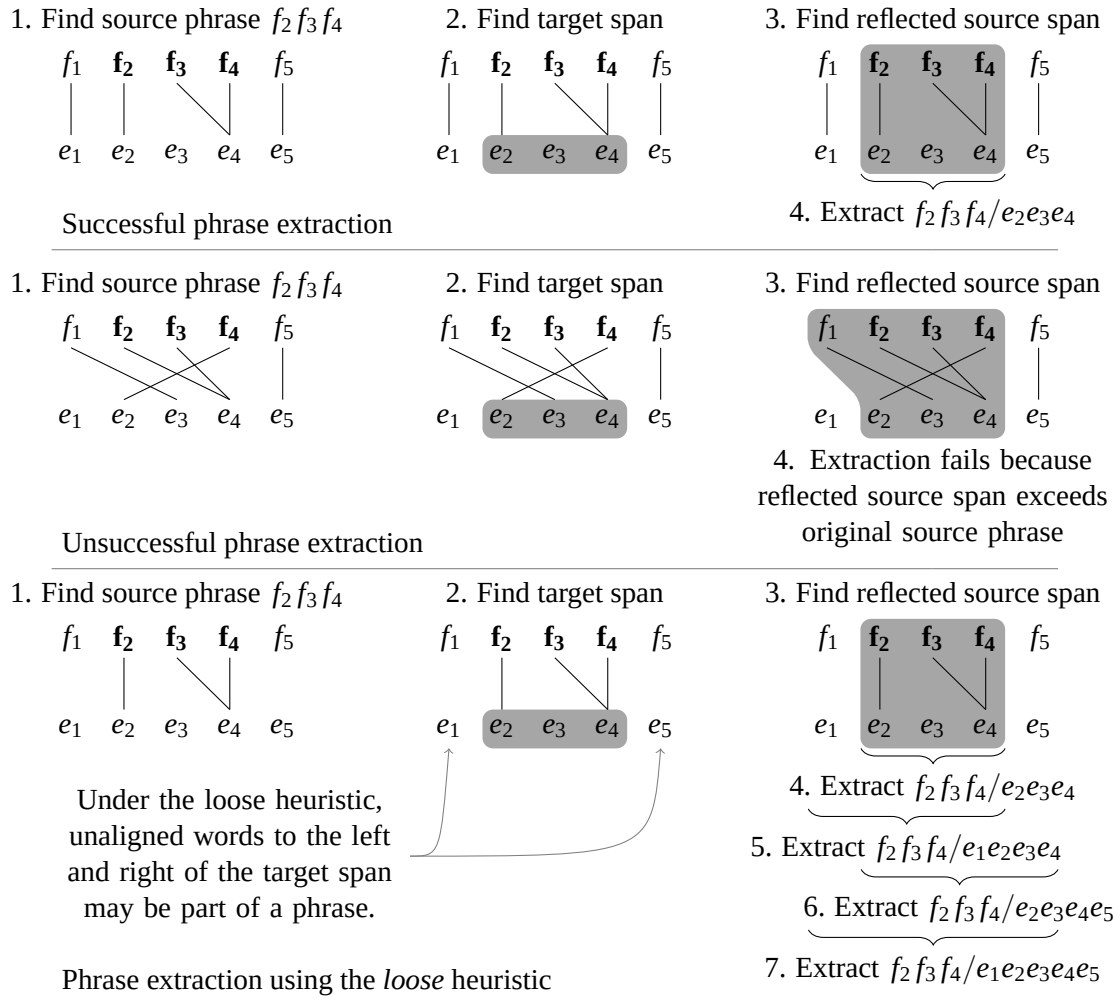


Figure 3.10: Examples of source-driven phrase extraction.

Once we have collected all of the target phrases that are aligned to a source phrase, we can compute the source-to-target translation probabilities and lexical weightings. For the latter we rely on a precomputed table of word-to-word translation probabilities computed from the word-level alignment.

The complexity of extraction and scoring is linear in the number of occurrences of a source phrase. In Figure 3.11, we see that the vast majority of source phrases occur only a handful of times in our corpus. However, a small handful of source phrases occur hundreds of thousands of times in our corpus. Extracting all of these examples would be extremely expensive. Callison-Burch et al. (2005) and Zhang and Vogel (2005) counteract this problem with sampling. Rather than extracting a translation for every occurrence of a source phrase, they place a cap on the number of of examples. Both groups arrived at a sample size of 100 via experimentation.[11] It is unclear how sampling interacts with the minimum error rate training algorithm. We discuss this

[11]This sample is obviously small for the handful of phrases occurring tens or hundreds of thousands of times. It is perhaps surprising that this should work as well as the full data. However, Och (2005) and Federico and Bertoldi (2006) show that phrase translation probabilities can be stored in four bits without loss of precision, meaning that they are in fact very crude even when computed from complete data.

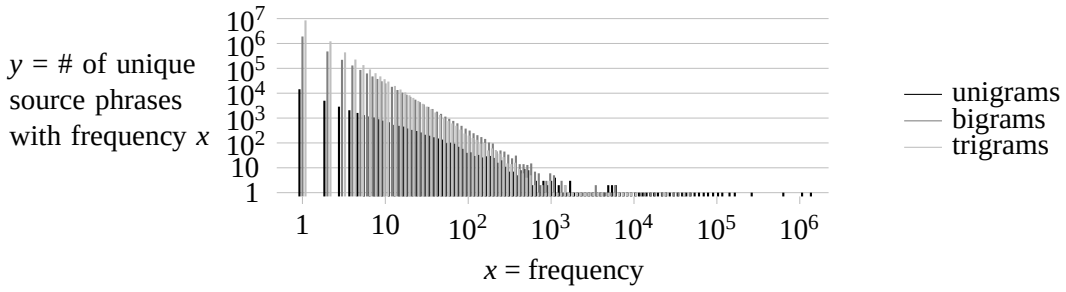


Figure 3.11: Histogram of source phrase frequencies, up to length three (double logscale).

in more detail below.

3.4 Bringing Performance up to the State of the Art

We need to address two questions. The first is the importance of the target-to-source phrase translation feature. This feature is often assumed to be important to the success of phrase-based-systems. However, a feature selection experiment by Lopez and Resnik (2006) suggests that several of the standard features were less important than previously thought. In particular, source-to-target feature and target-to-source features appeared to be redundant. Since our system can compute the former feature, we hope that it will not need the latter. This question is answered empirically in §3.5.

A second concern is the application of minimum error rate training (MERT, Och, 2003, §2.5.3.1). Neither Callison-Burch et al. (2005) nor Zhang and Vogel (2005) applied it to their models.

A standard approach to sampling in statistical models, *random sampling*, interacts with the MERT algorithm. Recall that the MERT algorithm works by iteratively collecting *n*-best hypotheses and their feature values. These are used to compute an approximation to the error surface. A problem with random sampling is that it causes the feature values for a hypothesis to vary between different runs of the system. This is especially problematic if the system produces the same hypothesis with different features during different iterations of the algorithm. In this case, the algorithm views these as separate hypotheses. We found that, under this strict interpretation, the algorithm would never converge, because it would never meet the convergence criterion that no new hypotheses be added in an iteration. To solve this problem, we modified the algorithm so that a hypothesis was not considered new if it had been seen before with different weights. However, this leads to a new problem: which set of weights should we choose for the hypothesis? We decided to take the first set of weights that had been seen with the hypothesis. Using this definition, the algorithm eventually terminates. However, on average it took twice as many iterations as it did for a standard decoder.

To resolve this issue, we used deterministic sampling. Whenever a source phrase occurs more frequently than the maximum sample size, we take our samples at uniform intervals over the set of locations returned by the suffix array. With this strategy in place, hypotheses receive the same feature weights between different runs of the decoder, the results are deterministic, and the MERT algorithm converges at the same rate as it does without sampling.

Configuration	BLEU
baseline with standard eight features	28.6
baseline without target-to-source translation feature	28.3
baseline without target-to-source translation or phrase count features	28.2
baseline without phrase count feature	28.1

Table 3.1: Baseline system results compared with systems missing one or more features.

3.5 Results

We experimented on Chinese to English translation in the newswire domain. Our training data consisted of over 1 million sentences compiled from various corpora provided by the Linguistic Data Consortium. The corpus is roughly the same as the one used for large-scale experiments by Chiang et al. (2005). To generate alignments, we used GIZA++ (Och and Ney, 2003). We symmetrized bidirectional alignments using the grow-diag-final-and heuristic (Koehn et al., 2003). Each configuration of the system was separately optimized on the NIST 2003 Chinese-English test set (919 sentences) using minimum error rate training (Och, 2003, §2.5.3.1). We measure translation accuracy using the NIST implementation of case-insensitive BLEU.[12] We test on the NIST 2005 Chinese-English test set (1082 sentences).

For our algorithms, we also measure the computational overhead required to search for, extract, and score rules. We report the average time required per sentence on the NIST 2003 data. All experiments were performed on identical time-shared cluster machines with 8 gigabytes of memory and two dual-core 3GHz Xeon processors running Red Hat linux 2.6.9. To minimize discrepancies caused by CPU load, we obtained exclusive use of the machines during timing runs.

Our decoder is Pyro, a clone of the Pharaoh decoder written by David Chiang in the interpreted language Python. We implemented the suffix array extensions in Pyrex, a language for writing compiled C extensions to Python. For speed, we compile our suffix array and other data structures offline into memory-mapped files, which are then read at decoder initialization. This takes only a few seconds, so the amortized cost over our data is negligible.[13]

3.5.1 Baseline System Results

Our first experiment measures the impact of losing the target-to-source translation feature (Equation 3.1). We did this using a standard direct representation of the phrase table using prefix trees, with a phrase length limit of four.

We noticed during development that the phrase count feature seemed to be minimally important. Therefore, we ran the experiments without this feature as well. This can be thought of as a kind of manual model selection. The results are shown in Table 3.1. We see a slight drop in accuracy when we lose the target-to-source translation feature, but it is not statistically significant. This indicates that removing the feature is not harmful to translation accuracy.

3.5.2 Translation by Pattern Matching Results

Our next experiment was designed to see if translation by pattern matching was a viable replacement for phrase tables. In order to make the comparison as fair as possible, we enforced

[12] ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v11b.pl

[13] In fact, this is the same approach taken by Zens and Ney (2007) for their direct representation.

Sample size	% sampled	time (s)	BLEU
Baseline	–	–	**28.6**
0	–	0.0094	–
10	75	0.0543	25.0
25	67	0.0910	26.4
50	62	0.1447	27.6
100	56	0.2198	27.7
200	51	0.3476	28.0
300	48	0.5216	28.4
400	**45**	**0.5557**	**28.6**
500	44	0.6492	28.5
800	40	0.9139	28.4

Table 3.2: Effect of different sample sizes on translation speed and translation accuracy. We show in column two the percentage of the ruleset that was computed by sampling. We include a sample size of zero to show the time required for lookup without phrase extraction or scoring. For comparison, we also show the baseline system using prefix trees and the full feature set from Table 3.1.

the same length restriction on phrases as in the baseline model (Experiments with longer phrases are in Chapter 5). Therefore, the translation model is nearly the same as in the baseline system. The only differences are in the missing target-to-source feature and the fact that the source-to-target feature is computed by sampling.

To measure the speed/accuracy tradeoff, we ran the system using several different sample sizes. We limited the maximum sample size to 800, because larger sizes would have been prohibitively slow. For each sample size we were curious about what fraction of the phrase table was computed via sampling. We include the percentage of rules computed by sampling out of the total number of rules computed. These statistics were measured on the development set.

Results are given in Table 3.2. Several conclusions are evident. The first is that translation by pattern matching is viable as a replacement for phrase tables. Neither Callison-Burch et al. (2005) and Zhang and Vogel (2005) matched state-of-the-art performance on a standard benchmark, but our careful consideration of sampling and minimum error rate training make this possible. Although the results for the table-based system are slightly higher, the difference is not statistically significant. A second result is that a surprisingly large fraction of the model is computed by sampling, well over half even for sample sizes giving the best performance. Finally, we see that accuracy plateaus once the sample size reaches about 300. This essentially confirms the results of Callison-Burch et al. (2005) and Zhang and Vogel (2005), although our best sample size is slightly larger than their suggested value of 100, which did not fare quite as well.

3.5.3 Analysis of Memory Use

Our implementation maps each source and target word to a unique 32-bit integer. To implement the algorithms, we require several memory-resident data structures.

- The source text F is an array of integers. Its length is dependent on the number of tokens in the source text. We also include special tokens representing both the end of sentence and the end of the text.

- The target text E is represented in the same way as the source text.
- The suffix array SA_F is an array of integers. Its length is identical to the length of the target text.
- The alignment is an array of integers. Its length is identical to the number of alignment links.
- We keep an array of target sentence numbers, allowing us to map from tokens to sentence number in constant time.
- We keep a compact array of word-to-word translation probabilities in order to compute lexical weighting scores.

These data structures are quite compact. On our data, they require a little less than 650 megabytes. In contrast a phrase table may require several gigabytes. We illustrated this using estimates in §3.2, and we will describe a real example of a very large phrase table in §5.3.

3.6 Conclusions

We have introduced translation by pattern matching, an algorithmic solution to the problem of translation model scaling. We have surveyed past work, identified shortcomings, and overcome them to produce a system that reproduces state-of-the-art performance in phrase-based translation. This exercise is a warm-up for more complex models that improve on the standard phrase-based model. In the next chapter, we will show how translation by pattern matching can be applied to these models.

4 Pattern Matching for Phrases with Gaps

> *People who analyze algorithms have double happiness. First of all they experience the sheer beauty of elegant mathematical patterns that surround elegant computational procedures. Then they receive a practical payoff when their theories make it possible to get other jobs done more quickly and more economically.*
>
> –Donald Knuth

Phrase-based translation is an important milestone in statistical machine translation, but as we saw in Chapter 2, it is far from the final word in translation modeling. In the last few years, statistical MT models have greatly diversified. Most new models are inspired in some way by phrase-based translation, and motivated by a desire to overcome its weaknesses. Some notable examples include non-contiguous phrase-based models (Simard et al., 2005), hierarchical phrase-based models (Chiang, 2005, 2007, §2.3.2.3), dependency treelet models (Quirk and Menezes, 2006; Quirk et al., 2005), and syntax-based tree-to-string transducer models (DeNeefe et al., 2007; Galley et al., 2004, 2006).

Given the heterogeneity of these models, it is notable that they all share three specific characteristics. First, like phrase-based models, they can translate multi-word units. Second, unlike phrase-based models, they can also translate phrases with gaps—that is, multi-word units composed of words that are not contiguous in the source sentence. Finally, as a consequence of their greater expressivity, they all require many more rules than standard phrase-based models. In general, the ruleset extracted from a corpus by any of them is at least an order of magnitude larger than the ruleset of a phrase-based model extracted from the same data. In fact, the vast size of extracted rulesets is a recurring topic in the literature of these models (see, e.g. Chiang, 2007; DeNeefe et al., 2007; Simard et al., 2005).

The size of these rulesets makes efficient scaling techniques even more relevant to these models than it was to standard phrase-based models. However, the introduction of gaps poses an algorithmic challenge for translation by pattern matching. The pattern matching algorithm presented in Chapter 3 depended crucially on the fact that our query pattern was a contiguous string. If we no longer enforce contiguity, we require new algorithms for pattern matching. To the extent that phrases with gaps represent the future of statistical machine translation, the relevance of translation by pattern matching depends on its applicability to these models. We therefore seek to develop efficient pattern matching algorithms for models where source phrases contain gaps.

To make matters concrete, we will focus on hierarchical phrase-based translation (Chiang, 2005, 2007, §2.3.2.3). This model gives statistically significant improvements in BLEU score over a standard phrase-based system trained on the same data. Although we consider this specific model, we emphasize that our pattern matching algorithms are general enough to be applied to any of the aforementioned models, with the proviso that the source-driven rule extraction algorithm is model-specific and must be redeveloped for each case.

With this mind, we can now succinctly state the problem of this chapter: *Given an input sentence, efficiently find and extract all hierarchical phrase-based translation rules for that sentence in the training corpus.*

We first review the relevant aspects of hierarchical phrase-based translation (§4.1). We show that the obvious solution using state-of-the-art pattern matching algorithms is hopelessly inefficient (§4.2). We then describe a series of algorithms to address this inefficiency (§4.3). Our algorithms reduce computation time by two orders of magnitude, making the approach feasible and enabling us to replicate state-of-the-art translation accuracy (§4.5).

4.1 Hierarchical Phrase-Based Translation (Redux)

Hierarchical phrase-based translation is based on synchronous context-free grammar (§2.3.2.3). The lexicalized translation rules of this grammar may contain a single nonterminal symbol, denoted X. We will use a, b, c and d to denote terminal symbols, and u, v, and w to denote (possibly empty) sequences of these terminals. We will additionally use α and β to denote (possibly empty) sequences containing both terminals and nonterminals. A translation rule is written $X \rightarrow \alpha/\beta$. This rule states that a span of the input matching α is replaced by β in translation. We require that α and β contain an equal number (possibly zero) of coindexed nonterminals. An example rule with coindexes is $X \rightarrow uX_{\boxed{1}}vX_{\boxed{2}}w/u'X_{\boxed{2}}v'X_{\boxed{1}}w'$. When discussing only the source side of such rules, we will leave out the coindexes. For instance, the source side of the above rule will be written $uXvXw$.[1]

The pattern matching problem for this model is illustrated in Figure 4.1 (cf. Figure 3.7). If arbitrary sequences of terminals and nonterminals may be rules, then the number of source phrases that cover a sentence is exponential in sentence length. This is especially problematic for training the model. Chiang (2007) employs several heuristics to limit the size of the extracted grammar.

- The span of any extracted rule in either the source or target text is restricted to some small value (henceforth *MaxPhraseSpan*).
- The number of nonterminal symbols in a rule is restricted to some small value (henceforth *MaxNonterminals*).
- The total number of terminal and nonterminal symbols in the source side of a rule is restricted to some small value (henceforth *MaxPhraseLength*).[2]

Our algorithms are parameterized for these constraints so they don't depend in any way on specific values for them. We explore this in greater detail in Chapter 5.

Abstractly, translation by pattern matching can be applied using the same generic algorithm that we used for the standard phrase-based model.

1. Enumerate all source phrases that are licensed by the model.
2. Query the source training text for each source phrase.
3. Extract and score the translations of each source phrase.
4. Decode using the scored translation rules.

Implementing these steps for the hierarchical phrase-based model requires new algorithms for pattern matching and phrase extraction.

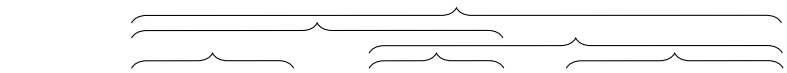
[1] In the canonical representation of the grammar, source-side coindexes always appear in numerical order, so source phrases are unambiguous despite this simplification.

[2] Chiang (2007) does not explicitly restrict the number of target-side symbols, making *MaxPhraseSpan* the *de facto* limit.

Input Sentence: *it persuades him and it disheartens him*

Query Patterns: *it, persuades, him, and, disheartens, it persuades, persuades him, him and, and it, it disheartens, disheartens him, it persuades him, persuades him and, him and it, and it disheartens, it disheartens him, it persuades him and, persuades him and it, him and it disheartens, and it disheartens him, it persuades him and it, persuades him and it disheartens, him and it disheartens him, it persuades him and it disheartens, persuades him and it disheartens him, it persuades him and it disheartens him, it X him, it X and, it X it, it X disheartens, it X him, persuades X and, persuades X it, persuades X disheartens, persuades X him, him X it, him X disheartens, him X him, and X disheartens, and X him, it X him and, it X and it, it X it disheartens, it X disheartens him, it persuades X and, it persuades X it, it persuades X disheartens, it persuades X him, it X him X it, it X him X disheartens, it X him X him, it X and X disheartens, it X and X him, it X it X him, persuades X and it, persuades X it disheartens, persuades X disheartens him, persuades him X it, persuades him X disheartens, persuades him X him, persuades X and X disheartens, persuades X and him, persuades X it X him, him X it disheartens, him X disheartens him, him and X disheartens, him and X him, him X it X him, it X him and it, it X and it disheartens, it X it disheartens him, it persuades X it disheartens, it persuades X disheartens him, it persuades him X disheartens, it persuades him X him, it X him X it disheartens, it X him X disheartens him, it X him and X disheartens, it X him and X him, it X and it X him, it persuades X and X disheartens, it persuades X and X him, it persuades X it X him, it X him X it X him, persuades X and it disheartens, persuades X it disheartens him, persuades him X it disheartens, persuades him X disheartens him, persuades him and X disheartens, persuades him and X him, him X it disheartens him, him and X disheartens him, him and it X him, it X him and it disheartens, it X and it disheartens him, it persuades X and it disheartens, it persuades X it disheartens him, it persuades him X it disheartens, it persuades him X disheartens him, it persuades him and X disheartens, it persuades him and X him, it X him X it disheartens him, it X him and X disheartens him, it X him and it X him, it persuades X and X disheartens him, it persuades X and it X him, it persuades him X it X him, persuades X and it disheartens him, persuades him X it disheartens him, persuades him and X disheartens him, persuades him and it X him, it X him and it disheartens him, it persuades X and it disheartens him, it persuades him X it disheartens him, it persuades him and X disheartens him, it persuades him and it X him, .*

Figure 4.1: Example input sentence and resulting query patterns for hierarchical phrase-based translation. There are many more query patterns than for a standard phrase-based system on the same sentence (cf. Figure 3.7).



it	makes	him	and	it	mars	him	,	it	sets	him	on	and	it	takes	him	off	.	#
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

3	and it mars him , it sets him on and it takes him off . #
12	and it takes him off . #
2	him and it mars him , it sets him on and it takes him off . #
15	him off . #
10	him on and it takes him off . #
6	him , it sets him on and it takes him off . #
0	it makes him and it mars him , it sets him on and it takes him off . #
4	it mars him , it sets him on and it takes him off . #
⋮	...

Figure 4.2: Matches in a suffix array fragment for the discontiguous query pattern *him X it*. For discontiguous patterns, there is no guarantee of a one-to-one correspondence between occurrences of the query pattern and suffixes in the same range of the suffix array (cf. Figure 3.9).

4.2 The Pattern Matching Problem for Hierarchical Phrases

As with standard phrase-based models, we can search for a contiguous source phrase $\alpha = u$ using a suffix array (§3.3.1). However, source phrases in form $\alpha = uXv$ or $\alpha = uXvXw$ complicate matters. We say that the contiguous sequences u and v are *collocated* because in order to form a rule they must occur in the same sentence. However, they do not need to be adjacent. The nonterminal symbol X can match an arbitrary (non-empty) sequence of text. Binary search will not work for these patterns.

Consider a query pattern uXv. All instances of this pattern contain the prefix u. Therefore, they all occur in the range of the suffix array containing suffixes with the prefix u. However, unlike the case of contiguous query patterns, there is no guarantee of a one-to-one mapping between suffixes in this range and occurrences of the query pattern (Figure 4.2). First, it is possible that a single suffix prefixed by u contains multiple instances of the search pattern. For instance, the suffix $uavv\#$ matches the query pattern twice. In the first match X spans a. In the second X spans av. Second, it is possible that non-matching suffixes are interspersed with matching suffixes. Suppose that our text has suffixes $uav...\#$, $ub\#$, and $ucv...\#$. These suffixes are in lexicographical order, yet only the first and third suffix contain the query pattern.

We will need another algorithm to find the source rules containing at least one X surrounded by nonempty sequences of terminal symbols.

4.2.1 Baseline Algorithm

In the pattern-matching literature, words spanned by the nonterminal symbols of Chiang's grammar are called *don't cares* and a nonterminal symbol in a query pattern that matches a se-

quence of don't cares is called a *variable length gap*. The search problem for patterns containing these gaps is a variant on approximate pattern matching (Navarro, 2001), a fundamental algorithmic problem in string processing that is central to bioinformatics and information retrieval.

The best algorithm for pattern matching with variable-length gaps using a suffix array is a recent algorithm by Rahman et al. (2006, henceforth RILMS). It works on a pattern $\alpha = w_1 X w_2 X ... w_{K_\alpha}$ consisting of K_α contiguous subpatterns $w_1, w_2, ... w_{K_\alpha}$, each separated by a gap. We wish to find all occurrences of α in text T. The algorithm is straightforward. We first locate each contiguous subpattern w_k in the suffix array. This takes $O(|w_k| + \log|T|)$ time. The result of the query is a set M_{w_k} of indices at which w_k occurs in the source text. To find occurrences of $w_1 X w_2$, we search for all pairs $(m_1, m_2) \in M_{w_1} \times M_{w_2}$ such that m_1 and m_2 are in the same sentence and meet the phrase length restrictions. The result set $M_{w_1 X w_2}$ must be the complete list of locations for $w_1 X w_2$. We repeat the computation for all pairs $w_1 X ... X w_{k-1}$ and w_k.

Consider the pattern *him X it*. Lookup on the example suffix array (Figure 3.9) is illustrated in Figure 4.3.

1. Look up all occurrences of *him*. These are enumerated in the suffix array range $[2,5]$. The result is $M_{him} = \{2, 15, 10, 6\}$.

2. Look up all occurrences of *it*. These are enumerated in the suffix array range $[6,9]$. The result is $M_{it} = \{0, 4, 8, 13\}$.

3. Compare elements of the first set with elements of the second to find instances of the pattern. In this simplified example, there is only one sentence, so the result set is $M_{him\ X\ it} = \{(2,4), (2,8), (2,13), (6,8), (6,13), (10,13)\}$. With a maximum span of ten, the instance $(2,13)$ would not qualify as a match.

Note that the result is a set of tuples. The kth element of each tuple is an index matching some occurrence of the kth subpattern of the query pattern in the source text. The location of query pattern α with K_α subpatterns is therefore a K_α-tuple. This is necessary to distinguish between cases in which multiple matches share subpatterns. There are several examples of this in Figure 4.3, including the matches $(2,4)$, $(2,8)$, and $(2,13)$, which share a subpattern located at position 2. The list $M_{w_1 X ... X w_{K_\alpha}}$ of occurrences is a set of these K_α-tuples.

The comparison step of the algorithm (step 3) is difficult because the set of locations that we find in the suffix array is not in numeric order—it is in lexicographical order. Performing this step efficiently will be a key problem for our algorithms. A naïve implementation would simply compare all of the elements in each set, giving an overall lookup complexity of $O(\sum_{k=1}^{K_\alpha} [|w_k| + log|T|] + \prod_{k=1}^{K_\alpha} |M_{w_k}|)$ for a single pattern $\alpha = w_1 X ... X w_{K_\alpha}$. To perform the comparison efficiently, RILMS inserts the elements of M_{w_k} into an efficient data structure called a *stratified tree* (van Emde Boas et al., 1977).[3] This is a priority queue in which the operations INSERT and NEXT-ELEMENT require $O(\log\log|T|)$ time.[4] To find collocations, the algorithm runs the NEXT-ELEMENT query for each element of $M_{w_1 X ... X w_{k-1}}$. This step is iterated until it returns a value that is in a different sentence or outside the phrase length constraints. Therefore, the total running time for an algorithm to find all contiguous subpatterns and compute their collocations is $O(\sum_{k=1}^{K} [|w_k| + log|T| + |M_{w_k}| \log\log|T|])$.

We can improve on RILMS using a variation on the idea of hashing. We exploit the fact that our large text is actually a collection of relatively short sentences, and that collocated patterns

[3] Often known in the literature as a *van Emde Boas tree* or *van Emde Boas priority queue*.

[4] Note that the dependence is on the size of the text, not the number of elements of the set. $\log\log|T|$ is a very mild term— on our corpus of 27 million words (§3.5) it is five. We can think of it as a very small corpus-specific constant.

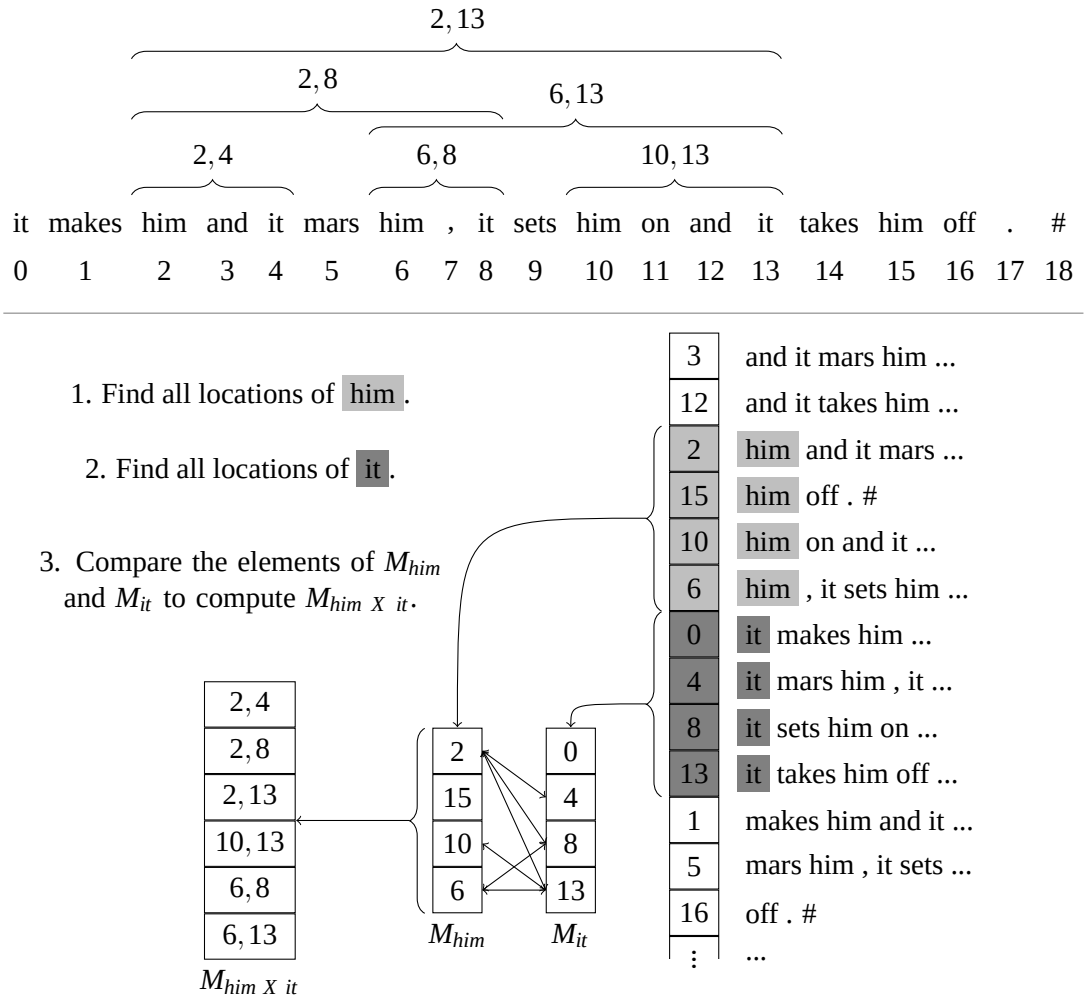


Figure 4.3: Illustration of baseline pattern matching algorithm for query pattern *him X it*.

must occur in the same sentence in order to be considered a rule. Therefore, we can use the sentence number (henceforth *SentenceNum*) of each subpattern occurrence as a kind of hash key.[5] We create a hash table whose size is equal to the number of sentences in our training corpus. Each location of the partially matched pattern $w_1X...Xw_k$ is inserted into the hash bucket with the matching sentence number. To find collocated patterns w_{k+1}, we probe the hash table with each of the $|M_{w_{k+1}}|$ locations for that subpattern. When we find a non-empty bucket, we compare the element with all elements in the bucket to find matches licensed by the phrase length constraints. Theoretically, the worst case for this algorithm occurs when all elements of both sets resolve to the same hash bucket, and we must compare all elements of one set with all elements of the other set. This leads to a worst case complexity of $O(\sum_{k=1}^{K_\alpha}[|w_k| + log|T|] + \prod_{k=1}^{K_\alpha}|M_{w_k}|)$. However, for real language data the average complexity will be much closer $O(\sum_{k=1}^{K}[|w_k| + log|T| + |M_{w_k}|])$, since on average any hash probes will return fewer than one match.

4.2.2 Analysis

It is instructive to compare the complexity of our baseline algorithm to the algorithm for the contiguous case. For a contiguous pattern w, the complexity of lookup is $O(|w| + log|T|)$. For a discontiguous pattern $\alpha = w_1Xw_2X...w_{K_\alpha}$, this complexity is $O(\sum_{k=1}^{K_\alpha}[|w_k| + log|T| + |M_{w_k}|])$. Note that the first two terms are analogous to the terms for the contiguous case. However, for discontiguous lookup the complexity includes the additional term $\sum_{k=1}^{K_\alpha}|M_{w_k}|$, which depends on the number of occurrences of each subpattern. This term dominates complexity if there is even one moderately frequent subpattern.

To make matters concrete, consider our 27 million word training corpus (§3.5). The three most frequent unigrams occur 1.48 million, 1.16 million and 688 thousand times—the first two occur on average more than once per sentence. In the worst case, looking up a contiguous phrase containing any number and combination of these unigrams requires no more than 25 comparison operations. In contrast, the worst case scenario for a pattern with a single gap, bookended on either side by the most frequent word, requires over thirteen million operations using RILMS and over two million using our improved baseline based on hashing. A single frequent unigram in an input sentence is enough to cause noticeable slowdowns, since it can appear as a subpattern of up to 84 hierarchical rules even using the tight grammar length restrictions of Chiang (2005, 2007), which we enumerate in §4.5.

This is not our sole worry. The full pattern matching algorithm must take into account all of the queries needed for a given model. As we've seen, the number of queries is quadratic in the case of a phrase-based model, and exponential in the case of a hierarchical phrase-based model with no length restrictions. Even with very tight length restrictions, the number of queries will be much higher than for the standard phrase-based system. This only compounds the problem of computationally expensive queries.

To analyze the cost empirically, we implemented an efficient version of our baseline algorithm as compiled C code using Pyrex and measured CPU time on the NIST 2003 test set under the grammar length restrictions of Chiang (2007, §4.5). The average per-sentence query time was 221.4 seconds (3.7 minutes), excluding extraction, scoring, and decoding. By comparison, per-sentence lookup time for the phrase-based model was 0.0094 seconds (§3.5)— four orders of magnitude faster.

[5] Our current implementation encodes the sentence number in a distinct length-$|T|$ array. This is an inefficient use of memory but enables constant-time access. We are currently investigating alternative encodings that use less memory while preserving constant-time access.

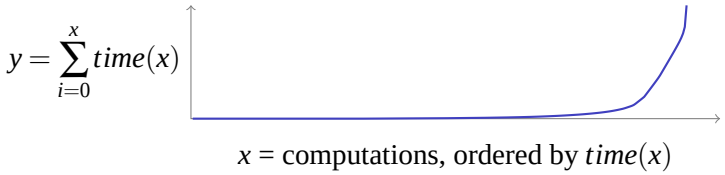


Figure 4.4: Cumulative time required for collocation computations.

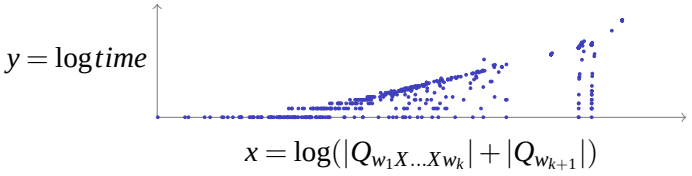


Figure 4.5: Size of input sets ($|M_1^k| + |M_{k+1}|$) compared with time required to compute collocation from the two sets using the baseline algorithm (double logscale).

4.3 Solving the Pattern Matching Problem for Hierarchical Phrases

Clearly, looking up patterns in this way is not practical. A detailed analysis confirmed the two predicted causes of computational expense.

1. The number of query patterns is large. With length restrictions, the hierarchical phrase-based model generates an average of 2825 query patterns per sentence. By comparison, a standard phrase-based model with an equivalent maximum phrase length (five) generates only 137 query patterns per sentence.

2. Cumulative lookup time was dominated by a very small fraction of the queries (Figure 4.4). As expected, further analysis showed that these expensive queries all involved at least one very frequent subpattern (Figure 4.5). In the worst cases a single pattern lookup required several tenths of a second.

Our solution addresses both of these problems. We introduce an algorithm for efficient enumeration that performs lossless pruning of unnecessary queries (§4.3.1). We also introduce several strategies to reduce the cost of individual queries (§4.3.2)

4.3.1 Efficient Enumeration

Although we found 2895 query patterns per sentence, the average sentence length is just over 29 words. Obviously, the query patterns are highly overlapping. We can exploit this to reduce computational expense.

4.3.1.1 The Zhang-Vogel Algorithm

Zhang and Vogel (2005) show an efficient algorithm for contiguous phrase searches in a suffix array. It is based on the observation that the prefix u of any possible source phrase ua is itself a possible source phrase. They exploit this fact to reduce the amount of work required

to search for *ua*. The set of suffixes with prefix *ua* is a subset of the set of suffixes with prefix *u*. Therefore, if we search for occurrences of *u* before searching for occurrences of *ua*, we can restrict the binary search for *ua* to the suffix array range containing suffixes prefixed by *u*. If there are no matches for *u*, we don't need to search for *ua* at all. This optimization improves efficiency for phrase search, although the improvement is modest since search for contiguous phrases is already very fast (§3.5).[6] However, the opportunity for improvement in discontiguous search is much greater.

Extension to hierarchical phrases is straightforward. A hierarchical phrase αa can only occur in T if its prefix α occurs in T. The actual pattern matching algorithm for hierarchical phrases is not as simple as binary search in a suffix array, so this doesn't enable an obvious search improvement as it does for contiguous phrases. However, it does allow us to rule out the existence of αa if the search for α fails. This prunes out many searches that are guaranteed to be fruitless.

4.3.1.2 Prefix Trees and Suffix Links

The Zhang-Vogel optimization is closely related to prefix trees. Recall that the prefix tree implementation of a phrase table encodes all legal source phrases in an unminimized finite state automaton (§3.2). We store target phrases and scores at the node associated with a source phrase. Representing hierarchical rules in the prefix tree requires no special modification. Since our nonterminal and terminal alphabets are mutually exclusive, we simply treat the nonterminal X as any other edge label.[7]

We implement the Zhang-Vogel algorithm using a prefix tree, which we construct for each source sentence. In fact, we can think of the pattern matching operation itself as an augmentation of edge traversal in a prefix tree. Suppose that we are at a node representing source phrase α, and we want to find translation rules for source phrase αa. If the node representing α does not have an outgoing *a*-edge, we first query the source text for phrase αa. If the query succeeds, we add an *a*-edge to a new node containing the newly extracted phrase pairs and scores. If the search fails, we still add the *a*-edge, but we mark the new node as *inactive*, indicating that the text contains no phrases with this prefix. In this way, the algorithm builds a prefix tree representing every query pattern licensed by our grammar for the input sentence, except patterns whose prefixes were not present in the training data (Figure 4.6). Now suppose that a source phrase occurs multiple times in the sentence (this frequently happens with determiners for example). The first occurrence is treated as described above, but for all subsequent occurrences we simply traverse the existing edge. We check the flag on the node to see if it is active or inactive, telling us whether the previous search was successful. Recall that the prefix tree is sentence-specific. We discard it and build a new tree for each sentence, enabling our decoder to run indefinitely without exhausting main memory.

We can improve on the Zhang-Vogel algorithm. The existence of phrase $a\alpha b$ in a text guarantees more than the existence of its prefix $a\alpha$. It also guarantees the existence of its suffix αb. If αb doesn't exist, we don't need search for $a\alpha b$ at all. Furthermore, note that unless $\alpha = X$, $a\alpha$ and αb must share at least one word. This means we can reduce the search to cases where $a\alpha$ and αb overlap, and their starting indices will differ by exactly one. To see why this is useful, consider a phrase $abXcd$. In our baseline algorithm, we would search for ab and cd, and then

[6] In fact, the results reported in the previous chapter incorporate both this optimization and one we introduce in §4.3.1.2.

[7] Conveniently, the decoder used in our experiments (Chiang, 2007) already encodes its grammars in a prefix tree. The implementation is similar to one described by Klein and Manning (2001). We simply augment this representation with information needed by our algorithms.

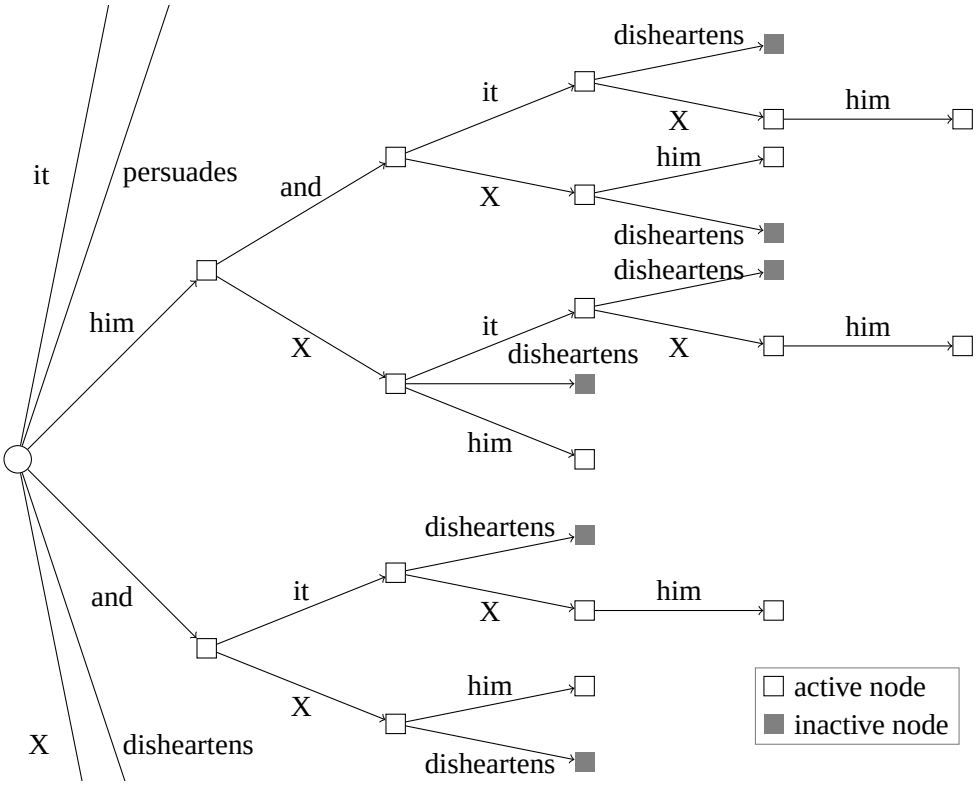


Figure 4.6: Portion of the prefix tree for tree for our example input (Figure 4.1).

perform a computation to see whether these subphrases were collocated within an elastic window. However, if we instead use the locations of *abXc* and *bXcd* as the basis of the computation, we gain two advantages. First, the number of elements in each set is likely to be smaller than in the former case. This is not guaranteed to be true, but it will be true in the vast majority of cases.[8] Second, the computation becomes simpler because there is no need to check whether the patterns cooccur within a variable-length window. It is sufficient to see whether they match exactly, except for the first index, which should differ by exactly one. For this to work, we need access to the set of locations for both the prefix and the suffix. Note that any pattern α can be the prefix or suffix of numerous source phrases. To facilitate reuse of the occurrence set M_α, we cache it at the corresponding prefix tree node. The full algorithm for computing occurrences $a\alpha b$ given all occurrences of $a\alpha$ and αb will be described in §4.3.1.5.

To access suffixes in constant time, we augment the prefix tree with *suffix links*. A suffix link is a pointer from a node representing $a\alpha b$ to a node representing its suffix αb (Figure 4.7). Via this connection, we can immediately check to see if the node has been marked inactive. If it is, we know that a query for the αb (or recursively, one of its suffixes) previously failed, and we don't need to query for $a\alpha b$ at all, since we know that it will not be found.[9]

As we described above, the prefix tree acts as a cache for phrases that occur multiple times in a source sentence. In these cases, we simply traverse the already-constructed edge, without needing to query the text for the phrase or extract its translations. Depending on our usage scenario and available memory, we could extend the caching behavior even further by retaining part or all of the tree between sentences instead of building a new tree for each sentence. For instance, a least recently used (LRU) strategy for cache pruning may improve translation speed for whole documents without exhausting memory. Though impractical for online settings, in batch translation we could simply keep the entire tree for the duration of the decoding process. This allows us to reuse queries that were already performed for previous sentences, potentially leading to further speedups.

4.3.1.3 Special Cases for Phrases with Gaps

The model permits gaps at the beginning or end of a hierarchical phrase. For instance, it permits source phrases Xu or uX or even XuX. However, even if our model disallowed such source phrases, they would still be prefixes or suffixes of valid sources sources, and therefore must appear in the prefix tree. Each of these phrases corresponds to a unique path in the prefix tree, although for pattern matching purposes they are all identical to phrase u. An analogous situation occurs with the patterns $XuXv$, $uXvX$, and uXv. There are two cases that we are concerned with.

Consider a pattern $\alpha = X\beta$. The path to its prefix tree node contains the X-edge originating at the root node. All paths containing this edge form a special subtree. Note that α occurs in the text at the same locations as β, although from the perspective of the translation model it is a different phrase. Therefore, we don't actually need to query the training text to find α. If the node representing β is active, we create an active node for α and set $M_\alpha = M_\beta$. If the node representing β is inactive, we create an inactive node for α.

[8] To see why this isn't always the case, consider an example. Suppose that our subpatterns are *ab* and *cd*, and the text contains the substring *ababdcdcd*. Although each subpattern occurs only twice, the combined pattern *abXcd* occurs four times.

[9] Note that the suffix of a one-symbol phrase is the empty string ε, represented by the root node of the prefix tree. The reader may notice that $a\alpha$ is undefined when the prefix pattern is ε. We define an auxiliary state $\perp$, following Ukkonen (1995). The suffix link from root points to $\perp$, while $\perp$ is connected to the root node by all symbols in the alphabet. Therefore, when $\alpha = \varepsilon$, $a\alpha = \perp$. Following the a-edge from $\perp$ returns us to the root. This simplifies much of the following algorithmic discussion without requiring the enumeration of several corner cases.

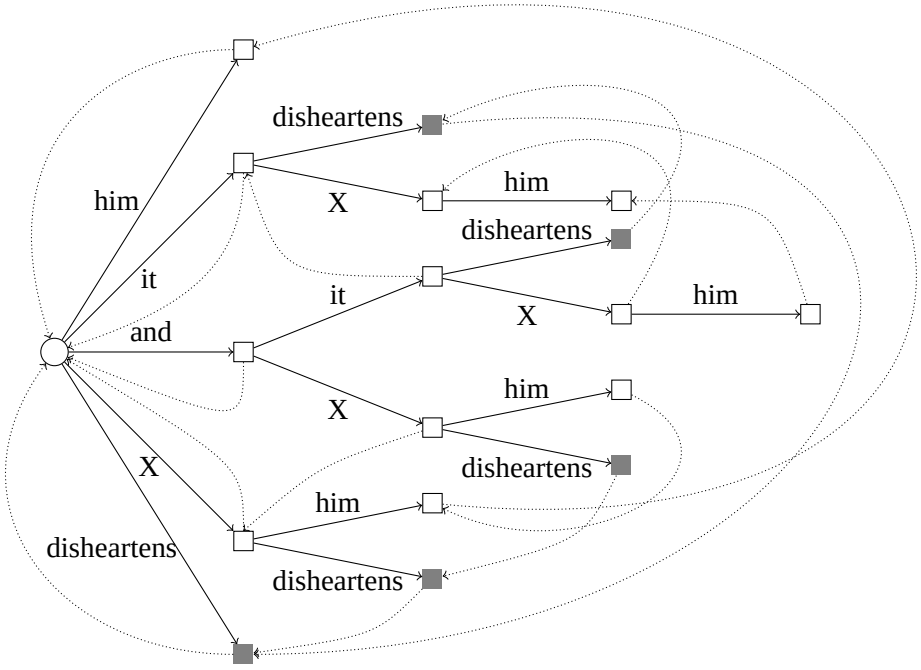


Figure 4.7: A fragment of the prefix tree (Figure 4.6) augmented with suffix links (dotted).

Now consider pattern $\alpha = \beta X$. For pattern matching purposes, this pattern is identical to its prefix β. Therefore, if we successfully find β, we automatically add an outgoing X-edge from its corresponding node, provided that βX is licensed by the length restrictions. Again we set $M_\alpha = M_\beta$.

Note that both special cases occur for a pattern in the form $\alpha = X\beta X$.

4.3.1.4 Putting It All Together: Prefix Tree Generation Algorithm

Given an input sentence, our algorithm (Listing 2) generates the corresponding prefix tree for its query patterns breadth-first. We maintain a queue of items, each consisting of a query pattern, its span in the input, and a pointer to the node corresponding to its prefix. The queue is initialized with all patterns containing a single terminal symbol. When we pop a pattern from the queue, there are two cases. If the corresponding edge exists in the prefix tree, we traverse it. Otherwise, we query the source text for the pattern. Irrespective of the query's success, we create a node for the pattern in the tree, and mark it active or inactive as appropriate. For found patterns, we cache either the endpoints of the suffix array range containing the phrase (if it is contiguous), or the full list of locations at which the phrase is found (if it is discontiguous).[10] We add to the queue any query patterns containing one more terminal for which the pattern is a prefix. This guarantees that all patterns containing m terminals are processed before any patterns containing $m+1$ terminals, which is sufficient to guarantee that any pattern is processed after both its suffix and prefix.

To query the text for pattern $a\alpha b$, we call function QUERY (§4.3.1.5), providing the sets of prefix and suffix locations $M_{a\alpha}$ and $M_{\alpha b}$ as parameters. $M_{a\alpha}$ is cached at the node corresponding to the prefix $a\alpha$, which is an element of the item popped from the queue. To find the node corresponding to the suffix αb, we first follow the suffix link from the node representing the prefix, $a\alpha$. This leads us to a node representing α. From this node we follow the b-edge, which leads to the node representing αb. This gives us constant-time access to both prefix and suffix information. If the suffix node is inactive, we can mark the new node inactive without a query. Some common cases are illustrated in Figure 4.6.

4.3.1.5 The Basic QUERY Algorithm

We need to define function QUERY called by the prefix tree algorithm (Listing 2). Its input consists of set $M_{a\alpha}$ representing all *matchings* of prefix $a\alpha$ and set $M_{\alpha b}$ representing all matchings of suffix αb. From this it must compute all matchings $M_{a\alpha b}$ of query pattern $a\alpha b$.

Recall from §4.2.1 that a matching in the text for pattern $\alpha = w_1 X \ldots X w_{K_\alpha}$ is a K_α-tuple $m_\alpha = (m_{\alpha,1}, \ldots, m_{\alpha,K_\alpha})$. The kth element $m_{\alpha,k}$ of m_α is the starting index of a substring in T matching subpattern w_k. If two patterns differ only by preceding or following nonterminal symbol X, then their matchings in T are identical. More formally, if $\alpha = \beta X$ or $\alpha = X\beta$ then $M_\alpha = M_\beta$. If $\alpha = X$ then $M_\alpha = ()$, the empty tuple.

For the discussion that follows, we will find it useful to define several formal properties of matchings. First, we define an ordering. For matchings m_α and m'_α of pattern α, $m_\alpha < m'_\alpha$ if and only if $\exists_{k \in \{1,\ldots,K_\alpha\}} (m_{\alpha,k} < m'_{\alpha,k}$ and $\forall_{\ell \in \{1,\ldots,k-1\}} m_{\alpha,\ell} = m'_{\alpha,\ell})$. Thus, a set of K_α-tuples is ordered on the first index, then the second, and so on.

The *suffix matching* $s(m_{a\alpha})$ of matching $m_{a\alpha}$ is the embedded matching of suffix α. If $\alpha = X\beta$ and $m_\alpha = s(m_{a\alpha})$ then $K_\alpha = K_{a\alpha} - 1$ and $m_\alpha = (m_{a\alpha,2}, \ldots, m_{a\alpha,K_{a\alpha}})$. If $\alpha = \beta c$ and $m_\alpha = s(m_{a\alpha})$ then $K_{\alpha,1} = K_{a\alpha,1} + 1$, $m_{\alpha,1} = m_{a\alpha,1} + 1$ and $\forall_{k \in \{2,\ldots,K_\alpha\}} m_{\alpha,k} = m_{a\alpha,k}$.

[10] As a practical matter, we can also store the scored translation rules after we extract them (§4.4).

Algorithm 2 Prefix Tree Lookup

Function GENERATE_PREFIX_TREE

Input: source sentence f_1^I, prefix tree root node p_ε

children(p_ε) $\leftarrow$ children(p_ε) $\cup\, p_X$
for i from 1 to I **do**
  Add $\langle f_i, i, i+1, p_\varepsilon \rangle$ to queue ▷ ⟨pattern, span start, span end, prefix node⟩
for i from 1 to I **do**
  Add $\langle X\, f_i, i-1, i+1, p_X \rangle$ to queue
while queue is not empty **do**
  Pop $\langle \alpha, i, j, p_{a\beta} \rangle$ from queue
  if $p_{a\beta f_j} \in$ children($p_{a\beta}$) **then**
    if $p_{a\beta f_j}$ is inactive **then**
      Continue to next item in queue
    else
      EXTEND_QUEUE($a\beta f_j, i, j, f_1^I$)
  else
    children($p_{a\beta}$) $\leftarrow$ children($p_{a\beta}$) $\cup\, p_{a\beta f_j}$
    $p_\beta \leftarrow$ *suffix_link*($p_{a\beta}$)
    if $p_{\beta f_j}$ is inactive **then**
      Mark $p_{a\beta f_j}$ inactive
    else
      $Q_{a\beta f_j} \leftarrow$ QUERY($a\beta f_j, Q_{a\beta}, Q_{\beta f_j}$)
      if $Q_{a\beta f_j} = \emptyset$ **then**
        Mark $p_{a\beta f_j}$ inactive
      else
        Mark $p_{a\beta f_j}$ active
        EXTEND_QUEUE($a\beta f_j, i, j, f_1^I$)

Function EXTEND_QUEUE

Input: $\alpha, i, j, f_1^I, p_\alpha$

if $|\alpha| <$ *MaxPhraseLength* and $j - i + 1 \leq$ *MaxPhraseSpan* **then**
  Add $\langle \alpha f_j, i, j+1, p_\alpha \rangle$ to queue
  if *arity*(α) $<$ *MaxNonterminals* **then**
    children(p_α) $\leftarrow$ children(p_α) $\cup\, p_{\alpha X}$
    Mark $p_{\alpha X}$ active
    $Q_{\alpha X} \leftarrow Q_\alpha$
    for k from $j+1$ to $min(I, i + MaxPhraseLength)$ **do**
      Add $\langle \alpha f_j X f_k, i, k, \alpha X \rangle$ to queue

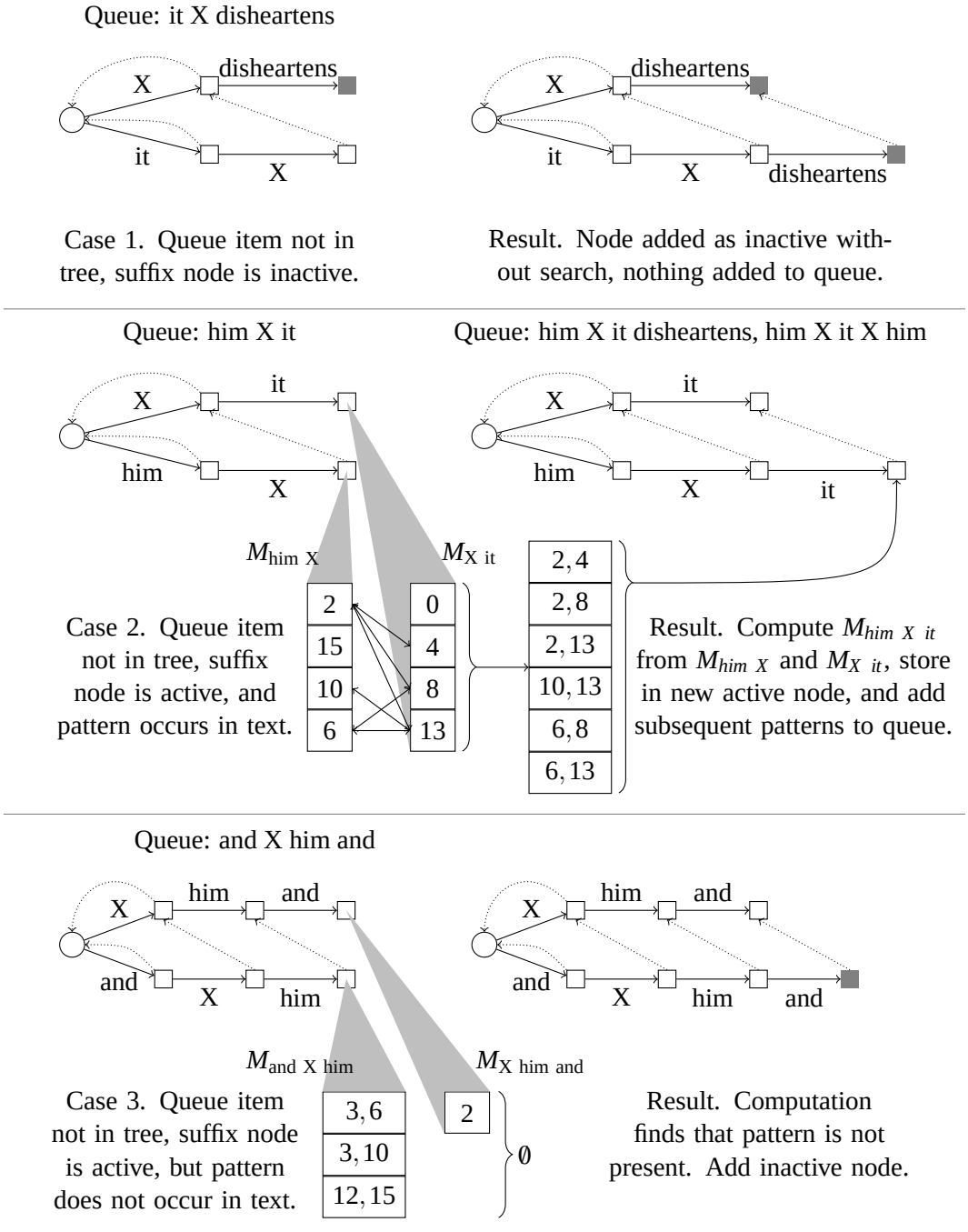


Figure 4.8: Illustration of several recursive cases in the prefix tree construction algorithm.

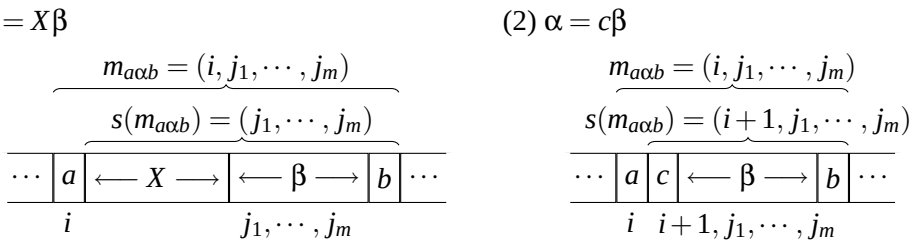


Figure 4.9: Relationship between a matching $m_{a\alpha b}$ of pattern $a\alpha b$ and its suffix matching $s(m_{a\alpha b})$ for two cases, depending on whether the first symbol in α is a nonterminal or terminal. Note that in both cases, there is only a single difference between the matchings. In case (1), there is an additional index. In case (2), the first index of each matching differs by exactly one. The situation for prefix matchings is analogous.

The relationship between a matching and its suffix matching are shown in Figure 4.9.

Analogously, the *prefix matching* $p(m_{\alpha b})$ of matching $m_{\alpha b}$ of pattern αb is the embedded matching of its prefix α. If $\alpha = \beta X$ and $m_\alpha = p(m_{\alpha b})$ then $K_\alpha = K_{\alpha b} - 1$ and $m_\alpha = (m_{\alpha b,1}, ..., m_{\alpha b,K_\alpha})$. If $\alpha = \beta c$ and $m_\alpha = p(m_{\alpha b})$ then $K_\alpha = K_{\alpha b}$ and $m_\alpha = m_{\alpha b}$.

Note that $p(s(m_{a\alpha b})) = s(p(m_{a\alpha b}))$. This gives a means to identify pairs $(m_{a\alpha}, m_{\alpha b})$ that imply a matching of $a\alpha b$. Specifically, if $s(m_{a\alpha}) = p(m_{\alpha b})$, then we say that $m_{a\alpha}$ and $m_{\alpha b}$ are *partners* and we can *join* ($\bowtie$) them to form $m_{a\alpha b}$. If $\alpha = \beta X$ and $m_{a\alpha b} = m_{a\alpha} \bowtie m_{\alpha b}$ then $K_{a\alpha b} = K_{a\alpha} + 1$, $\forall_{k \in \{1,...,K_{a\alpha}\}} m_{a\alpha b,k} = m_{a\alpha,k}$ and $m_{a\alpha b,K_{a\alpha b}} = m_{\alpha b,K_{\alpha b}}$. Otherwise, if $\alpha = \beta c$ then $K_{a\alpha b} = K_{a\alpha}$ and $m_{a\alpha b} = m_{a\alpha}$.

A special case occurs for query pattern aXb, since for any of its matchings m_{aXb}, $s(p(m_{aXb})) = p(s(m_{aXb})) = ()$. In this case we compute the pure collocation of subpatterns a and b. We must ensure that $m_{aX} \in M_{aX}$ and $m_{Xb} \in M_{Xb}$ occur in the same sentence and are separated by the minimum gap length. Note that we don't need to check these properties for longer patterns because our enumeration strategy ensures that they are satisfied by this recursive base case.

For all candidate partners we must check to ensure that the resultant matching does not exceed the maximum phrase span. With this and our other constraints in mind, we can now define special comparison relations on $M_{a\alpha} \times M_{\alpha b}$. To distinguish them from comparison on items drawn from the same set we use the decorated operators $\ddot{=}$ (partners), $\ddot{>}$ (precedes), and $\ddot{<}$ (follows). Suppose that we have query pattern $a\alpha b = w_1 X ... X w_{K_{a\alpha b}}$. Let $m_{a\alpha}$ be a matching of $a\alpha$ and $m_{\alpha b}$ be a matching of αb. There are two cases. First, suppose that the prefix and suffix do not overlap on any words—that is $\alpha = X$.

- If $SentenceNum(m_{a\alpha}) > SentenceNum(m_{\alpha b})$ then $m_{a\alpha} \ddot{>} m_{\alpha b}$.
- If $SentenceNum(m_{a\alpha}) < SentenceNum(m_{\alpha b})$ then $m_{a\alpha} \ddot{<} m_{\alpha b}$.
- If $SentenceNum(m_{a\alpha}) = SentenceNum(m_{\alpha b})$ then:
  - If $m_{a\alpha,1} \geq m_{\alpha b,1} - 1$ then $m_{a\alpha} \ddot{>} m_{\alpha b}$.
  - If $m_{a\alpha,1} \leq m_{\alpha b,1} - MaxPhraseSpan$ then $m_{a\alpha} \ddot{<} m_{\alpha b}$.
  - Otherwise $m_{a\alpha} \ddot{=} m_{\alpha b}$.

Next, suppose that the prefix and suffix overlap on some words—that is $\alpha \neq X$.

- If the prefix occurs after the suffix, $s(m_{a\alpha}) > p(m_{\alpha b})$, then $m_{a\alpha} \ddot{>} m_{\alpha b}$.

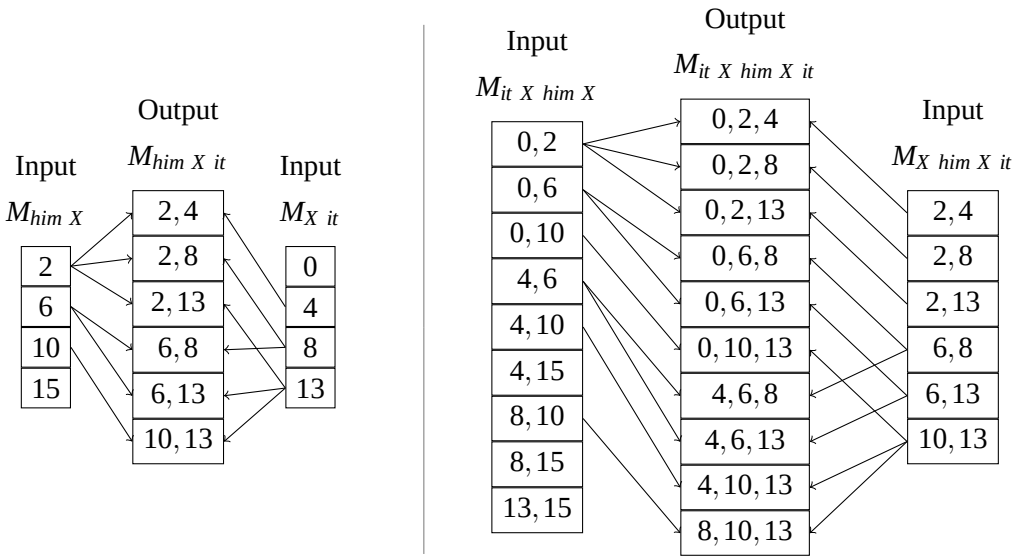


Figure 4.10: QUERY examples showing all input and output sets in sorted order and identifying the pair of input matchings that contribute to each output matching.

- If the prefix and suffix matchings overlap, $s(m_{a\alpha}) = p(m_{\alpha b})$, and the length of the combined pattern does not exceed the maximum phrase length, $m_{a\alpha,1} > m_{\alpha b,K_{\alpha b}} + |w_{K_{a\alpha b}}| - MaxPhraseSpan$, then we have found a match, $m_{a\alpha b} \doteq m_\alpha$.
- If the prefix occurs before the suffix, $s(m_{a\alpha}) < p(m_{\alpha b})$, or the combined pattern exceeds the maximum phrase length, $m_{a\alpha,1} \leq m_{\alpha b,K_{\alpha b}} + |w_{K_{a\alpha b}}| - MaxPhraseSpan$, then $m_{a\alpha b} \ddot{<} m_\alpha$.

Let's turn to the design of the QUERY algorithm that finds these partners. First, consider a simple algorithm INTERSECT on two sorted sets L_1 and L_2. It computes sorted output set $L_{1\cap 2}$ containing all elements common to both L_1 and L_2 by iteratively comparing the their topmost elements. At each iteration, the lesser of the two elements is popped from its respective set and discarded. If the elements are equal, both are popped and a copy is appended to $L_{1\cap 2}$. The complexity of INTERSECT is linear, $O(|L_1| + |L_2|)$.

We can implement QUERY using roughly the same logic as INTERSECT if $M_{a\alpha}$ and $M_{\alpha b}$ are sorted. For the moment, we will simply assume sortedness. Later, we will show how this property is maintained. The comparison operators that we have defined for partners will stand in for the comparisons in INTERSECT. However, we still need to address a few subtleties. An important difference between QUERY and INTERSECT should be apparent from an inspection of example inputs and outputs (Figure 4.10). It is possible that a matching from either input list partners with multiple matchings from the other set. We don't want to pop an item from the set until we are certain that we have found all of its partners.

Let's first consider a matching $m_{a\alpha}$ of prefix pattern $a\alpha$. Let $m_\alpha = s(m_{a\alpha})$. Any partner $m_{\alpha b}$ must satisfy the constraint $p(m_{\alpha b}) = m_\alpha$. From the definition of the prefix matching, we see that if $\alpha = \beta c$, then the only valid value for $m_{\alpha b}$ is in fact m_α. If $\alpha = \beta X$, then m_α completely determines all elements of $m_{\alpha b}$ except for the final one, $m_{\alpha b,K_{\alpha b}}$. In either case, since $M_{\alpha b}$ is sorted, all matchings of αb meeting the constraint must occur in the same (possibly empty) contiguous region of $M_{\alpha b}$. Sortedness further guarantees that for all elements $m_{\alpha b} \in M_{\alpha b}$ subsequent to this region, $m_{a\alpha} \ddot{<} m_{\alpha b}$ by definition of the comparison operators. It likewise guarantees that for all

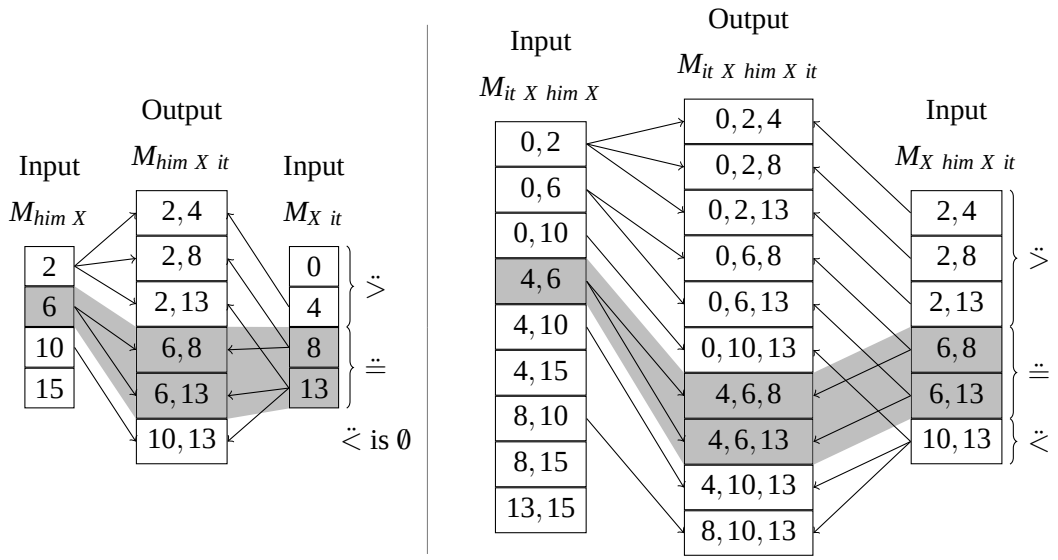


Figure 4.11: Examples showing the division of $M_{\alpha b}$ into regions by $m_{a\alpha} \in M_{a\alpha}$.

elements $m_{\alpha b} \in M_{\alpha b}$ prior to this region, $m_{a\alpha} \ddot{>} m_{\alpha b}$. Therefore, matching $m_{a\alpha}$ divides $M_{\alpha b}$ into three distinct regions, any of which may be empty (Figure 4.11). This means that we can pop $m_{a\alpha}$ when we encounter the first element $m_{\alpha b}$ such that $m_{a\alpha} \ddot{<} m_{\alpha b}$ or when we reach the end of the set.

Now let's consider a matching $m_{\alpha b}$ of suffix pattern αb. Let $m_\alpha = p(m_{\alpha b})$. Any partner $m_{a\alpha}$ must satisfy the constraint $s(m_{a\alpha}) = m_\alpha$. From the definition of the suffix matching, we see that if $\alpha = c\beta$, then there is only one valid value for $m_{a\alpha}$. In this case, sortedness ensures that $m_{\alpha b}$ divides $M_{a\alpha}$ into three regions, just as we saw for the prefix matchings. Matters are different if $\alpha = X\beta$. In this case, m_α completely determines all elements of $m_{a\alpha}$ except for the first one, $m_{a\alpha,1}$. This element could take several values. Furthermore, each of these values can be the first element of several matchings. Only one of these matching will meet our constraint, and thus the rest won't be partners with $m_{\alpha b}$. As a consequence, it is possible for partners of $m_{\alpha b}$ to be interspersed with non-partners (Figure 4.12). However, our definitions ensure that there is a contiguous range of possible values for $m_{\alpha b,1}$. Sortedness ensures that the set of all matchings $m_{a\alpha} \in M_{a\alpha}$ for which $m_{\alpha b,1}$ is in this range must occur in a contiguous region of $M_{a\alpha}$. Furthermore, our definitions ensure that for any value i, if $m_{a\alpha} \ddot{>} m_{\alpha b}$ for the first occurrence of a matching $m_{a\alpha}$ such $m_{ab\alpha,1} = i$, then for all subsequent matchings $m_{a\alpha} \in M_{a\alpha}$, $m_{a\alpha} \ddot{>} m_{\alpha b}$. An analogous case occurs for matchings $m_{a\alpha} \in M_{a\alpha}$ such that $m_{a\alpha} \ddot{<} m_{\alpha b}$. Therefore, $m_{\alpha b}$ divide $M_{a\alpha}$ into three regions, each of which may possibly be empty. However, it is possible for values in the central region to have any relationship with $m_{\alpha b}$, and therefore we must check all of them. We can safely pop $m_{\alpha b}$ when we encounter a matching $m_{a\alpha}$ such that $m_{a\alpha} \ddot{>} m_{\alpha b}$ and whose first element $m_{a\alpha,1}$ has not been seen before.

The full QUERY algorithm (Listing 3) is non-destructive – we advance a pointer rather than popping matchings from each set. Its operation is similar to INTERSECT, though we take a different approach to popping matchings from the stack. For each matching $m_{a\alpha} \in M_{a\alpha}$, we scan downwards from the current top of $M_{\alpha b}$, joining $m_{a\alpha}$ with any partners as we find them, until we encounter $m_{\alpha b}$ such that $m_{a\alpha} \ddot{<} m_{\alpha b}$. We then pop $m_{a\alpha}$. Elements $m_{\alpha b} \in M_{\alpha b}$ are popped whenever we uncover a new top matching $m_{a\alpha} \in M_{a\alpha}$ such that $m_{a\alpha} \ddot{>} m_{\alpha b}$ and whose first element $m_{a\alpha,1}$ has not been seen before.

The upper bound complexity of QUERY is $O(|M_{a\alpha}| \times |M_{\alpha b}|)$. However, most matchings

Algorithm 3 The basic QUERY algorithm

Function QUERY_INTERSECT

Input: sorted prefix matchings $M_{a\alpha}$ and sorted suffix matchings $M_{\alpha b}$

1: $M_{a\alpha b} \leftarrow \emptyset$
2: $I \leftarrow |M_{a\alpha}|$
3: $J \leftarrow |M_{\alpha b}|$
4: $j \leftarrow 0$
5: $m_1 \leftarrow -1$
6: **for** i from 0 to I **do**
7: $\quad m_{a\alpha} \leftarrow M_{a\alpha}[i]$
8: $\quad m_{\alpha b} \leftarrow M_{\alpha b}[j]$
9: $\quad$ **if** $m_{a\alpha,1} \neq m_1$ **then**
10: $\quad\quad$ **while** $m_{a\alpha} \ddot{>} m_{\alpha b}$ and $j < J$ **do**
11: $\quad\quad\quad j \leftarrow j+1$
12: $\quad\quad\quad m_{\alpha b} \leftarrow M_{\alpha b}[j]$
13: $\quad\quad$ **if** $m_{a\alpha} \ddot{>} m_{\alpha b}$ and $j = J$ **then**
14: $\quad\quad\quad$ Break
15: $\quad k \leftarrow j$
16: $\quad$ **while** not $m_{a\alpha} \ddot{<} m_{\alpha b}$ **do**
17: $\quad\quad$ **if** $m_{a\alpha} \ddot{=} m_{\alpha b}$ **then**
18: $\quad\quad\quad$ Append $m_{a\alpha} \bowtie m_{\alpha b}$ to $M_{a\alpha b}$
19: $\quad\quad$ **if** $k = J$ **then**
20: $\quad\quad\quad$ Break
21: $\quad\quad$ **else**
22: $\quad\quad\quad k \leftarrow k+1$
23: $\quad\quad\quad m_{\alpha b} \leftarrow M_{\alpha b}[k]$
24: **return** $M_{a\alpha b}$

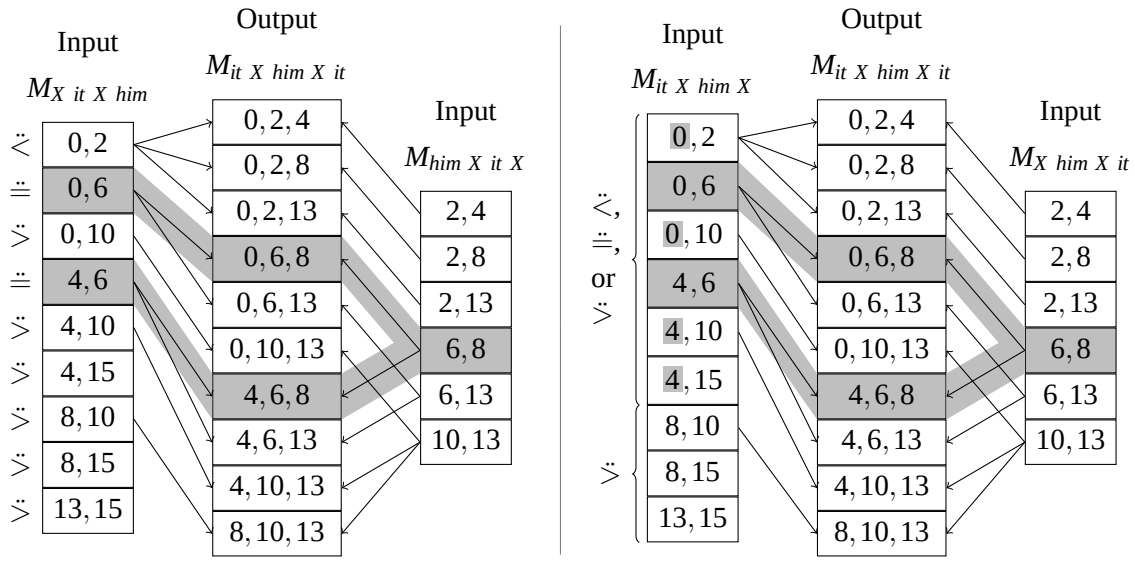


Figure 4.12: Example showing the relationships of $m_{a\alpha} \in M_{a\alpha}$ to an element $m_{\alpha b}$, and the subsequent delineation of $M_{a\alpha}$ into regions by $m_{\alpha b}$.

can be popped from their respective sets after a single comparison, so the average case complexity is closer to $O(|M_{a\alpha}| + |M_{\alpha b}|)$. Both upper bound and average complexity are identical to that of the baseline algorithm. As we've seen, though, our enumeration algorithm enables us to call it much less frequently using smaller input sets than in the baseline.

We assume that $M_{a\alpha}$ is sorted, and we process its matchings in order. Furthermore, for any $m_{a\alpha} \in M_{a\alpha}$, recall that its partners can only differ on their last element, and that these are encountered in sorted order. Therefore, the output of QUERY is sorted. This means that if we call QUERY using input sets that resulted from a previous invocation of QUERY, they will already be sorted. We must still ensure sortedness in the base case, which occurs when either prefix $a\alpha$ or suffix αb contains a single contiguous pattern. In this case, the set $M_{a\alpha}$ or $M_{\alpha b}$ is the result of a search in the suffix array. As we saw earlier, this set is not returned in sorted order. To solve this, we explicitly sort the occurrences by inserting them into a stratified tree (van Emde Boas et al., 1977) and reading the sorted sequence from the tree. Sort time is $O(|M_\alpha| \log\log |T|)$ for a pattern α. This is superlinear, but we need to sort only once, since we cache the result at the corresponding prefix tree node. Since we expect to query the text for many patterns containing this subpattern, the cost is amortized over all computations. Overall, this means that ensuring sortedness adds computational expense to our algorithm. This will be counterbalanced by additional strategies that exploit sortedness, introduced in the next sections.

4.3.1.6 Precomputation of Inverted Indices for Frequent Subpatterns

Sorting the matchings of a contiguous pattern w adds an $O(|M_w| \log\log |T|)$ term to query complexity. This is fine for infrequent patterns. However, if $|M_w|$ is large, this may be quite expensive.

We can circumvent this problem by precomputing an *inverted index* (Zobel and Moffat, 2006). This is simply the list $|M_w|$ in sorted order. It can be computed in one pass over the data. The memory consumption of inverted indices for all n-grams up to some maximum n requires

$n|T|$ space, so using this strategy for all n-grams is infeasible. Instead we precompute the inverted index only for the most frequent n-grams.[11] For less frequent n-grams, we continue to generate the index on the fly using stratified trees as before.

4.3.2 Faster Pattern Matching for Individual Query Patterns

The complexity of the comparison step in both the baseline algorithm and our merge algorithm is linear in the number of occurrences of each subpattern. Therefore, the main improvement we have introduced so far is reduction in the number of unnecessary lookups. The cost of pattern matching for a single query pattern is mostly unchanged, and as we have seen, it can be very expensive whenever the query pattern contains a frequent subpattern. However, there is a silver lining. Recall that patterns follow a Zipf distribution (Figure 3.11), so the number of pattern types that cause the problem is quite small. The vast majority of patterns are rare. Therefore, our solution focuses on patterns with one or more frequent subpatterns. To simplify matters, we focus on the intermediate computation for pattern $w_1 X \ldots X w_k$. This requires us to compute the collocation of subpatterns $w_1 X \ldots X w_{k-1}$ and w_k. There are three cases.

- If both patterns are rare, we use the QUERY algorithm (§4.3.1.5).
- If one pattern is frequent and the other is rare, we use an algorithm whose complexity depends mainly on the frequency of the rare pattern (see §4.3.2.1, below). It can also be used for pairs of rare patterns when one pattern is much rarer than the other.
- If both patterns are frequent, we resort to a precomputed intersection (see §4.3.2.2, below). We are not aware of any algorithms to substantially improve the efficiency of this computation at runtime, but the result can be precomputed in a single pass over the text.[12]

4.3.2.1 Fast Intersection via Double Binary Search

For collocations of frequent and rare patterns, we use a fast set intersection method for sorted sets called *double binary search* (Baeza-Yates, 2004). Suppose that we wish to intersect a sorted set Q with a much larger sorted set Q'. Note that we can compute this intersection efficiently by performing a binary search in Q' for each element of Q. The complexity is $\Theta(|Q| \log |Q'|)$, which is better than the INTERSECT algorithm complexity of $O(|Q| + |Q'|)$ if $Q \ll Q'$. Note that this is a tight bound.

Double binary search takes this idea a step further. It performs a binary search in Q' for the median element of Q. Whether or not the element is found, the result divides both sets into two pairs of smaller sets that can be processed recursively. In many cases, one of the recursive inputs will be empty, and we don't need to do any work at all. This results in a loose bound on complexity, $O(|Q| \log |Q'|)$, and the average case is often much better than this (Baeza-Yates, 2004; Baeza-Yates and Salinger, 2005). We can modify the algorithm to compute collocation rather than intersection, just as we did for the merge algorithm (§4.3.1.5).

If $|Q| \log |Q'| < |Q| + |Q'|$ then the performance is guaranteed to be sublinear in $|Q| + |Q'|$. Because the bound is loose, it is often sublinear even if $|Q| \log |Q'|$ is somewhat larger than $|Q| + |Q'|$. In our implementation we simply check for the condition $\lambda |Q| \log |Q'| < |Q| + |Q'|$ to decide

[11] We identify the most frequent patterns in a single traversal over the *longest common prefix (LCP) array*, an auxiliary data structure of the suffix array (Manber and Myers, 1993). We only need the LCP array for this purpose, so we compute it once offline using a fast algorithm due to Kasai et al. (2001).

[12] We combine this with the precomputation of inverted indices (§4.3.1.6).

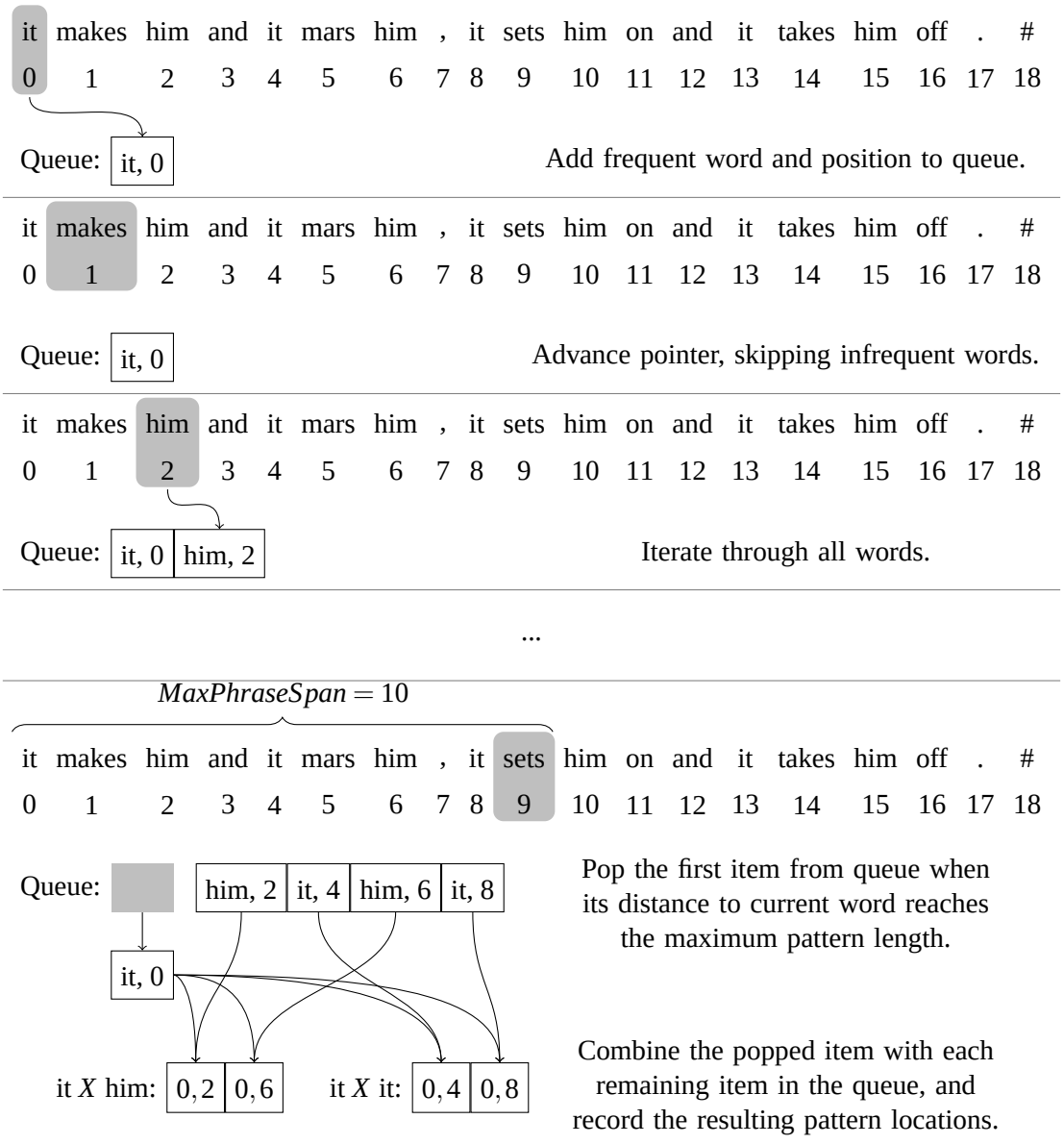


Figure 4.13: Illustration of the precomputation algorithm.

whether we should use double binary search or the merge algorithm. This check is applied in the recursive cases as well as for the initial inputs. The variable λ can be adjusted for speed. We explore possible values for it empirically in §4.5.1.

4.3.2.2 Precomputation of Collocations

Double binary search only helps if one subpattern is infrequent. If both subpatterns are frequent, there is no clever algorithm to efficiently compute their collocation at runtime. Therefore, we precompute these expensive collocations in a single pass over the text. As input, our algorithm requires the identities of the k most frequent contiguous patterns. We then iterate over the corpus.

Whenever a pattern from the list is seen, we push a tuple consisting of its identity and current location onto a queue. Whenever the oldest item on the queue falls on the edge of the maximum span window with respect to the current position, we pop it from the queue and compute its collocation with all other items in the queue (subject to any gap length constraints) . We repeat this step for every item that falls outside the window. At the end of each sentence, we compute collocations for any remaining items in the queue and then empty it. The algorithm is illustrated in Figure 4.13.

Our precomputation includes the most frequent n-gram subpatterns. Most of these are unigrams, though we found 5-grams among the 1000 most frequent patterns. We precompute the locations of source phrase uXv for any pair u and v that both appear on this list. There is also a small number of patterns uXv that are very frequent. We cannot easily obtain a list of these in advance, but we observe that they always consist of a pair u and v of patterns from near the top of the frequency list. Therefore we also precompute the locations of patterns $uXvXw$ in which both u and v are among these super-frequent patterns (all unigrams), treating this as the collocation of the frequent pattern uXv and frequent pattern w. We also compute the analogous case for $uXvXw$ when v and w are super-frequent.

4.3.2.3 Putting it all Together: The Root QUERY algorithm

We've described several algorithms for pattern matching on text, including suffix array lookup for contiguous patterns (§3.3.1) and multiple algorithms for discontiguous queries including the QUERY algorithm (§4.3.1.5), double binary search (§4.3.2.1), and cache retrieval (§4.3.2.2). To make clear when each of these algorithms is called, we include here the root QUERY algorithm that dispatches to the appropriate algorithm for each case (Listing 4). This is the only QUERY called for all phrase queries from the prefix tree algorithm (Listing 2).

4.4 Source-Driven Phrase Extraction

Our pattern matching algorithms are general. They can be used for many models that use discontiguous source phrases. Once we find all occurrences of a source phrase, we are faced with the task of extracting its translations. This is a model-specific problem, which we turn to in this section with a focus on hierarchical phrase-based translation.

Chiang (2005, 2007) bootstraps hierarchical phrase extraction from standard phrase extraction using a simple algorithm. First, all phrases meeting the standard phrase extraction heuristic are found (up to the maximum phrase span). Recall that this heuristic requires words in the phrase pair to be unaligned to words outside the phrase pair. Next, phrase pairs that are completely contained within larger phrase pairs are subtracted to form gaps.

We augment the source-driven extraction algorithm (§3.3.2) following the same principle. We first find the smallest reflected source span containing our source phrase. This acts as the main phrase from which smaller phrases are subtracted. We then take any parts of the source span corresponding to gaps in our query pattern, and subtract them from the large phrase pair.

There are a few subtleties. Consider a query pattern α. This is not the only source phrase that matches a given location in the text. Query patterns $X\alpha$, αX, and $X\alpha X$ also match the location (provided that they are licensed by the length restrictions). We want to extract all of these variants. This requires some modifications. Recall that in the basic algorithm, extraction failed if the reflected source span did not match the original source span. Suppose that we have found α in a particular span, and the reflected source span includes some words to the left of α. In the basic algorithm, this prevents extraction. However, in the hierarchical model, we can interpret the

Algorithm 4 The root QUERY algorithm

Function QUERY_ROOT

Input: pattern $a\alpha b$; one of the following: suffix array indices low $\ell_{a\alpha}$ and high $h_{a\alpha}$ if $\alpha = uX$, sorted prefix matchings $M_{a\alpha}$ otherwise; suffix array indices low $\ell_{\alpha b}$ and high $h_{\alpha b}$ if $\alpha = Xu$, sorted prefix matchings $M_{\alpha b}$ otherwise.

if $\alpha = u$ **then** ▷ α is a contiguous pattern
  return SUFFIX-ARRAY-LOOKUP($SA_F, a\alpha b, \ell_{a\alpha}, h_{a\alpha}$)
else ▷ α is a discontiguous pattern
  if $\alpha = uX$ **then** ▷ prefix is a contiguous pattern
    $M_{a\alpha} \leftarrow$ SORT_MATCHINGS($a\alpha, \ell_{a\alpha}, h_{a\alpha}$)
  if $\alpha = Xu$ **then** ▷ suffix is a contiguous pattern
    $M_{\alpha b} \leftarrow$ SORT_MATCHINGS($\alpha b, \ell_{\alpha b}, h_{\alpha b}$)
  if $M_{a\alpha b}$ has been precomputed **then**
    Retrieve $M_{a\alpha b}$ from cache of precomputations
  else
    if $|M_{a\alpha}| < |M_{\alpha b}|$ and $\lambda|M_{a\alpha}| \log |M_{\alpha b}| < |M_{a\alpha}| + |M_{\alpha b}|$ **then**
      $M_{a\alpha b} \leftarrow$ QUERY_DOUBLE_BINARY($M_{a\alpha}, M_{\alpha b}$)
    else if $|M_{\alpha b}| < |M_{a\alpha}|$ and $\lambda|M_{\alpha b}| \log |M_{a\alpha}| < |M_{\alpha b}| + |M_{a\alpha}|$ **then**
      $M_{a\alpha b} \leftarrow$ QUERY_DOUBLE_BINARY($M_{a\alpha}, M_{\alpha b}$)
    else
      $M_{a\alpha b} \leftarrow$ QUERY_INTERSECT($M_{a\alpha}, M_{\alpha b}$)
  return $M_{a\alpha b}$

Function SORT_MATCHINGS

Input: pattern α, suffix array indices ℓ and h

if inverted index M_α has been precomputed **then**
  return precomputed M_α from cache
else
  Let S be a stratified tree
  $M_\alpha \leftarrow \emptyset$
  for k from ℓ to h **do**
    INSERT($SA_F[k], S$)
  $k \leftarrow$ NEXT_ELEMENT($-1, S$)
  while $k \neq -1$ **do**
    APPEND(k, M_α)
  return M_α

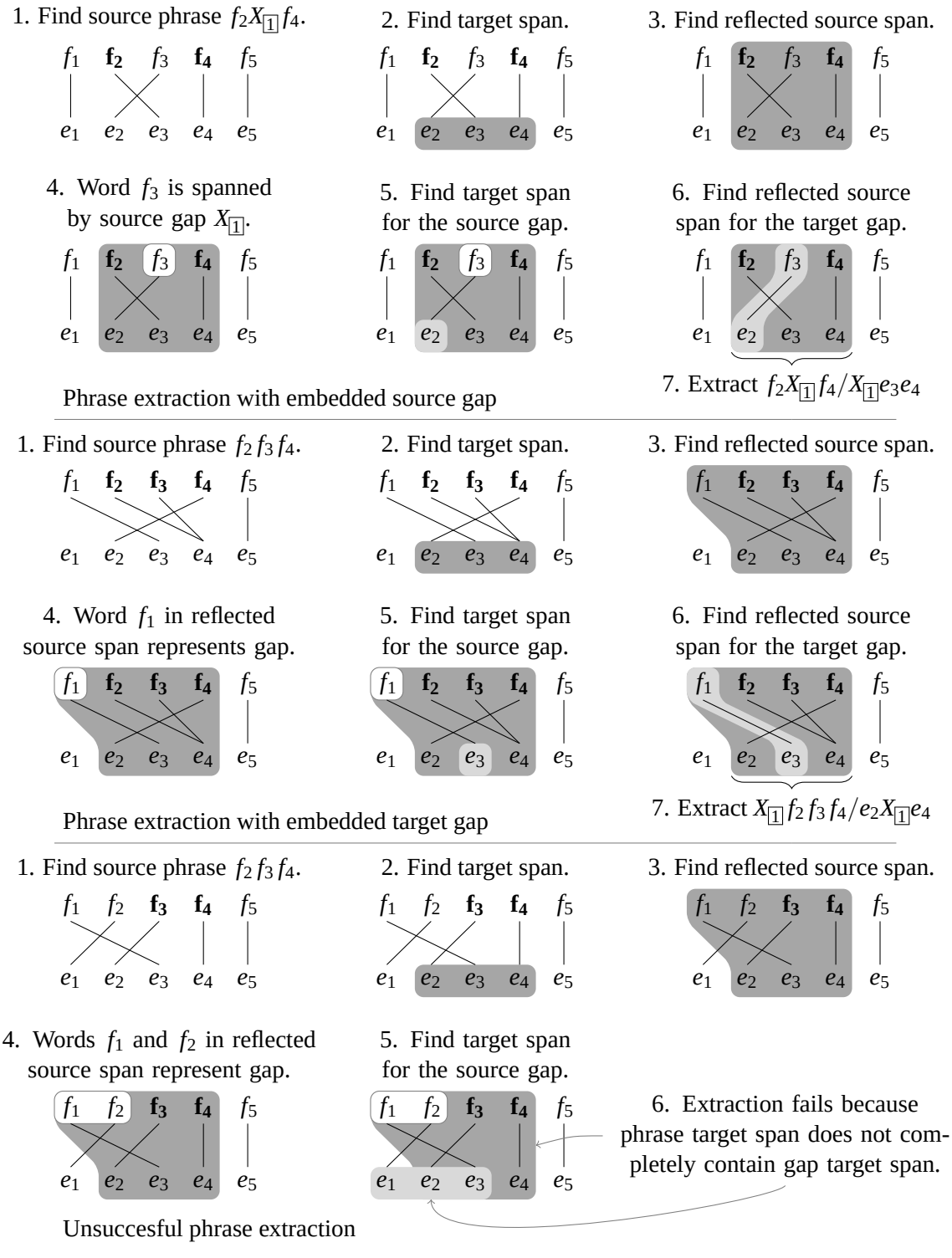


Figure 4.14: Examples of hierarchical base phrase extraction.

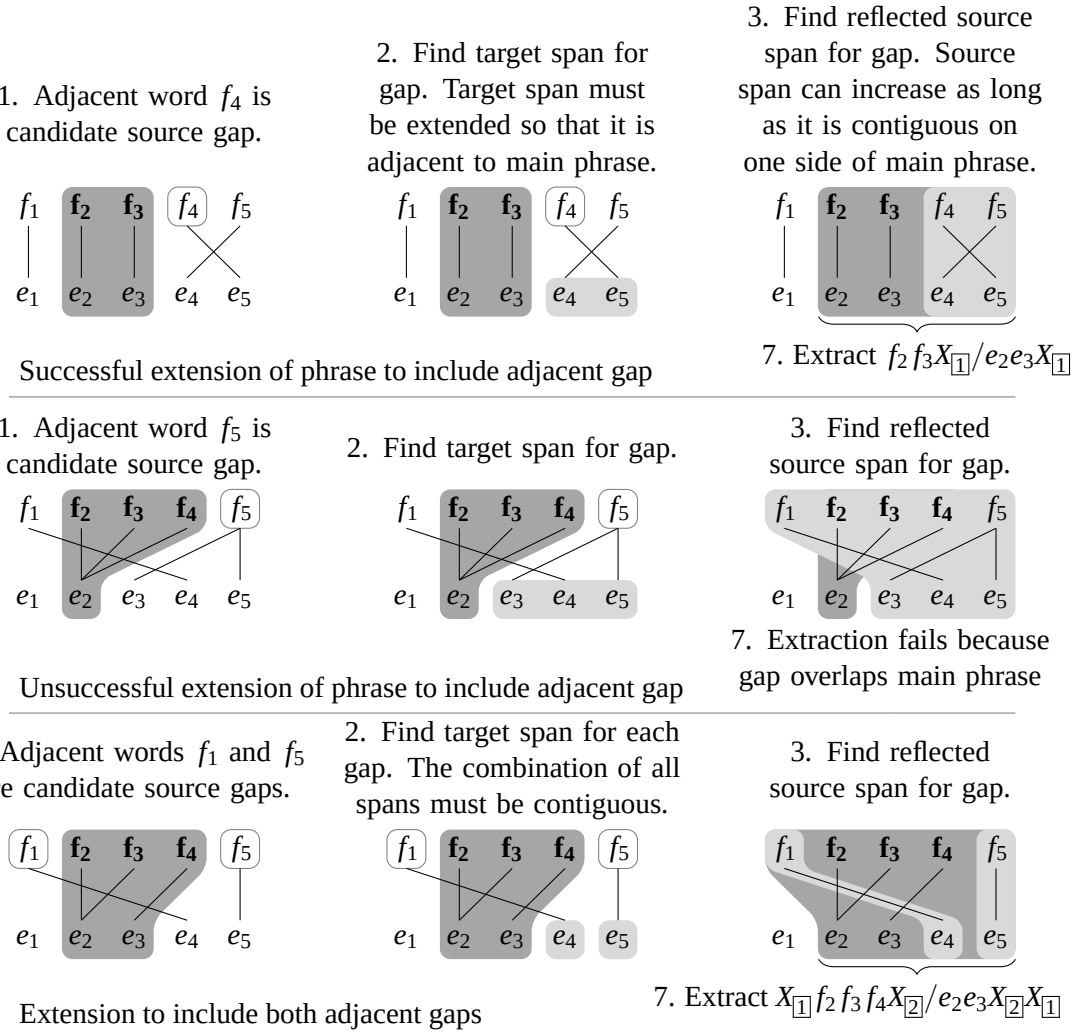


Figure 4.15: Examples of extended hierarchical phrase extraction.

reflected source span as $X\alpha$, provided that the new words in the source span form a valid phrase pair with words in the target span. Therefore, we modify the algorithm so that it adds these new words as a gap. For each gap in the reflected source span, including those that were part of the original query, we check to see whether they form a valid phrase under the previous definition (Figure 4.14).

We are not quite done. For each occurrence of source phrase α, we must check to see whether $X\alpha$, αX, and $X\alpha X$ can be extracted. This implies that a span of words adjacent to the right or left of the phrase forms a valid phrase whose target span is adjacent to the target span of α. The length of this span is unimportant, as long as the combined span is less than the maximum phrase span. Therefore, we initialize the span to a length of one, expand away from α by iteratively reflecting the span until a fixpoint is reached (Figure 4.15). If the final span combined with the main span is less than *MaxPhraseSpan*, then we extract the new phrase pair. We repeat this step for $X\alpha$, αX, and $X\alpha X$.

In order to exactly replicate the grammars in Chiang (2007), we check several additional constraints.

- That the number of nonterminal symbols in a rule is no more than *MaxNonterminals*.
- That the total number of terminal and nonterminal symbols in a rule is no more than *MaxPhraseLength*.
- That the span of neither source nor target phrase is more than *MaxPhraseSpan*.
- That the source phrase contain at least one aligned terminal.
- That all phrases are tight, that is, the edge words of both the main phrase and the aligned phrases must be aligned. Our implementation also allows us to relax this restriction (we will examine this in §5.2.3.3).

One final detail is required to complete our algorithm. Chiang (2007) uses the following strategy to count each extracted phrase: for each main phrase, a count of one is distributed uniformly over all possible phrases that can be formed from it via subtraction. We did not want to enumerate all possible phrases for each span, as many of them would not be source phrases in the current input sentence. Therefore our method diverges on this detail. We assign a count of one to each source span. Note that under the loose heuristic, it is possible to extract multiple target phrases for each source phrase. In this case we distribute the count uniformly over them.

4.5 Results

Our experimental decoder is Hiero (Chiang, 2007), an implementation of the hierarchical phrase-based translation model written in a combination of Python and Pyrex. For our baseline experiments using direct representation, we employ a modified version of this decoder (Dyer, 2007) that implements the grammar as an external prefix tree (Zens and Ney, 2007).

Our algorithms are implemented in Pyrex as extensions of our phrase-based implementation (§3.5).[13] This allows us to reuse much of the suffix array code. In fact, both our standard and

[13]Our implementation generates mostly compiled code, so it is substantially faster than initial results reported in Lopez (2007), which were obtained using a pure Python implementation. In order to conduct a fair comparison, we also reimplemented the baseline algorithm in the same way. It should be noted, however, that there is still room for increased speed in all algorithms via an optimized native C implementation.

hierarchical phrase-based decoders can be run from the same memory-mapped representation of the training data and alignment, though the hierarchical decoder requires additional data files to handle precomputation collocations and inverted indices.

As with the standard phrase-based model, we wished to see if hierarchical phrase-based translation by pattern matching is a viable replacement for direct representation. For this purpose, we adhere to the restrictions described by Chiang (2007) for rules extracted from the training data. We emphasize again that our algorithms do not depend in any way on these specific values. We will relax several of these restrictions in Chapter 5.

- Rules can contain at most two nonterminals.
- The source side of a rule can contain at most five symbols, which may be a mix of terminals and nonterminals.[14]
- Rules can span at most ten words in either training or test data.
- Nonterminals must span at least two words.
- Adjacent nonterminals are disallowed in the source side of a rule.

Expressed more economically, we say that our goal is to search for a source phrase α in the form u, uXv, or $uXvXw$, where $1 \leq |\alpha| \leq 5$, $1 \leq |u|$, $1 \leq |v|$, and $1 \leq |w|$.

Our experimental scenario is identical to the one for our phrase-based system, including all data as well as optimization and evaluation scripts (§3.5). Therefore the results are directly comparable. As before, we optimize every system configuration separately using MERT (§2.5.3.1; Och, 2003).

4.5.1 Timing Results

Our work included several distinct algorithmic enhancements.

- A prefix tree-based enumeration strategy that prunes many unnecessary queries (§4.3.1).
- Precomputed inverted indices to avoid sorting lists of locations for frequent patterns (§4.3.1.6).
- Double binary search to compute collocations between frequent and infrequent patterns (§4.3.2.1).
- Precomputed collocations for pairs of frequent patterns (§4.3.2.2).

For precomputation of both collocations and inverted indices, we chose to use the 1000 most frequent patterns. For precomputation of collocations involving super-frequent patterns (§4.3.2.2), we use the 10 most frequent patterns. For double binary search, the λ parameter (§4.3.2.1) was set to one.

Double binary search and both precomputation optimizations were implemented as extensions of the prefix tree algorithm, but are otherwise independent of each other. In order to measure their respective contributions, we ran the system using several different combinations of optimizations. Per-sentence query time, excluding extraction and decoding, is reported in Table 4.1. The

[14] Lopez (2007) restricted the source side to five *terminal* symbols, irrespective of nonterminals. The differs from the present restriction, which is identical to Chiang (2007) and results in a smaller grammar.

baseline	prefix tree	+1 optimization	+2 optimizations	+3 optimizations
		double binary 174.1	double binary collocations 7.0	
221.4	177.0	collocations 8.3	double binary indices 43.6	double binary collocations indices 0.97
		indices 47.2	collocations indices 2.3	

Table 4.1: Timing results for different combinations of algorithms (seconds per sentence), not including extraction or decoding time.

λ	0.2	0.4	0.6	0.8	1.0	2.0	3.0	4.0	5.0
seconds per sentence	1.00	0.97	0.97	0.97	0.97	1.02	1.05	1.08	1.10

Table 4.2: Effect of double binary λ parameter on per-sentence query time.

full complement of optimizations reduces the amount of computation from 221.4 seconds per sentence in the baseline to a mere 0.97 seconds per sentence, an improvement of over two orders of magnitude. We find from examining the various combinations that each algorithm is important to achieving the overall result, though the most critical pieces seem to be the precomputed collocations and inverted indices. Adding double binary search to these optimizations reduces the query time by 58% absolute. However, its contribution to overall time reduction is minor in all settings. In contrast, preliminary results (Lopez, 2007) suggested that it played a more important role in algorithmic improvements. This may reflect differences in the relative cost of operations in the our implementation platforms (Python versus C/Pyrex). Sanders and Transier (2007) suggest that the cost of recursive function calls in the double binary algorithm impede performance in efficient compiled implementations.

To study the performance of the double binary algorithm more closely, we experimented with the λ parameter used as a threshold to determine when the algorithm should be used. With the previous implementation (Lopez, 2007), we found that tuning this parameter yielded substantial differences in computation time. We turned on all optimizations and varied only this parameter. The results, given in Table 4.2, show that this parameter had very little effect on performance in this implementation.

Recall that our prefix tree could be used as a cache in the case of batch translations. To study the impact of this, we ran the decoder using the prefix tree as a full cache. We found that this only improved the per-sentence query time to 0.93 seconds versus 0.97 seconds without caching. It is interesting to note that caching performs a very similar function to our precomputed indices and collocations. We were curious to see whether caching could simulate these functions, so we disabled them and ran using the cache instead. Results are in Figure 4.16. We found that query

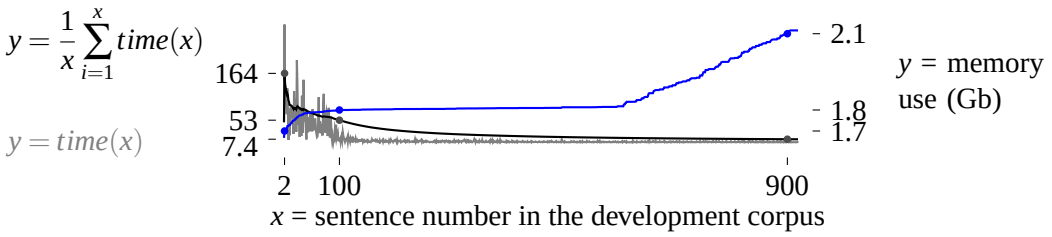


Figure 4.16: Effect of caching on average and per-sentence lookup time and memory use.

Number of frequent patterns	memory use	query time
1000	2.1G	0.97
100	1.1G	1.62

Table 4.3: Effect of precomputation on memory use and processing time. Here we show the memory requirement of the entire system (*sans* language model), including all data structures on the training data.

times were extremely slow for the first one hundred sentences, but afterwards, they converged towards query times using our optimizations. We also found that memory consumption grew over time. This highlights two important features of the precomputation optimizations. First, they prevent slow processing of the early inputs. Second, their overall memory consumption profile is stable. In contrast, with full caching memory consumption continues to grow indefinitely, which is unsuitable for large batch or online scenarios. We conclude that our optimizations enable more flexibility than a simple caching approach. However, we note that it may still be possible to combine our optimizations with partial caching (such as an LRU strategy) to achieve further speedups.

Finally, we observed that much of the gain from precomputation seemed to occur with very frequent patterns. To test this, we ran experiments using precomputed collocations and inverted indices for only the 100 most frequent patterns. We found that the cost in memory use was reduced by nearly half. In trade, per-sentence query time increased by 67%, but in absolute terms the increase was less than a second.

4.5.2 Translation Quality Results

We measured translation accuracy the same way that we did for our phrase-based system. It is important to note that the baseline hierarchical model uses a slightly different feature set (Chiang, 2007).

- A source-to-target phrase translation probability.
- A target-to-source phrase translation probability.
- A source-to-target lexical weight.
- A target-to-source lexical weight.
- A language model feature.
- A word count feature.

Configuration	BLEU
baseline with standard eight features	30.7
baseline without target-to-source translation feature	30.5
baseline without target-to-source translation or phrase count features	30.6
baseline without phrase count feature	30.7

Table 4.4: Baseline system results compared with systems missing one or more features.

sample size	time	BLEU
0	0.97	–
10	1.19	27.8
25	1.41	29.6
50	1.66	30.1
100	2.04	30.4
200	2.70	30.8
300	3.27	30.9
400	3.76	30.8
500	4.25	31.2
1000	6.25	30.9

Table 4.5: Effect of different sampling sizes on per-sentence times for query, extraction, and scoring and translation accuracy.

- A phrase count feature only for nonlexicalized rules (i.e. a rule containing no terminal symbols).
- A phrase count feature for lexicalized rules.

We did not use any special translation modules for numbers, dates, names, and bylines. Therefore, we leave out the associated features used by Chiang (2007).

We again study the effect of sampling and the effect of losing the baseline target-to-source feature, and we again noticed during development that the phrase count features did not correlate with accuracy. Therefore, we tested the effect of leaving out these features. The results on a baseline system using direct representation are given in Table 4.4. None of the differences were statistically significant. This confirms that neither the the target-to-source translation probability nor the phrase penalties are essential to the system. Therefore, we run the remaining experiments without the phrase penalties. By design, we must leave out the target-to-source probability.

Results for translation by pattern matching are given in Table 4.5. As with the phrase-based system, we find that we can match our baseline system results with a sampling size between two and three hundred. Minor improvements obtained with larger sample sizes were not statistically significant. For results reported in the remainder of this dissertation, we use a sample size of 300.

As before, we find that extraction adds significant time to the overall speed. This is partly an artifact of our implementation, which scans all alignment links for a sentence in order to compute the target span and its reflection. To improve efficiency, we could design an alignment data structure that only stores the indices of the rightmost and leftmost aligned words for each source and target word. This representation would enable random access, since the number of tokens would be equal to the size of the text, and thus its indexes would correspond. However, we are

able to replicate the results of a direct representation using quite a small sample size.

4.6 Conclusion

The innovations described in this chapter solve a computationally challenging puzzle of efficient pattern matching with discontiguous phrases. We believe this is intrinsically interesting from an algorithmic perspective. However, our main interest is that it enables us to apply translation by pattern matching to practically any MT model, provided that model-specific rule extraction algorithms are developed. In the next chapter, we will show that our algorithms facilitate streamlined experimentation with hierarchical models, and that we can improve hierarchical phrase-based translation in a way that is not practical with a direct representation.

5 Tera-Scale Translation Models

All models are wrong. Some models are useful.

–George Box

We've taken translation by pattern matching from a novel idea to a fully realized implementation and shown that it is a viable replacement for direct representation of models. Furthermore, we've solved a difficult algorithmic challenge in applying it to hierarchical phrase-based translation. These results match the current state of the art. Now we seek to improve on this result. To our knowledge, none of the past work in this area (Callison-Burch et al., 2005; Zhang and Vogel, 2005) has demonstrated improvement over a credible baseline.

The lack of prior positive results has encouraged skepticism. Zens and Ney (2007) specifically criticize the work of Callison-Burch et al. (2005). Their argument centers on two key points. First, they claim that the primary advantage posited by that work— use of longer phrases—does not lead to any improvements in translation quality. They cite both their own results and those of Koehn et al. (2003). Second, they object to the memory requirements of the training data and indices. Phrase tables can easily exceed these requirements, but they solve this by storing their phrase table in an external prefix tree. This drastically reduces memory use.

The external prefix tree is certainly useful. However, the argument employed to promote it overlooks some key points. First, though it may be true that longer phrases are not particularly useful, there are many other axes along which models can be scaled up. Second, the memory use of translation by pattern matching along these axes is constant. It supports very large models from a single underlying representation. In contrast, an external prefix tree representation must be generated for each model variant, and is proportional to model size. As model size increases, this becomes a significant drawback. We will show that translation by pattern matching enables us to use models that are impractical to implement using prefix trees. Since it computes parameters on an as-needed basis, there is no need to compute or store the entire model. This enables us to translate with arbitrarily large models.

There is an additional benefit. Since the underlying representation is static, it is possible to experiment with new models simply by modifying algorithmic parameters or model features. We never need to recompute a full model offline or design specialized extraction utilities. This makes translation by pattern matching an ideal platform for model prototyping.

In this chapter we apply translation by pattern matching to the problem of translation model scaling. Our experiments focus on hierarchical phrase-based translation, since it is superior to standard phrase-based translation (cf. §3.5, §4.5). We first illustrate the rapid prototyping capabilities of translation by pattern matching (§5.1). We then explore several scaling axes—some obvious and some novel—and show that we can substantially improve on the baseline system (§5.2.3). Our experiments highlight interesting characteristics of the model and interactions of various components. Finally, we conclusively demonstrate that reproducing our results with phrase tables would be highly impractical (§5.3).

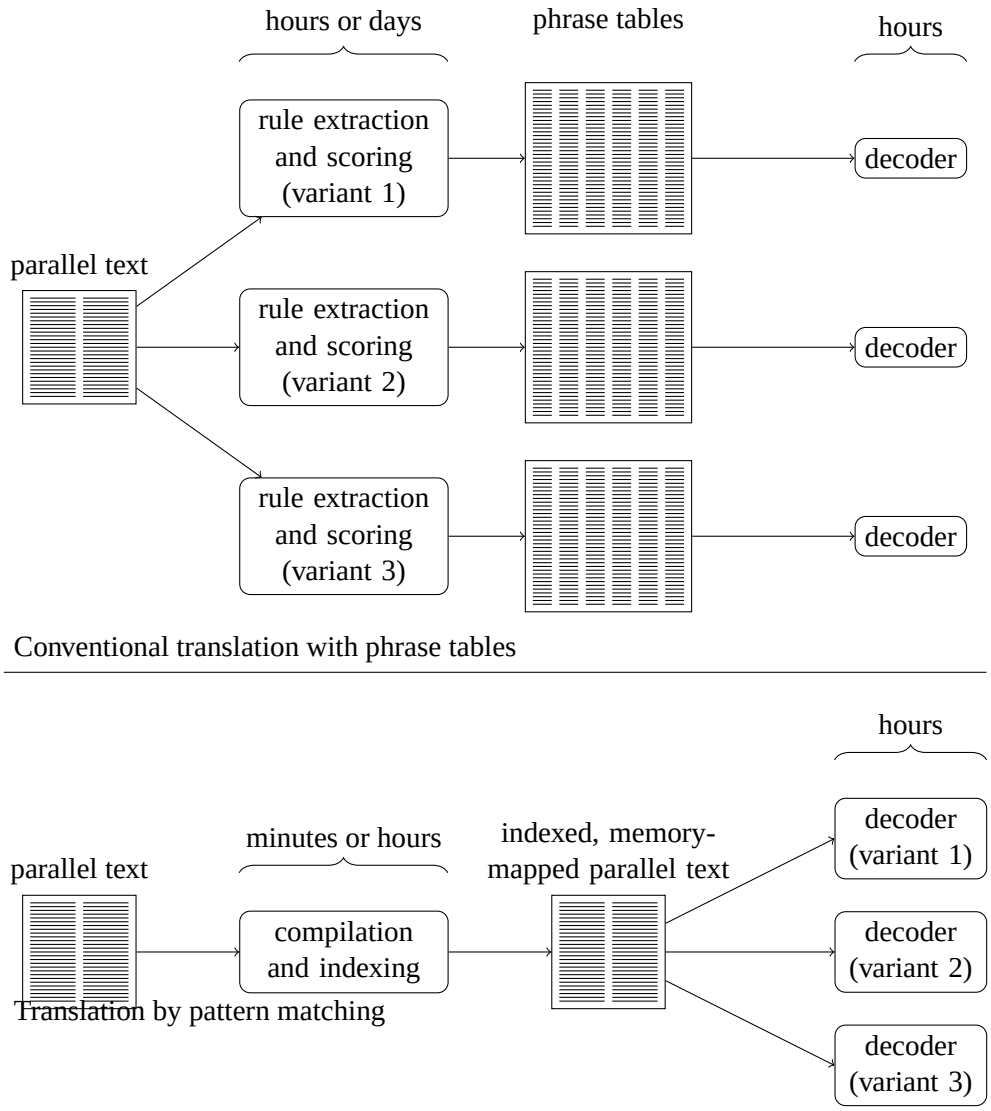


Figure 5.1: Experimental workflow for conventional architecture vs. translation by pattern matching.

5.1 Rapid Prototyping via Pattern Matching

Translation by pattern matching shortens experimental cycles by removing time-consuming offline model computation. Consider the conventional architecture illustrated in Figure 3.3. It requires us to re-extract rules and recompute all model parameters for each new model variant. This process can require several hours or even days. Translation by pattern matching enables us to explore many model variants because all models use the same underlying representation. It is generated once, often in a matter of minutes (Figure 5.1).

Generating a conventional system using an external prefix tree (Zens and Ney, 2007) takes a few steps.

1. Extract all rule instances from the corpus.

Method	CPU time (seconds)
Phrase tables (external prefix tree)	**39051**
Phrase extraction	31771
Phrase table scoring	5322
External prefix tree generation	1958
Translation by pattern matching	**445**
Suffix array construction (source)	100
Array construction (target)	40
Precomputations	126
Array construction (alignments)	96
Lexical weight computation	73

Table 5.1: CPU time needed to generate the baseline model.

2. Sort and score the rule instances.

3. Write the prefix tree a memory-mapped file.

In many experimental scenarios, we must redo these steps for each system variant. For instance, to experiment with different maximum phrase lengths, we must generate a new phrase table for each length. Furthermore, complexity depends on the size of the model. The time needed to sort and score rules increases with their number, so a model that generates more rules will take longer to create. In contrast, generating a pattern matching system requires the following steps.

1. Write the source training data as a memory-mapped suffix array.[1]

2. Write the target training data as a memory-mapped array.

3. Write the alignment as a memory-mapped array.

4. Write the lexical weighting table as a memory-mapped data structure.

5. Write precomputed collocations and inverted indexes as memory-mapped data structures.

The complexity of this offline initialization is mostly model-independent. Even when it is not, its modularity is a benefit. For instance, to use a different word alignment of the same corpus, we need only generate a new alignment data structure. To move from a standard phrase-based model to a hierarchical model, we only need to generate the precomputed collocations and inverted indices. The cost of generating these modular components is minor.

To make matters concrete, we measured the generation time for both representations of our baseline hierarchical model (§4.5). We generated a conventional system using an external prefix tree and a pattern matching system (precomputing for the 100 most frequent patterns). The results are given in Table 5.1. The size of each representation is illustrated in Table 5.2 The prefix tree representation takes nearly 90 times as long to produce and requires over 7 times as much memory as our indirect representation.

If we had a sufficiently large cluster, we might reduce the computation of phrase tables to a few hours using distributed methods.[2] However, this is still dependent on model complexity.

[1] For suffix array construction we use an algorithm due to Larsson and Sadakane (1999). Puglisi et al. (2007) identify several faster algorithms.

[2] Chris Dyer, personal communication.

Phrase Tables (external prefix tree)	
Rule instances extracted	195 million
Unique rules	67 million
Compressed extract file size	1.4 GB
Uncompressed extract file size	9.3 GB
Representation size	**6.1 GB**
Translation by Pattern Matching	
Source-side suffix array size	343 MB
Target-side array size	123 MB
Alignment size	119 MB
Lexical weight table size	27 MB
Precomputation size (top 100)	240 MB
Total representation size	**852 MB**

Table 5.2: Size of representations for the baseline model.

We now turn to the topic of scaling our models to a size that dramatically outstrips our ability to represent them using phrase tables.

5.2 Experiments

We exploited the rapid prototyping capabilities of our system to run a large number of experiments. Our experiments focus on scaling, but they also highlight some interesting characteristics of the underlying model.

5.2.1 Relaxing Grammar Length Restrictions

Callison-Burch et al. (2005) and Zhang and Vogel (2005) focused on two specific scaling axes for translation models: larger corpora and longer phrases. We generalize these ideas under the rubric of adding more data, and generating more rules from available data. We examine the former in §5.2.3 below. Here we are concerned with the latter. In hierarchical phrase-based translation, the analogue to an increase in maximum phrase length is relaxation of the grammar length restrictions (§4.1). We experimented with several such restrictions.

First, we considered the effect of the maximum phrase span (*MaxPhraseSpan*) for rules extracted from training data. Recall that this parameter limits the span of phrases that can be extracted from the training data. To see how this affects our ability to translate, suppose that an input sentence contains a source phrase uXv. It is possible that subpatterns u and v are collocated in a training sentence, but within a window longer than the maximum phrase span. In this case, the restriction prevents us from learning a translation rule that translates the collocated patterns as a single unit. Relaxing the span limit increases the chance of finding translation rules for collocated source phrases, or for finding additional examples of such rules. The risk is an increased chance of learning translation rules for otherwise meaningless collocations. Chiang (2007) fixes the limit at 10. To paint a complete picture of the effect of this parameter, we considered all values between 1 and 15. Results are reported in Table 5.3 (a). We found that accuracy plateaus just below the baseline setting.

Next, we considered the effect of the maximum phrase length (*MaxPhraseLength*), which limits the number of symbols that can appear in a source phrase. This is the closest analogue of

(a)

MaxPhraseSpan	BLEU
1	16.1
2	23.8
3	25.5
4	28.1
5	28.5
6	30.3
7	30.3
8	30.5
9	30.8
10	**30.9**
11	31.1
12	31.0
13	31.2
14	31.2
15	30.8

(b)

MaxPhraseLength	BLEU
1	16.9
2	25.4
3	29.7
4	30.4
5	**30.9**
6	30.9
7	30.8
8	30.8
9	30.9
10	30.7

Table 5.3: Effect of varying phrase span and phrase length parameters. Baseline values are in bold.

the maximum phrase length in a standard phrase-based model. By increasing this parameter, we increase the number of words (either contiguous or in collocated subpatterns) that can be translated as a single unit. As with the span parameter, we also run the risk of translating words based on purely coincidental collocation. We considered values between 1 and 10. Results are reported in Table 5.3 (b). As with the previous experiment, we found that accuracy plateaus just at the baseline setting.

The results here essentially extend the findings of Koehn et al. (2003) and Zens and Ney (2007) to the hierarchical case. Though scaling along these axes is unhelpful, there is still a large space for exploration and improvement.

5.2.2 Interlude: Hierarchical Phrase-Based Translation and Lexical Reordering

Along a related line of inquiry, we considered the effect of increasing the number of nonterminals in translated rules. Although Chiang (2007) limits the number of nonterminals to two, his decoder implementation can translate rules with arbitrary numbers of nonterminal symbols. For completeness, we also consider the effect of reducing this parameter to one or zero. We also extend this experiment to gain insight into the benefits of hierarchical model over a conventional phrase-based model.

A popular hypothesis posits that hierarchical phrase-based translation is essentially equivalent to *lexicalized reordering* in conventional phrase-based models (Al-Onaizan and Papineni, 2006; Koehn et al., 2005; Tillman, 2004).[3] Lexicalized reordering models parameterize reordering decisions on the words or phrases being translated. Proponents of this hypothesis argue that, since the synchronous grammar encodes reordering decisions along with lexical translations, it performs the same function. However, this differs from the typical justification for models that translate such phrases (Chiang, 2007; Quirk and Menezes, 2006; Simard et al., 2005, e.g.). They

[3] This hypothesis was suggested to me independently in personal communications with several researchers, including Chris Callison-Burch, Chris Dyer, Alex Fraser, and Franz Och.

Subpatterns	Nonterminals	Max Pattern	BLEU	Remark
1	0	u	26.3	monotone (cf. Chiang, 2007)
–	–	–	29.4	Moses, no lexicalized reordering
–	–	–	30.7	Moses with lexicalized reordering
1	1	uX, Xu	30.2	lexicalized reordering equivalent
1	2	XuX	30.0	lexicalized reordering equivalent
2	1	uXv	30.5	
2	2	$uXvX, XuXv$	30.8	
3	2	$uXvXw$	30.9	Hiero (Chiang, 2005, 2007)
4	3	$uXvXwXy$	30.9	
5	4	$uXvXwXyXz$	30.8	

Table 5.4: Effect of varying the maximum number of nonterminals and subpatterns.

argue that the ability to translate discontiguous phrases is central to the power of these models. If this is not true, then the focus on discontiguous phrases may be a dead end, since, as we have seen, their rulesets are quite unwieldy. We can instead simply pursue phrase-based translation using lexicalized distortion, since its representation is more compact.

To tease apart these claims, we make the following distinction. Hierarchical phrases containing a single subpattern—that is, phrases in the form u, Xu, uX, or XuX—are interchangeable with lexicalized reordering in a conventional phrase-based model. In contrast, hierarchical phrases representing discontiguous units—minimally uXv—encode translation knowledge that is strictly outside the purview of lexical reordering, specifically the translation of collocated patterns as a single unit. This delineation is arguably too fine, but for scientific purposes it is good first approximation, allowing us to cleanly interpret the results.

To test the lexical reordering hypothesis, we implemented a restriction on the number of subpatterns that could appear in either the source or target side of a hierarchical phrase. With a limit of one, only one subpattern can appear in a hierarchical phrase, though the phrase may contain between zero and two nonterminals.[4] We ran several experiments varying both the number of subpatterns and the number of nonterminals. Results are in Table 5.4. For comparison, we also include results of the phrase-based system Moses (Koehn et al., 2007) with and without lexicalized reordering.[5]

Our results are consistent with those found elsewhere in the literature. The most strict setting allowing no nonterminal symbols replicates a result in Chiang (2007, Table 7), with significantly worse accuracy than all other models. The most striking result is that the accuracy of Moses with lexicalized reordering is indistinguishable from the accuracy of the hierarchical system. Both improve over non-lexicalized Moses by about 1.4 BLEU. However, the hierarchical emulation of lexicalized reordering shows that the full hierarchical model owes its power only partly to this effect. Additional improvement is seen when we add discontiguous phrases. That the effect of lexicalized reordering is weaker in the hierarchical model is unsurprising, since its parameterization is much simpler than the one used by the Moses, which includes several spe-

[4] Note that this differs from a restriction on the number of nonterminals, which by definition must appear in equal number in both the source and target. This is not so with subpatterns. For instance, the rule $X \longleftarrow X_{\boxed{1}}u/vX_{\boxed{1}}w$ is a legal SCFG rule, though it contains only one source subpattern and two target subpatterns. Therefore we apply the restriction independently to both source and target phrases.

[5] This is a different decoder than the one we modified in §3.5, which does not support lexicalized reordering. The results with the Moses decoder are slightly better overall.

cialized features for this purpose. This suggests that the hierarchical model could be improved through better parameterization, and still benefit from the translation of discontiguous phrases.

Finally, we observe that using more than two nonterminals does not improve the hierarchical model.

5.2.3 Complementary Scaling Axes

So far, we have extended the results of Zens and Ney (2007) and Koehn et al. (2003) to hierarchical phrase-based translation, confirming that longer phrases do not substantially improve these systems. We have also shed some light on the interpretation of hierarchical phrase-based translation. While these are interesting scientific findings, they don't improve our baseline system. We now investigate other axes along which we can scale our models, some obvious and some novel.

Because we find these axes to be complementary, we present them together in this section, along with results showing improved translation accuracy. Our improved system incorporates a large amount of out-of-domain data (§5.2.3.1), sparse discriminatively trained word alignments (§5.2.3.2), loose phrase extraction heuristics (§5.2.3.3), and a novel parameterization enabled by our approach (§5.2.3.4). These factors represent a synthesis of recent trends in statistical machine translation and lead to substantial improvement (§5.2.3.5). In §5.3 we will show that this synthesis could not have been done without translation by pattern matching.

5.2.3.1 Out of Domain Data

Translation by pattern matching enables us to train from very large corpora. In our experiments, we were not able to find corpora that were too large to comfortably fit in memory on our research machines, each of which had 8GB RAM. To our knowledge, the training corpus of Chinese-English system that we used in our baseline systems is among the largest corpora of processed *in-domain* training data currently in use— that is, the genre of this training data is identical to that of the test data.[6] To move to larger corpora, we added a large quantity of *out-of-domain data* —data drawn from a different genre than the test, though in the same language pairs. We used a large parallel text of United Nations proceedings. It is almost three times the size of our in-domain training corpus, and makes our full training corpus among the largest assembled for Chinese-English translation tasks.[7] Corpus statistics are reported in Table 5.5. The corpus was tokenized, aligned, and otherwise preprocessed separately from the newswire corpus using identical tools.

Making the best use of out-of-domain data in statistical machine translation is a difficult problem that has only recently received any attention (Foster and Kuhn, 2007). The simplest strategy is to simply concatenate the out-of-domain data with the in-domain data. This raises a concern. A source phrase with a different sense in the out-of-domain data may occur so frequently that it outweighs the correct sense found in the in-domain data. Therefore, we treat each corpus independently. From each corpus, we sample up to 300 rule occurrences, and compute both lexical weighting and the source-to-target phrase translation probability separately on each sample. For the UN corpus, the resulting probabilities are incorporated into three new features. These features receive a value of zero for any rule computed from the newswire data. Likewise, the original source-to-target phrase translation probability and lexical weighting features receive a value of

[6] This corpus contains nearly all of the parallel newswire data available from the Linguistic Data Consortium.

[7] Our corpus is over half the size of one used by Zens and Ney (2007) for a Chinese-English task, and comparable in size to those used by Fraser and Marcu (2006) for Arabic-English and French-English tasks.

Corpus	Sentences	Chinese tokens	English tokens
Newswire	1.02 million	27 million	28 million
UN	2.53 million	82 million	79 million
Combined	3.55 million	107 million	107 million

Table 5.5: Sizes of experimental corpora.

zero for rules computed from the UN data. This is not as elegant as the model used by Foster and Kuhn (2007) but is similar and easy to implement in our setting.

5.2.3.2 Discriminatively Trained Word Alignments

The second axis that we vary concerns the underlying word alignment. Two trends are apparent in recent word alignment research. First, there has been substantial success in discriminative training of word alignments as measured by intrinsic criteria. Second, there is considerable debate over whether these improvements carry over to the translation task itself (Ayan and Dorr, 2006a; Fraser and Marcu, 2007a; Lopez and Resnik, 2006).

A discriminative aligner that improves in both intrinsic measures and translation tasks is the maximum entropy aligner of Ayan and Dorr (2006b). Their alignment improves translation accuracy from 24.0 to 25.6 BLEU over a standard aligner on the NIST 2003 Chinese-English task. This is a significant improvement. However, both the baseline and the improved system are far from state-of-the-art due to the relatively small training set of 107 thousand sentences.[8] The newswire corpus used in our experiments is ten times larger, and the full corpus including out-of-domain data is 35 times larger. Experience has shown that improvements found in small data conditions often do not hold in large data conditions (Fraser and Marcu, 2007a; Lopez and Resnik, 2006), so we believe this is important to test. Ayan and Dorr (2006b) also used a quite strict phrase length setting due to computational constraints. These constraints were disk space and offline computation time, which are not a concern our system due to its small footprint in both disk use and CPU time.

For comparison, we also considered the heuristically combined GIZA++ alignments used by our baseline system. In both cases, the in-domain and out-of-domain corpora were aligned separately.

5.2.3.3 Phrase Extraction Heuristics

Translation by pattern matching becomes extremely relevant when we consider the interaction of word alignment and phrase extraction heuristics. The GIZA++ alignments used in our baseline system are dense alignments. The number of alignment links is over 96% of the number of word tokens in each language. The actual percentage of aligned words may be somewhat less than this, since some words may participate in multiple alignments. However, we can safely assert that most words are aligned. Recall that the tight extraction heuristic requires aligned words at both the edges of main phrases and subtracted phrases (§3.3.2, §4.4). Since most words are aligned, this heuristic filters out relatively few candidate phrases. Conversely, the loose heuristic, which allows phrases to contain unaligned words at the borders of aligned phrases, will result in only a small number of additional extracted phrases.

[8] For comparison, our baseline system achieves a score of 32.9 BLEU on this task, though we use it as a development corpus.

In contrast, the maximum entropy alignments are sparse. The number of alignment links is less than 70% of the number of word tokens in each language. These alignments interact strongly with the extraction heuristics. Consider a phrase uXv containing two collocated subpatterns and a single gap. We will make the coarse assumption that any given word has only a 70% chance of being aligned. Since the tight heuristic requires that both the main phrase and the subphrase have aligned edge words, the chance that an arbitrary phrase meets this criterion is less than 25%. The situation is even worse for phrases with two gaps. In contrast, the loose heuristic not only removes this restriction, but permits the extraction of many variant translations containing unaligned words at the edges of the main phrase and subtracted phrases. This results in the extraction of a very large inventory of phrases. Ayan and Dorr (2006a) show that this interaction dramatically affects translation performance. They found that using the loose heuristic with sparse alignments improved accuracy by 3.7 BLEU over the tight heuristic. This finding is supported by Fraser and Marcu (2007a) and Lopez and Resnik (2006), who found that larger phrase inventories tend to improve translation accuracy, particularly for Chinese to English translation.

Scaling this result to a very large dataset is a natural application for translation by pattern matching, since it implies that a very large number of phrases would have to be processed if we constructed a phrase table offline. Indeed, Ayan and Dorr (2006a) do not provide results for some more permissive settings due to computational cost, even in their small data setting. Later, we will show that the interaction of loose extraction heuristics on sparse alignments on a large corpus produces a very large model §5.3.

5.2.3.4 Coherent Translation Units

An interesting modification to our baseline model is enabled by pattern matching. Recall from §3.1 that for every phrase, we compute a marginal for phrase translation probabilities using the number of rules containing the source phrase.

We saw in §3.3.2 and §4.4 that several conditions can prevent extraction of a translation for any particular occurrence of a source phrase. The target phrase might be too long, there might be unaligned words at its edges if we are using a tight extraction heuristic, the reflected source span might not match the main phrase, or one of the subtracted phrases violates these restrictions.

Translation by pattern matching gives us the ability to incorporate the knowledge that we gain from failed phrase extraction into our model. Informally, we think of this as follows. If a phrase occurs frequently in our corpus, but we are often unable to extract a translation for it, then our confidence that it represents a coherent unit of translation is diminished. Conversely, if we are usually able to extract a translation of the phrase, then we believe that the phrase is a natural unit of translation. We call this property *coherence*. Note that this property is not captured in the conventional offline extraction method in which only valid phrase pairs are extracted. If a phrase occurs many times but we can only extract a translation for it a few times, then those translations receive very high probabilities, even though they may only reflect coincidence or even misalignments.

We define coherence as the ratio of successful extractions to attempted extractions. This is added as a feature in our log-linear model. As an alternative, we can incorporate the notion of coherence directly into the phrase translation probability. We call this the coherent phrase-to-phrase translation probability. We found in preliminary experiments that these alternatives performed equally well. In the interest of keeping our model as simple as possible, we choose the latter option since it does not increase the number of features.

System	BLEU	loss
Best (all modifications)	32.6	–
with grow-diag-final-and alignment instead of maximum entropy alignment	32.1	-0.5
with tight extraction heuristic instead of loose	31.6	-1.0
without UN data	31.6	-1.0
without separate UN grammar	32.2	-0.4
with standard $p(f\|e)$ instead of coherent $p(f\|e)$	31.7	-0.9
Baseline (Phrase Tables)	30.7	-1.9
Baseline (Pattern Matching)	30.9	-1.7

Table 5.6: Results of scaling modifications and ablation experiments.

5.2.3.5 Results

As in §3.5 and §4.5, each system configuration was optimized separately on the development set. We repeat the baseline system results from §4.5 and show the results for a system combining all of the modifications described above. In addition, we performed ablation experiments to study the contribution of each modification to the improved system (Table 5.6). Our result improves significantly over the baseline. The improvement over the conventional baseline is 1.9 BLEU. Since this is a strong baseline, our improvements are quite substantial. We find that all the modifications described above contribute in some way to the improvement. The results highlight the complementary nature of our modifications. In some cases the removal of even one of them causes a loss of over one BLEU point.

To confirm these results, we ran two additional experiments. First, we compared the baseline and improved systems on the NIST 2006 Chinese-English task. Since we did not perform any other experimentation on this dataset it serves as a validation set. The baseline system scores 27.0, while our modified system improves this to 28.4, a statistically significant improvement.

We also compared our modified system against an augmented baseline using a 5-gram language model and rule-based number translation. The objective of this experiment is to ensure that our improvements are complementary to better language modeling, which often subsumes other improvements. The new baseline achieves a score of 31.9 on the NIST 2005 set, making it nearly the same as the strong results reported by Chiang (2007). Our modified system increases this to 34.5, a substantial improvement of 2.6 BLEU.

5.3 A Tera-Scale Translation Model

To get a sense of the what our improvements would require if we attempted them in a system based on direct representation, we estimated size of the synchronous grammar table that would be equivalent to the configuration of our best system. Due to constraints on disk space and time, we did not construct the table itself, for reasons that will become clear below. We modified the extractor to simply report the number of rules that it would normally extract under our model setting. To estimate phrase table size, we assume that the disk space requirements and the number of scored rules grows linearly in the number of extracted rules.[9] Therefore, we simply multiply the numbers in Table 5.1 by the ratio of extracted rules between the improved and baseline systems.

The number of extracted rules and the resulting estimates of phrase table size (Table 5.7) show that it would be completely impractical to compute such a phrase table. The large corpus,

[9]This is perhaps conservative, since our corpus is heterogeneous, and since the ratio of unique rules tends to increase with their overall number.

Phrase Tables (external prefix tree)	
Extract time	**77 CPU days**
Actual rule instances extracted	19.3 billion
Estimated unique rules	6.6 billion
Estimated compressed extract file size	135 GB
Estimated uncompressed extract file size	917 GB
Estimated total representation size	**604 GB**
Translation by Pattern Matching	
Total generation time	**1.3 CPU hours**
Source-side suffix array size	1.36 GB
Target-side array size	461 MB
Alignments size	311 MB
Lexical weight table size	63 MB
Precomputation size (top 100)	1.24 GB
Total representation size	**3.4 GB**

Table 5.7: Generation times and size of representations for the improved model (cf. Table 5.2).

sparse alignments, and loose extraction heuristics combine to produce a number of rule occurrences that is two orders of magnitude larger than in the baseline system. Merely counting all of these occurrences took almost five days on our 17-node research cluster. We estimate that the resulting files would require about a terabyte of disk space. Assuming that we even had adequate external algorithms to sort, score, and generate a memory-mapped prefix tree for such large files, we hesitate to estimate how long it would take. This estimate does not even address the question of computing the coherent phrase translation probabilities in an offline setting. In order to compute this feature using offline methods, we would need to extract an event for each occurrence of a pattern that might be considered a legal phrase, even if the occurrence was not part of an extractable phrase pair. This would substantially inflate the number of events extracted from the corpus and increase the amount of offline computation beyond the estimate reported here.

By comparison, it took 1.3 hours to generate the data files for our pattern matching decoder on a single CPU, and we completed the full complement of experiments in Table 5.6 in substantially less time than it took to simply count the rules that would be generated in the conventional system.

5.4 Conclusions

Our results demonstrate that translation by pattern matching enables practical, rapid exploration of vastly larger models than those currently in use. As a first step in this direction, we have explored a number of different scaling axes, and shown that we can significantly outperform a strong baseline generated with a state-of-the-art model. Furthermore, the rapid prototyping capabilities of our system enable experimentation with many different model variants using a single representation that is easy and efficient to generate. We explored some interesting characteristics of the hierarchical phrase-based translation model using this capability, finding that while longer phrases are no more helpful in the hierarchical setting than in the standard setting, the use of discontiguous phrases adds power to hierarchical models, in addition to acting as a simplified type of lexicalized distortion.

6 Conclusions and Future Work

We can only see a short distance ahead, but we can see plenty there that needs to be done.

–Alan Turing

Algorithms have been central to the development of statistical machine translation (Chapter 2). Stack decoding algorithms (e.g., Koehn, 2004a; Wang and Waibel, 1997) originating in information theory (Jelinek, 1969) and classical artificial intelligence (Hart et al., 1968) were essential to the development of early models. Context-free parsing algorithms originating in the parsing community (Cocke, 1970; Earley, 1970; Kasami, 1965; Younger, 1967) have been pivotal to the development of hierarchical translation models, and continue to produce new variants and embellishments (Huang and Chiang, 2007; Venugopal et al., 2007). Minimum error rate training (Och, 2003) is a novel optimization procedure originating in the statistical machine translation community that has been central to rapid progress over the last several years.

This dissertation continues the tradition of algorithmic innovation in statistical machine translation. We turned a novel idea for scaling (Callison-Burch et al., 2005; Zhang and Vogel, 2005) into a fully realized algorithmic framework to address a crucial scaling problem (Chapter 3) and solved a difficult algorithmic puzzle to make it applicable to a wide range of models (Chapter 4). Our algorithms draw on pattern matching techniques originally developed in bioinformatics and information retrieval. We applied our framework and significantly improved a strong baseline, showing that it could support models vastly larger than the current state of the art (Chapter 5).

The contributions of this dissertation include the following.

- We generalized a previous idea, bringing it to fruition as a complete algorithmic framework called machine translation by pattern matching. It enables scaling to large data, richer modeling, and rapid prototyping.
- We identified open questions in the prior work that provides the foundation for our framework. We solved these problems, giving state-of-the-art performance in standard phrase-based translation models.
- We introduced a novel approximate pattern matching algorithm that extends our framework to a variety of important models based on discontiguous phrases. Our algorithm solves a challenging computational puzzle. It is two orders of magnitude faster than the previous state-of-the art, and enables a number of previously difficult or impossible applications.
- We showed how our framework can be applied to scale statistical translation models along several axes, leading to substantial improvement in translation accuracy over a state-of-the-art baseline.
- We showed that our method can scale up to extremely large models, at least two orders of magnitude larger than most contemporary models.

- We include the most comprehensive contemporary survey of the field of statistical machine translation.

We believe that these results only scratch the surface of the potential of translation by pattern matching. Our algorithms enable the development and application of models that are at least two orders of magnitude larger than most contemporary models. This makes them applicable to both the ever-increasing size of parallel corpora, and the development of more complex models trained on these corpora.

6.1 Future Work

A number of obvious extensions to the work in this thesis include extension to syntactic phrase-based models (DeNeefe et al., 2007; Galley et al., 2004, 2006) and factored translation models (Koehn and Hoang, 2007; Koehn et al., 2007). The former requires only a model-specific source-driven extraction algorithm, while the latter requires additional innovations in pattern matching, since it depends on a layered representation of training and input data.

We also plan to incorporate discriminative training methods into our framework. Chan et al. (2007), Carpuat and Wu (2007), and Subotin (2008) improve translation accuracy by recasting target phrase selection as a multi-class classification problem. They train a local classifier using contextual features of the source phrase. A drawback is that model complexity is substantially increased due to the richness of the feature space, harming the efficiency of offline training methods. However, these features are easy to obtain at runtime using our approach, which finds source phrases in context. We could investigate other novel features using this framework. For instance, we could consider using entire document context, which has not received much attention in the statistical translation literature.

Although our algorithms are quite fast, we believe that their efficiency can be substantially improved. This is especially true in the extraction step. We can improve this using an online/offline mixture (Zhang and Vogel, 2005), in which we extract a small, fully computed phrase table containing only the most frequent source phrases, and use translation by pattern matching for phrases not found in this table. This is the same strategy that we used to break down the pattern matching problem in Chapter 4, by focusing on the most frequent phrases. Since the number of frequent types is small, this limited extraction would be feasible even for the extravagantly large model we developed in Chapter 5. Although our current decoder is only about twice as slow as decoding with a direct representation, this improvement would enable our decoder to achieve speed close to that of offline methods, while retaining the scaling capabilities of our approach.

In our experiments, we were able to keep a very large corpus in memory without substantially taxing our moderate cluster resources. However, the available parallel data are constantly growing, particularly with continued development of web-mined and comparable corpora. Extension to factored models may also significantly tax memory resources. Therefore, we may consider using more efficient representations. Navarro and Mäkinen (2007) describe compressed self-indexes, a class of data structures that support both random access and fast pattern matching while consuming only slightly more than the information-theoretic minimum space required by the data. They are several times more compact than suffix arrays, and would thus allow scaling to substantially larger corpora without significant changes to the architecture we have laid out in this dissertation.

Moving beyond just translation, there is an affinity between natural language processing and bioinformatics. Where natural language processing seeks to analyze and manipulate sequences of words in human language, bioinformatics seeks to analyze and manipulate sequences of proteins

and other biological markers. Unsurprisingly, there is a some history of cross-development. Recently, there has been some interest in the application of natural language parsing techniques to protein folding problems (Hockenmaier et al., 2006). Synchronous grammars used in machine translation can also be used in the analysis of protein structures (Chiang, 2004). It has been shown that suffix arrays and prefix trees, the basis of many pattern matching algorithms in bioinformatics, have novel applications in the analysis of corpora (Yamamoto and Church, 2001) and parallel corpora (McNamee and Mayfield, 2006).

This dissertation deepens this relationship by using bioinformatics algorithms to improve machine translation. As discussed in §4.2.1, the pattern matching problem for discontiguous phrases is a variant of the approximate pattern matching problem. A major application of approximate pattern matching in bioinformatics is query processing in protein databases for purposes of sequencing, phylogeny, and motif identification (Gusfield, 1997). Using our approximate pattern matching algorithms, we conjecture that machine translation could be treated very much like search in a protein database. In this scenario, the goal is to select training sentences that match the input sentence as closely as possible, under some evaluation function that accounts for both matching and mismatched sequences, as well as possibly other data features.

Taking this scenario further, we imagine deepening the connection to bioinformatics by applying multiple sequence alignment algorithms to the translation problem. These algorithms take a set of similar sequences and attempt to find an optimal alignment of their matching subsequences (Durbin et al., 1998). One natural outcome that can be computed from the multiple sequence alignment is the consensus alignment. Suppose that we find many partial matches of a sentence in our training data. We could apply multiple sequence alignment to their extracted translations to synthesize a new translation. Where most current decoding algorithms force the translations of overlapping source phrases to compete, this method reinforces their common elements. Therefore, it might be a better use of the sparse evidence in our training data. Similar algorithms have been used in translation system combination (Rosti et al., 2007), speech recognition (Mangu et al., 2000), and automatic summarization (Barzilay and Lee, 2003).

Finally, our work coincides with related efforts in the use of string kernels, a type of kernel used in machine learning methods that computes the overlap of non-contiguous subsequences between documents (Lodhi et al., 2002). The pattern matching algorithm for hierarchical phrases can be seen as computing a string subsequence kernel between an input sentence and the entire training text, a quite difficult problem. String kernels and related learning techniques are becoming widespread in statistical language processing and computational linguistics. More broadly, these techniques are related to the notion of distributional similarity between words, which has received substantial attention in computational linguistics (e.g. Lee, 1999; Manning and Schütze, 1999; Pereira et al., 1993). Our algorithms could be used to investigate such hypotheses on a very large scale using kernel methods. However, they also have application in small scale, where they could be used to extract all sets of subsequences from small parallel corpora, thus making the most use of finite resources.

6.2 Concurrent Trends and Related Work

Other work contemporary with ours has focused on scalable representations of language models, complementing to our work. Brants et al. (2007) and Zhang et al. (2006b) use distributed representations of extremely large language models. Notably, the latter approach is based on suffix arrays, an index data structure used in our pattern matching algorithms. Talbot and Osborne (2007a,b) employ a Bloom filter, a data structure with enormous compression rates that admits a

small probability of error. This enables them to store a very large language model in a surprisingly small amount of memory. They aim to put large-scale models in reach of researchers with moderate computing resources. This encourages widespread experimentation with datasets and models that would only be accessible to those with large computing clusters. In this sense it is most complementary to our own work. A system combining their language model representation and our translation model representation could scale to extremely large models of both while using only a modest amount of memory.

Bibliography

Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, Mar 2004.

Alfred V. Aho and Jeffrey D. Ullman. Syntax directed translations and the pushdown assembler. *Journal of Computer and System Sciences*, 3:37–57, 1969.

Lars Ahrenberg, Magnus Merkel, Anna Sågvall Hein, and Jörg Tiedmann. Evaluation of word alignment systems. In *Proc. of LREC*, volume 3, pages 1255–1261, May 2000.

Yaser Al-Onaizan and Kishore Papineni. Distortion models for statistical machine translation. In *Proc. of ACL-COLING*, pages 529–536, Jul 2006.

Yaser Al-Onaizan, Jan Curin, Michael Jahr, Kevin Knight, John Lafferty, Dan Melamed, Franz Josef Och, David Purdy, Noah A. Smith, and David Yarowsky. Statistical machine translation: Final report. Technical report, Johns Hopkins University Center for Speech and Language Processing, 1999.

Joshua Albrecht and Rebecca Hwa. Regression for sentence-level MT evaluation with pseudo references. In *Proc. of ACL*, pages 296–303, Jun 2007.

Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. Learning dependency translation models as collections of finite state head transducers. *Computational Linguistics*, 26(1):45–60, Mar 2000.

Necip Fazil Ayan. *Combining Linguistic and Machine Learning Techniques for Word Alignment Improvement*. PhD thesis, University of Maryland, Nov 2005.

Necip Fazil Ayan and Bonnie Dorr. Going beyond AER: An extensive analysis of word alignments and their impact on MT. In *Proc. of ACL-COLING*, pages 9–16, Jul 2006a.

Necip Fazil Ayan and Bonnie J. Dorr. A maximum entropy approach to combining word alignments. In *Proc. of HLT-NAACL*, pages 96–103, Jun 2006b.

Necip Fazil Ayan, Bonnie Dorr, and Christof Monz. Neuralign: Combining word alignments using neural networks. In *Proc. of HLT-EMNLP*, pages 65–72, Oct 2005a.

Necip Fazil Ayan, Bonnie Dorr, and Christof Monz. Alignment link projection using transformation-based learning. In *Proc. of HLT-EMNLP*, pages 185–192, Oct 2005b.

Ricardo Baeza-Yates. A fast intersection algorithm for sorted sequences. In *Proc. of Combinatorial Pattern Matching*, number 3109 in LNCS, pages 400–408, Berlin, 2004. Springer-Verlag.

Ricardo Baeza-Yates and Alejandro Salinger. Experimental analysis of a fast intersection algorithm for sorted sequences. In M. Consens and G. Navarro, editors, *Proc. of SPIRE*, number 3772 in LNCS, pages 13–24, Berlin, 2005. Springer-Verlag.

Rafael E. Banchs, Josep M. Crego, Adrià de Gispert, Patrik Lambert, and José B. Mariño. Statistical machine translation of euparl data by using bilingual n-grams. In *Proc. of ACL Workshop on Building and Using Parallel Texts*, pages 133–136, Jun 2005.

Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proc. of ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Jun 2005.

Colin Bannard and Chris Callison-Burch. Paraphrasing with bilingual parallel corpora. In *Proc. of ACL*, pages 597–604, Jun 2005.

Regina Barzilay and Lillian Lee. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *Proc. of HLT-NAACL*, pages 16–23, May 2003.

Leonard E. Baum. An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process. In *Proceedings of the Third Symposium on Inequalities*, volume 3 of *Inequalities*, pages 1–8. Academic Press, 1972.

Adam L. Berger, Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, John R. Gillett, John D. Lafferty, Robert L. Mercer, Harry Printz, and Luboš Ureš. The Candide system for machine translation. In *Proc. of the ARPA Workshop on Human Language Technology*, pages 157–162, Mar 1994.

Adam L. Berger, Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Andrew S. Kehler, and Robert L. Mercer. Language translation apparatus and method using context-based translation models, Apr 1996a. United States Patent 5510981.

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, Mar 1996b.

Alexandra Birch, Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Constraining the phrase-based, joint probability statistical translation model. In *Proc. of HLT-NAACL Workshop on Statistical Machine Translation*, pages 154–157, Jun 2006.

Phil Blunsom and Trevor Cohn. Discriminative word alignment with conditional random fields. In *Proc. of ACL-COLING*, pages 65–72, Jul 2006.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In *Proc. of EMNLP-CoNLL*, pages 858–867, Jun 2007.

Peter F. Brown, John Cocke, Stephen Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, Jun 1990.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based *n*-gram models of natural language. *Computational Linguistics*, 18(4):467–479, Dec 1992.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19 (2):263–311, Jun 1993.

Ralf D. Brown. A modified Burrows-Wheeler transform for highly-scalable example-based translation. In *Proc. of AMTA*, number 3265 in LNCS, pages 27–36. Springer, Sep 2004.

Andrea Burbank, Marine Carpuat, Stephen Clark, Markus Dreyer, Pamela Fox, Declan Groves, Keith Hall, Mary Hearne, I. Dan Melamed, Yihai Shen, Andy Way, Ben Wellington, and Dekai Wu. Final report of the 2005 language engineering workshop on statistical machine translation by parsing. Technical report, Johns Hopkins University Center for Speech and Language Processing, Nov 2005.

Chris Callison-Burch, David Talbot, and Miles Osborne. Statistical machine translation with word- and sentence-aligned parallel corpora. In *Proc. of ACL*, pages 176–183, Jul 2004.

Chris Callison-Burch, Colin Bannard, and Josh Schroeder. Scaling phrase-based statistical machine translation to larger corpora and longer phrases. In *Proc. of ACL*, pages 255–262, Jun 2005.

Chris Callison-Burch, Philipp Koehn, and Miles Osborne. Improved statistical machine translation using paraphrases. In *HLT-NAACL 2006*, Jun 2006a.

Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluating the role of BLEU in machine translation research. In *Proc. of EACL*, pages 249–256, Apr 2006b.

Chris Callison-Burch, Cameron Fordyce, Philipp Koehn, Christof Monz, and Josh Schroeder. (meta-) evaluation of machine translation. In *Proc. of the Workshop on Statistical Machine Translation*, pages 136–158, Jun 2007.

Marine Carpuat and Dekai Wu. Word sense disambiguation vs. statistical machine translation. In *Proc. of ACL*, pages 387–394, Jun 2005.

Marine Carpuat and Dekai Wu. Improving statistical machine translation using word sense disambiguation. In *Proc. of ACL*, pages 61–72, Jun 2007.

Yee Seng Chan, Hwee Tou Ng, and David Chiang. Word sense disambiguation improves statistical machine translation. In *Proc. of ACL*, pages 33–40, Jun 2007.

Eugene Charniak, Kevin Knight, and Kenji Yamada. Syntax-based language models for statistical machine translation. In *Proc. of MT Summit IX*, Sept 2003.

Ciprian Chelba and Frederick Jelinek. Exploiting syntactic structure for language modeling. In *Proc. of ACL-COLING*, pages 225–231, Aug 1998.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Computer Science Group, Harvard University, Aug 1998.

Colin Cherry and Dekang Lin. A probability model to improve word alignment. In *Proc. of ACL*, Jul 2003.

David Chiang. *Evaluation of Grammar Formalisms for Applications to Natural Language Processing and Biological Sequence Analysis*. PhD thesis, University of Pennsylvania, 2004.

David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proc. of ACL*, pages 263–270, June 2005.

David Chiang. An introduction to synchronous grammars. Part of a tutorial given at ACL 2006, Jul 2006.

David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228, 2007.

David Chiang, Adam Lopez, Nitin Madnani, Christof Monz, Philip Resnik, and Michael Subotin. The Hiero machine translation system: Extensions, evaluation, and analysis. In *Proc. of HLT-EMNLP*, pages 779–786, Oct 2005.

Kenneth Church and Eduard Hovy. Good applications for crummy machine translation. *Machine Translation*, 8:239–258, 1993.

Kenneth Church and Ramesh Patil. Coping with syntactic ambiguity or how to put the block in the box on the table. *Computational Linguistics*, 8(3–4):139–149, Jul 1982.

John Cocke. *Programming languages and their compilers*. Courant Institute of Mathematical Sciences, New York University, 1970.

Michael Collins, Jan Hajič, Lance Ramshaw, and Christoph Tillman. A statistical parser for Czech. In *Proc. of ACL*, pages 505–512, Jun 1999.

Michael Collins, Philipp Koehn, and Ivona Cučerová. Clause restructuring for statistical machine translation. In *Proc. of ACL*, pages 531–540, Jun 2005.

J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics*, 43(5):1470–1480, Oct 1972.

Rene de la Briandias. File searching using variable length keys. In *Proceedings of the Western Joint Computer Conference*, pages 295—298, 1959.

Herve Dejean, Eric Gaussier, Cyril Goutte, and Kenji Yamada. Reducing parameter space for word alignment. In *Proc. of HLT-NAACL Workshop on Building and Using Parallel Texts*, pages 23–26, May 2003.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

Steve DeNeefe, Kevin Knight, and Hayward H. Chan. Interactively exploring a machine translation model. In *Proc. of ACL (Companion Vol.)*, pages 97–100, Jun 2005.

Steve DeNeefe, Kevin Knight, Wei Wang, and Daniel Marcu. What can syntax-based MT learn from phrase-based MT? In *Proc. of EMNLP-CoNLL*, pages 755–763, Jun 2007.

John DeNero and Dan Klein. Tailoring word alignments to syntactic machine translation. In *Proc. of ACL*, pages 17–24, Jun 2007.

John DeNero, Dan Gillick, James Zhang, and Dan Klein. Why generative phrase models underperform surface heuristics. In *Proc. of HLT-NAACL Workshop on Statistical Machine Translation*, pages 31–38, Jun 2006.

George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proc. of HLT*, 2002.

Bonnie J. Dorr, Pamela W. Jordan, and John W. Benoit. A survey of current paradigms in machine translation. In M. Zelkowitz, editor, *Advances in Computers*, volume 49, pages 1–68. Academic Press, 1999.

Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Apr 1998.

Christopher J. Dyer. The University of Maryland translation system for IWSLT 2007. In *Proc. of IWSLT*, Oct 2007.

Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2): 94–102, Feb 1970.

Matthias Eck, Stephan Vogel, and Alex Waibel. Language model adaptation for statistical machine translation based on information retrieval. In *Proc. of LREC*, May 2004.

Marcello Federico and Nicola Bertoldi. How many bits are needed to store probabilities for phrase-based translation? In *Proc. of NAACL Workshop on Statistical Machine Translation*, pages 94–101, Jun 2006.

George Foster and Roland Kuhn. Mixture model adaptation for SMT. In *Proc. of the Workshop on Statistical Machine Translation*, pages 128–135, Jun 2007.

George Foster, Roland Kuhn, and Howard Johnson. Phrasetable smoothing for statistical machine translation. In *Proc. of EMNLP*, pages 53–61, Jul 2006.

Heidi J. Fox. Phrasal cohesion and statistical machine translation. In *Proc. of EMNLP*, pages 304–311, Jul 2002.

Alexander Fraser and Daniel Marcu. Semi-supervised training for statistical word alignment. In *Proc. of ACL*, pages 769–776, Jun 2006.

Alexander Fraser and Daniel Marcu. Measuring word alignment quality for statistical machine translation. *Computational Linguistics*, 33(3), Sep 2007a.

Alexander Fraser and Daniel Marcu. Getting the structure right for word alignment: LEAF. In *Proc. of EMNLP-CoNLL*, pages 51–60, Jun 2007b.

Edward Fredken. Trie memory. *Communications of the ACM*, 3(9):490–499, Sep 1960.

William A. Gale and Kenneth W. Church. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(1):75–102, Mar 1993.

William A. Gale and Kenneth W. Church. Identifying word correspondences in parallel text. In *Proc. of Darpa Workshop on Speech and Natural Language*, pages 152–157, 1991.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In *Proc. of HLT-NAACL*, pages 273–280, May 2004.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. Scalable inference and training of context-rich syntactic translation models. In *Proc. of ACL*, pages 961–968, Jun 2006.

Ulrich Germann. Greedy decoding for statistical machine translation in almost linear time. In *Proc. of HLT-NAACL*, pages 72–79, May 2003.

Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. Fast decoding and optimal decoding for machine translation. In *Proc. of ACL-EACL*, Jul 2001.

Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. Fast and optimal decoding for machine translation. *Artificial Intelligence*, 154(1–2):127–143, Apr 2004.

Daniel Gildea. Dependencies vs. constituencies for tree-based alignment. In *Proc. of EMNLP*, pages 214–221, Jul 2004.

Sharon Goldwater and David McClosky. Improving statistical MT through morphological analysis. In *Proc. of HLT-EMNLP*, pages 676–683, Oct 2005.

Jonathan Graehl and Kevin Knight. Training tree transducers. In *Proc. of HLT-NAACL*, pages 105–112, May 2004.

Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2):100–107, Jul 1968.

Julia Hockenmaier, Aravind K. Joshi, and Ken A. Dill. Protein folding and chart parsing. In *Proc. of EMNLP*, pages 293–300, Jul 2006.

John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

Eduard Hovy, Margaret King, and Andrei Popescu-Belis. Principles of context-based machine translation evaluation. *Machine Translation*, 17(1):43–75, Mar 2002.

Liang Huang and David Chiang. Better *k*-best parsing. In *Proc. of IWPT*, pages 53–64, Oct 2005.

Liang Huang and David Chiang. Forest rescoring: Faster decoding with integrated language models. In *Proc. of ACL*, pages 144–151, Jun 2007.

John Hutchins. Towards a definition of example-based machine translation. In *Proceedings of the MT Summit Workshop on Example-Based Machine Translation*, pages 63–70, Sep 2005.

John Hutchins. Machine translation: A concise history. In Chan Sin Wai, editor, *Computer aided translation: theory and practice*. Chinese University of Hong Kong, 2007.

Rebecca Hwa, Philip Resnik, Amy Weinberg, Clara Cabezas, and Okan Kolak. Bootstrapping parsers via syntactic projection across parallel texts. *Natural Language Engineering*, 11(3): 311–325, Sep 2005.

Abraham Ittycheriah and Salim Roukos. A maximum entropy word aligner for Arabic-English machine translation. In *Proc. of HLT-EMNLP*, pages 89–96, Oct 2005.

Frederick Jelinek. A stack algorithm for faster sequential decoding of transmitted information,. Technical Report RC2441, IBM Research Center, Yorktown Heights, NY, 1969.

Frederick Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1998.

Howard Johnson, Joel Martin, George Foster, and Roland Kuhn. Improving translation quality by discarding most of the phrasetable. In *Proc. of EMNLP-CoNLL*, pages 967–975, Jun 2007.

Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin, 1997.

Aravind K. Joshi, K. Vijay-Shanker, and David Weir. The convergence of mildly context-sensitive grammar formalisms. In Peter Sells, Stuart Shieber, and Tom Wasow, editors, *Foundational Issues in Natural Language Processing*, chapter 2, pages 31–81. MIT Press, Cambridge, MA, 1991.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, 2000.

Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Annual Symposium on Combinatorial Pattern Matching*, LNCS, pages 181–192. Springer, Jul 2001.

Tadao Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Lab, 1965.

Katrin Kirchhoff and Mei Yang. Improved language modeling for statistical machine translation. In *Proc. of ACL Workshop on Building and Using Parallel Texts*, pages 125–128, Jun 2005.

Dan Klein and Christopher D. Manning. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn Treebank. In *Proc. of ACL-EACL*, pages 330–337, Jul 2001.

Kevin Knight. Automating knowledge acquisition for machine translation. *AI Magazine*, 18(4): 81–96, 1997.

Kevin Knight. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615, Dec 1999a.

Kevin Knight. A statistical MT tutorial workbook. Unpublished, 1999b.

Kevin Knight and Yaser Al-Onaizan. Translation with finite-state devices. In *Proc. of AMTA*, pages 421–437, Oct 1998.

Kevin Knight and Jonathan Graehl. An overview of probabilistic tree transducers for natural language processing. In *Proc. of CICLing*, LNCS, 2004.

Kevin Knight and Jonathan Graehl. An overview of probabilistic tree transducers for natural language processing. In *Proc. of CICLing*. Springer Verlag, 2005.

Kevin Knight and Daniel Marcu. Machine translation in the year 2004. In *Proc. of ICASSP*, Mar 2005.

Philipp Koehn. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In *Proc. of AMTA*, Sep 2004a.

Philipp Koehn. Statistical significance tests for machine translation evaluation. In *Proc. of EMNLP*, pages 388–395, Jul 2004b.

Philipp Koehn. *PHARAOH, a Beam Search Decoder for Phrase-Based Statistical Machine Translation Models, User Manual and Description for Version 1.2*, Jul 2004c.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proc. of MT Summit*, 2005.

Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2008. To appear.

Philipp Koehn and Hieu Hoang. Factored translation models. In *Proc. of EMNLP-CoNLL*, pages 868–876, 2007.

Philipp Koehn and Christof Monz. Shared task: Statistical machine translation between european languages. In *Proc. of ACL Workshop on Building and Using Parallel Texts*, pages 119–124, Jun 2005.

Philipp Koehn and Christof Monz. Manual and automatic evaluation of machine translation between european languages. In *Proc. of NAACL Workshop on Statistical Machine Translation*, pages 102–121, 2006.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proc. of HLT-NAACL*, pages 127–133, May 2003.

Philipp Koehn, Amittai Axelrod, Alexandra Birch Mayne, Chris Callison-Burch, Miles Osborne, and David Talbot. Edinburgh system description for the 2005 IWSLT speech translation evaluation. In *Proc. of IWSLT*, 2005.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proc. of ACL Demo and Poster Sessions*, pages 177–180, Jun 2007.

Alex Kulesza and Stuart M. Shieber. A learning approach to improving sentence-level MT evaluation. In *Proc. of TMI*, Oct 2004.

Shankar Kumar and William Byrne. Minimum bayes-risk decoding for statistical machine translation. In *Proc. of HLT-NAACL*, pages 169–176, May 2004.

Shankar Kumar, Yonggang Deng, and William Byrne. A weighted finite state transducer translation template model for statistical machine translation. *Natural Language Engineering*, 12(1): 35–75, Mar 2006.

K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4(1), 1990.

N. Jesper Larsson and Kunihiko Sadakane. Faster suffix sorting. Technical Report LU-CS-TR:99-214, Dept. of CS, Lund University, Sweden, 1999.

Lillian Lee. Measures of distributional similarity. In *Proc. of ACL*, pages 25–32, Jun 1999.

P. M. II Lewis and R. E. Stearns. Syntax-directed transductions. *Journal of the ACM*, 15:465–488, 1968.

Percy Liang, Alexandre Bouchard-Côté, Ben Taskar, and Dan Klein. An end-to-end discriminative approach to machine translation. In *Proc. of ACL-COLING*, pages 761–768, Jul 2006a.

Percy Liang, Ben Taskar, and Dan Klein. Alignment by agreement. In *Proc. of HLT-NAACL*, pages 104–111, Jun 2006b.

Chin-Yew Lin and Franz Josef Och. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In *Proc. of ACL*, pages 606–613, Jul 2004.

Lucian Lita, Monica Rogati, and Alon Lavie. BLANC: Learning evaluation metrics for MT. In *Proc. of HLT-EMNLP*, pages 740–747, Oct 2005.

Ding Liu and Daniel Gildea. Source-language features and maximum correlation training for machine translation evaluation. In *Proc. of HLT-NAACL*, pages 41–48, Apr 2007.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification with string kernels. *Journal of Machine Learning Research*, 2:419–444, Feb 2002.

Adam Lopez. Hierarchical phrase-based translation with suffix arrays. In *Proc. of EMNLP-CoNLL*, pages 976–985, Jun 2007.

Adam Lopez. Statistical machine translation. *ACM Computing Surveys*, 40(3), Sep 2008. In press.

Adam Lopez and Philip Resnik. Improved HMM alignment models for languages with scarce resources. In *Proc. of ACL Workshop on Building and Using Parallel Texts*, pages 83–86, Jun 2005.

Adam Lopez and Philip Resnik. Word-based alignment, phrase-based translation: What's the link? In *Proc. of AMTA*, pages 90–99, Aug 2006.

Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal of Computing*, 22(5):935–948, 1993.

Lidia Mangu, Eric Brill, and Andreas Stolcke. Finding consensus in speech recognition: Word error minimization and other applications of confusion networks. *Computer Speech and Language*, 14(4):373–400, 2000.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, May 1999.

Daniel Marcu and William Wong. A phrase-based, joint probability model for statistical machine translation. In *Proc. of EMNLP*, pages 133–139, Jul 2002.

Daniel Marcu, Wei Wang, Abdessamad Echihabi, and Kevin Knight. SPMT: Statistical machine translation with syntactified target language phrases. In *Proc. of EMNLP*, pages 44–52, Jul 2006.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19(2):314–330, Jun 1993.

Evgeny Matusov, Richard Zens, and Hermann Ney. Symmetric word alignments for statistical machine translation. In *Proc. of COLING*, pages 219–225, Jul 2004.

Paul McNamee and James Mayfield. Translation of multiword expressions using parallel suffix arrays. In *Proc. of AMTA*, pages 100–109, Aug 2006.

I. Dan Melamed. Automatic construction of clean broad-coverage translation lexicons. In *Proc. of AMTA*, 1996.

I. Dan Melamed. Manual annotation of translational equivalence: The blinker project. Technical Report 98-07, University of Pennsylvania Institute for Research in Cognitive Science, 1998.

I. Dan Melamed. Models of translational equivalence among words. *Computational Linguistics*, 26(2):221–249, Jun 2000.

I. Dan Melamed. Multitext grammars and synchronous parsers. In *Proc. of HLT-NAACL*, pages 79–86, May 2003.

I. Dan Melamed. Statistical machine translation by parsing. In *Proc. of ACL*, pages 654–661, Jul 2004a.

I. Dan Melamed. Algorithms for syntax-aware statistical machine translation. In *Proc. of TMI*, 2004b.

I. Dan Melamed, Ryan Green, and Joseph P. Turian. Precision and recall of machine translation. In *Proc. of HLT-NAACL (Companion Vol.)*, pages 61–63, May 2003.

I. Dan Melamed, Giorgio Satta, and Benjamin Wellington. Generalized multitext grammars. In *Proc. of ACL*, pages 662–669, Jul 2004.

Bernard Merialdo. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–172, 1994.

Rada Mihalcea and Ted Pedersen. An evaluation exercise for word alignment. In *Proc. of HLT-NAACL Workshop on Building and Using Parallel Texts*, pages 1–10, May 2003.

Einat Minkov, Kristina Toutanova, and Hisami Suzuki. Generating complex morphology for machine translation. In *Proc. of ACL*, pages 128–135, Jun 2007.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

Robert C. Moore. Improving IBM word-alignment model 1. In *Proc. of ACL*, pages 519–526, Jul 2004.

Robert C. Moore. A discriminative framework for bilingual word alignment. In *Proc. of HLT-EMNLP*, pages 81–88, Oct 2005a.

Robert C. Moore. Association-based bilingual word alignment. In *Proc. of ACL Workshop on Building and Using Parallel Texts*, pages 1–8, Jun 2005b.

Makoto Nagao. A framework of a mechanical translation between Japanese and English by analogy principle. In A. Elithorn and R. Banerji, editors, *Artificial and Human Intelligence*, pages 173–180. Elsevier, 1984.

Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33 (1):31–88, Mar 2001.

Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39 (1), Apr 2007.

Sonja Nießen and Hermann Ney. Statistical machine translation with scarce resources using morpho-syntactic information. *Computational Linguistics*, 30(2):182–204, Jun 2004.

Sonja Nießen, Stephan Vogel, Hermann Ney, and Christoph Tillman. A DP based search algorithm for statistical machine translation. In *Proc. of ACL-COLING*, pages 960–967, Aug 1998.

Douglas W. Oard and Franz Josef Och. Rapid-response machine translation for unexpected languages. In *Proc. of MT Summit IX*, Sept 2003.

Douglas W. Oard, David Doermann, Bonnie Dorr, Daqing He, Philip Resnik, Amy Weinberg, William Byrne, Sanjeev Khudanpur, David Yarowsky, Anton Leuski, Philipp Koehn, and Kevin Knight. Desperately seeking Cebuano. In *Proc. of HLT-NAACL (Companion Vol.)*, pages 76–78, May 2003.

Franz Josef Och. An efficient method for determining bilingual word classes. In *Proc. of EACL*, pages 71–76, Jun 1999.

Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proc. of ACL*, Jul 2003.

Franz Josef Och. Statistical machine translation: The fabulous present and future. In *Proc. of ACL Workshop on Building and Using Parallel Texts*, Jun 2005. Invited talk.

Franz Josef Och and Hermann Ney. A comparison of alignment models for statistical machine translation. In *Proc. of COLING*, pages 1086–1090, Jul 2000.

Franz Josef Och and Hermann Ney. Statistical multi-source translation. In *Proc. of MT Summit*, Sep 2001.

Franz Josef Och and Hermann Ney. Discriminative training and maximum entropy models for machine translation. In *Proc. of ACL*, pages 156–163, Jul 2002.

Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, Mar 2003.

Franz Josef Och and Hermann Ney. The alignment template approach to machine translation. *Computational Linguistics*, 30(4):417–449, Jun 2004.

Franz Josef Och, Christoph Tillman, and Hermann Ney. Improved alignment models for statistical machine translation. In *Proc. of EMNLP-VLC*, pages 20–28, Jun 1999.

Franz Josef Och, Nicola Ueffing, and Hermann Ney. An efficient A* search algorithm for statistical machine translation. In *Proc. of ACL Workshop on Data-Driven Methods in Machine Translation*, pages 55–62, Jul 2001.

Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. Final report of johns hopkins 2003 summer workshop on syntax for statistical machine translation. Technical report, Johns Hopkins University, February 2004a.

Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. A smorgasbord of features for statistical machine translation. In *Proc. of HLT-NAACL*, pages 161–168, May 2004b.

Marian Olteanu, Chris Davis, Ionut Volosen, and Dan Moldovan. Phramer - an open source statistical phrase-based translator. In *Proc. of HLT-NAACL Workshop on Statistical Machine Translation*, pages 146–149, 2006.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*, pages 311–318, Jul 2002.

Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional clustering of English words. In *Proc. of ACL*, pages 183–190, Jun 1993.

Maja Popovic, Adrià de Gispert, Deepa Gupta, Patrik Lambert, Hermann Ney, José B. Mariño, Marcello Federico, and Rafael Banchs. Morpho-syntactic information for automatic error analysis of statistical machine translation output. In *Proc. of NAACL Workshop on Statistical Machine Translation*, pages 1–6, Jun 2006.

Simon J. Puglisi, W. F. Smyth, and Andrew H. Turpin. A taxonomy of suffix array construction algorithms. *ACM Computing Surveys*, 39(2), Jun 2007.

Chris Quirk and Arul Menezes. Do we need phrases? Challenging the conventional wisdom in statistical machine translation. In *Proc. of HLT-NAACL*, pages 8–16, Jun 2006.

Chris Quirk, Arul Menezes, and Colin Cherry. Dependency treelet translation: Syntactically informed phrasal SMT. In *Proc. of ACL*, pages 271–279, Jun 2005.

Mohammad Sohel Rahman, Costas S. Iliopoulos, Inbok Lee, Manal Mohamed, and William F. Smyth. Finding patterns with variable length gaps or don't cares. In *Proc. of COCOON*, Aug 2006.

Adwait Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania, 1998.

Philip Resnik and Noah A. Smith. The web as parallel corpus. *Computational Linguistics*, 29(3): 349–380, Sep 2003.

Philip Resnik, Mari Broman Olsen, and Mona Diab. Creating a parallel corpus from the "book of 2000 tongues". In *Proceedings of the Text Encoding Initiative 10th Anniversary User Conference (TEI-10)*, 1997.

Roni Rosenfeld. Two decades of statistical language modeling: where do we go from here? *Proc. of IEEE*, 88(8):1270–1278, Aug 2000.

Antti-Veikko Rosti, Necip Fazil Ayan, Bing Xiang, Spyros Matsoukas, Richard Schwartz, and Bonnie Dorr. Combining outputs from multiple machine translation systems. In *Proc. of HLT-NAACL*, pages 228–235, Apr 2007.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition, 2003.

Grazia Russo-Lassner, Jimmy Lin, and Philip Resnik. A paraphrase-based approach to machine translation evaluation. Technical Report LAMP-TR-125, University of Maryland, Aug 2005.

Peter Sanders and Frederik Transier. Intersection in integer inverted indices. In *Proc. of ALENEX*, pages 71–83, Jan 2007.

Satoshi Sato and Makoto Nagao. Toward memory-based translation. In *Proc. of COLING*, pages 247–252, Aug 1990.

Charles Schafer and Elliott Franco Drabek. Models for inuktitut-english word alignment. In *Proc. of ACL Workshop on Building and Using Parallel Texts*, pages 79–82, Jun 2005.

Libin Shen, Anoop Sarkar, and Franz Josef Och. Discriminative reranking for machine translation. In *Proc. of HLT-NAACL*, pages 177–184, May 2004.

Stuart M. Shieber and Yves Schabes. Synchronous tree-adjoining grammars. In *Proc. of COLING*, pages 253–258, 1990.

Michel Simard, Nicola Cancedda, Bruno Cavestro, Marc Dymetman, Eric Gaussier, Cyril Goutte, Kenji Yamada, Philippe Langlais, and Arne Mauser. Translating with non-contiguous phrases. In *Proc. of HLT-EMNLP*, pages 755–762, Oct 2005.

Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 2nd edition, Feb 2005.

David A. Smith and Noah Smith. Bilingual parsing with factored estimation: Using English to parse Korean. In *Proc. of EMNLP*, pages 49–56, Jul 2004.

Noah A. Smith. From words to corpora: Recognizing translation. In *Proc. of EMNLP*, pages 95–102, Jul 2002.

Noah A. Smith. *Novel Estimation Methods for Unsupervised Discovery of Latent Structure in Natural Language Text*. PhD thesis, Johns Hopkins University, Oct 2006.

Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proc. of AMTA*, pages 223–231, Aug 2006.

Harold Somers. An overview of EBMT. In Michael Carl and Andy Way, editors, *Recent Advances in Example-Based Machine Translation*, chapter 4, pages 115–153. Kluwer, 2003.

Michael Subotin. Conditional random fields for statistical machine translation. Generals paper (draft), Jan 2008.

David Talbot and Miles Osborne. Randomised language modelling for statistical machine translation. In *Proc. of ACL*, pages 512–519, Jun 2007a.

David Talbot and Miles Osborne. Smoothed bloom filter language models: Tera-scale LMs on the cheap. In *Proc. of ACL*, pages 468–476, Jun 2007b.

Ben Taskar. *Learning Structured Prediction Models: A Large-Margin Approach*. PhD thesis, Stanford University, Dec 2004.

Ben Taskar, Simon Lacoste-Julien, and Dan Klein. A discriminative matching approach to word alignment. In *Proc. of HLT-EMNLP*, pages 73–80, Oct 2005.

Christoph Tillman. A unigram orientation model for statistical machine translation. In *Proc. of HLT-NAACL: Short Papers*, pages 101–104, May 2004.

Christoph Tillman and Hermann Ney. Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational Linguistics*, 29(1):98–133, Mar 2003.

Christoph Tillmann and Tong Zhang. A discriminative global training algorithm for statistical MT. In *Proc. of ACL-COLING*, pages 721–728, Jul 2006.

Christoph Tillmann, Stephen Vogel, Hermann Ney, and Alex Zubiaga. A DP-based search using monotone alignments in statistical translation. In *Proc. of ACL-EACL*, pages 289–296, 1997.

Kristina Toutanova, H. Tolga Ilhan, and Christopher D. Manning. Extensions to HMM-based statistical word alignment models. In *Proc. of EMNLP*, pages 87–94, Jul 2002.

Joseph P. Turian, Luke Shen, and I. Dan Melamed. Evaluation of machine translation and its evaluation. In *Proc. of MT Summit IX*, Sep 2003.

Nicola Ueffing and Hermann Ney. Word-level confidence estimation for machine translation using phrase-based translation models. In *Proc. of HLT-EMNLP*, pages 763–770, Oct 2005.

Nicola Ueffing, Franz Josef Och, and Hermann Ney. Generation of word graphs in statistical machine translation. In *Proc. of EMNLP*, pages 156–163, Jul 2002.

Nicola Ueffing, Gholamreza Haffari, and Anoop Sarkar. Transductive learning for statistical machine translation. In *Proc. of ACL*, pages 25–32, Jun 2007.

Esko Ukkonen. On-line construction of suffix-trees. *Algorithmica*, 14(3):249–260, Sep 1995.

Peter van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10(2):99–127, 1977.

Ashish Venugopal, Andreas Zollmann, and Alex Waibel. Training and evaluating error minimization rules for statistical machine translation. In *Proc. of ACL Workshop on Building and Using Parallel Texts*, pages 208–215, Jun 2005.

Ashish Venugopal, Andreas Zollmann, and Stephan Vogel. An efficient two-pass approach to synchronous-CFG driven statistical MT. In *Proc. of HLT-NAACL*, 2007.

K. Vijay-Shanker, David Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *Proc. of ACL*, pages 104–111, 1987.

Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

Stephan Vogel, Hermann Ney, and Christoph Tillman. HMM-based word alignment in statistical machine translation. In *Proc. of COLING*, pages 836–841, Aug 1996.

Chao Wang, Michael Collins, and Philipp Koehn. Chinese syntactic reordering for statistical machine translation. In *Proc. of EMNLP-CoNLL*, pages 737–745, Jun 2007.

Jianqiang Wang. *Matching Meaning for Cross-Language Information Retrieval*. PhD thesis, University of Maryland, 2005.

Ye-Yi Wang and Alex Waibel. Decoding algorithm in statistical machine translation. In *Proc. of ACL-EACL*, pages 366–372, Jul 1997.

Taro Watanabe and Eiichiro Sumita. Bidirectional decoding for statistical machine translation. In *Proc. of COLING*, pages 1079–1085, Aug 2002.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. Online large-margin training for statistical machine translation. In *Proc. of EMNLP-CoNLL*, pages 764–773, Jun 2007.

Warren Weaver. Translation. In William N. Locke and A. Donald Booth, editors, *Machine Translation of Languages: Fourteen Essays*, chapter 1, pages 15–23. MIT Press, 1955. Reprint of 1949 memorandum.

Benjamin Wellington, Joseph Turian, Chris Pike, and I. Dan Melamed. Scalable purely-discriminative training for word and tree transducers. In *Proc. of AMTA*, pages 251–260, Aug 2006a.

Benjamin Wellington, Sonjia Waxmonsky, and I. Dan Melamed. Empirical lower bounds on the complexity of translational equivalence. In *Proc. of ACL-COLING*, pages 977–984, Jun 2006b.

John S. White, Theresa O'Connell, and Francis O'Mara. The ARPA MT evaluation methodologies: Evolution, lessons, and future approaches. In *Proc. of AMTA*, 1994.

Dekai Wu. Stochastic inversion transduction grammars, with application to segmentation, bracketing, and alignment of parallel corpora. In *Proc. of IJCAI*, pages 1328–1335, Aug 1995a.

Dekai Wu. Grammarless extraction of phrasal translation examples from parallel texts. In *Proc. of TMI*, pages 354–372, Jul 1995b.

Dekai Wu. A polynomial-time algorithm for statistical machine translation. In *Proc. of ACL*, pages 152–158, Jun 1996.

Dekai Wu. MT model space: statistical versus compositional versus example-based. *Machine Translation*, 19(3–4):213–227, Dec 2005.

Dekai Wu and Hongsing Wong. Machine translation with a stochastic grammatical channel. In *Proc. of ACL-COLING*, pages 1408–1415, Aug 1998.

Deyi Xiong, Qun Liu, and Shouxun Lin. Maximum entropy based phrase reordering model for statistical machine translation. In *Proc. of ACL-COLING*, pages 521–528, Jul 2006.

Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proc. of ACL-EACL*, 2001.

Kenji Yamada and Kevin Knight. A decoder for syntax-based statistical MT. In *Proc. of ACL*, pages 303–310, Jul 2002.

Mikio Yamamoto and Kenneth Church. Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. *Computational Linguistics*, 27(1):1–30, Mar 2001.

David Yarowsky and Grace Ngai. Inducing multilingual POS taggers and NP bracketers via robust projection across aligned corpora. In *Proc. of NAACL*, pages 200–207, Jun 2001.

David Yarowsky, Grace Ngai, and Richard Wicentowski. Inducing mulilingual text analysis tools via robust projection across aligned corpora. In *Proc. of HLT*, pages 109–116, 2001.

Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.

Richard Zens and Hermann Ney. A comparative study on reordering constraints in statistical machine translation. In *Proc. of ACL*, pages 144–151, Jul 2003.

Richard Zens and Hermann Ney. Improvements in phrase-based statistical machine translation. In *Proc. of HLT-NAACL*, pages 257–264, May 2004.

Richard Zens and Hermann Ney. Efficient phrase-table representation for machine translation with applications to online MT and speech translation. In *Proc. of HLT-NAACL*, 2007.

Hao Zhang and Daniel Gildea. Stochastic lexicalized inversion transduction grammar for alignment. In *Proc. of ACL*, pages 475–482, Jun 2005.

Hao Zhang, Liang Huang, Daniel Gildea, and Kevin Knight. Synchronous binarization for machine translation. In *Proc. of HLT-NAACL*, pages 256–263, Jun 2006a.

Ying Zhang and Stephan Vogel. An efficient phrase-to-phrase alignment model for arbitrarily long phrase and large corpora. In *Proc. of EAMT*, May 2005.

Ying Zhang, Almut Silja Hildebrand, and Stephan Vogel. Distributed language modeling for N-best list re-ranking. In *Proc. of EMNLP*, pages 216–223, Jul 2006b.

Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), Jul 2006.