STONE SOUP TRANSLATION:
THE LINKED AUTOMATA MODEL

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the Graduate

School of the Ohio State University

By

Paul C. Davis, M.A.

* * * * *

The Ohio State University
2002

Dissertation Commitee:

Chris Brew, Adviser

Walt Detmar Meurers

Robert T. Kasper

Erhard Hinrichs

Approved by

____

Adviser
Department of Linguistics

ABSTRACT

The automated translation of one natural language to another, known as *machine translation* (MT), typically requires successful modeling of the grammars of the languages and the relationship between them. Rather than hand-coding these grammars and relationships, some machine translation efforts employ data-driven methods, where the goal is to learn from a large amount of training examples of accurate translations. One such data-driven approach is statistical MT, where language and alignment models are automatically induced from parallel corpora. This work has also been extended to probabilistic finite-state approaches, most often via transducers.

This dissertation introduces and begins an investigation of an MT model consisting of a novel combination finite-state devices. The model proposed is more flexible than transducer models, giving increased ability to handle word order differences between languages, as well as crossing and discontinuous alignments between words. The linked automata MT model consists of a source language automaton, a target language automaton, and an alignment table—a function which probabilistically links sequences of source and target language transitions. It is this augmentation to the finite-state base which gives the linked automata model its flexibility.

The dissertation describes the linked automata model from the ground up, beginning with a description of some of the relevant MT history and empirical MT literature, and the preparatory steps for building the model, including a detailed discussion of word alignment and the introduction of a new technique for word alignment evaluation. Discussion then centers on the description of the model and its use of probabilities, including algorithms for its construction from word-aligned bitexts and for the translation process. The focus next moves to expanding the linked automata approach, first through generalization and techniques for extracting partial results, and then by increasing the coverage, both in terms of using additional linguistic information and using more complex alignments. The dissertation presents preliminary results for a test corpus of English to Spanish translations, and suggests ways in which the model can be further expanded as the foundation of a more powerful MT system.

ACKNOWLEDGMENTS

I have heard people compare writing a dissertation to running a long-distance race. It's a grueling process, testing both your will and your endurance, and there always seems to be one last hill to climb. You have to run the race yourself, but there are friends and family who come out to shout an encouraging word, and a whole team of people whose job it is to support you—be it those who lend a hand on the race day, or those who have coached you during training. And of course there are the other runners who have gone before, run with you, or will compete later, ready to offer advice and to commiserate.

Sometimes you don't thank everyone who helped—perhaps you don't even know who it was that clapped their hands at the moment when you might otherwise have quit. So it is with this dissertation. I won't be able to thank everyone who played a part, and this is especially true for those who helped me prior to the dissertation-writing phase, such as friends and teachers from the University of Wisconsin, or everyone who has had an impact on my life at Ohio State. For those I neglect to name, please know that your help mattered.

So, let me start with my family. A really great part of returning to Ohio was the opportunity to spend a lot of time with family and friends from Cleveland. In particular I want to thank my parents, Jay and Jane Davis, and my sister and brother, Maria and Josh, for their support.

Many others also helped. Philip Resnik and his collaborators at the University of Maryland graciously made available the parallel corpora from which the dissertation models were built. Franz Joseph Och made the Giza^{++} software, which I used to create word alignments, building on the work done by the researchers who had selflessly made Giza and the rest of the MT tools in the Egypt toolkit. Dan Melamed provided advice on both word alignment tools and word alignment evaluation, and Hiyan Alshawi answered my questions about MT evaluation metrics. Additionally, a number of anonymous reviewers provided valuable comments on presentations of portions of this work. We must also follow the money trail—this work would have been much more difficult without the financial support of a four-year University Partnerships in Research Grant from Motorola; Bob Kasper and Craige Roberts deserve thanks for the original grant, as do Mike McLaughlin and Harry Bliss of Motorola. Portions of the dissertation work were also funded by a summer grant from the OSU Center for Cognitive Science, made possible by the efforts of my adviser, Chris Brew, and Neelam Soundarajan, of Computer and Information Science.

I'd like to thank a number of friends and colleagues, at OSU and elsewhere, who played some part in the dissertation process, from providing LaTeX style files to keeping me company while I typed—this list is far from complete: Laurie Maynell, Ed Hubbard, Jennifer Venditti, Svetlana Godjevac, Martin Jansche, Vanessa Metcalf, Kordula DeKuthy, Nathan Vaillette, Wes Collins, Neal Whitman, Markus Dickinson, James Levy, and, in particular, Hope Dawson, who generously took on the task of proofreading the entire manuscript; the Clippers discussion group; the participants of TMI 2002; and current and former members of the NLP group at Motorola Labs, especially Harry Bliss, Dale Russell, Will Thompson, Tom Hayosh, and Guido Minnen.

Last, I would like to thank those people who had the most direct impact on the dissertation, the members of the dissertation committee: Chris Brew, Erhard Hinrichs, Bob Kasper, and Detmar Meurers. Detmar came last to the work—offering new perspectives, fresh insights, and lots of encouragement. I feel lucky to have had the chance to both work with him and call him a friend. Erhard deserves special thanks, serving as both a mentor at OSU and in Tübingen, and having the special ability to pinpoint exactly where a key comment needs to be made and how work is likely to be perceived; his comments always constructive and his advice invariably wise—he's a great person to have in your corner. I had two advisers at OSU: Bob for the early years and Chris for the later ones. Both taught courses which had direct impact on this work. Bob and Erhard co-taught a great seminar on finite-state methods which kicked off my interest in automata in NLP. Chris opened the world of statistical NLP to me, sparking a whole new range of interests; as the adviser in my final years at OSU, he had the most influence on the linked automata work, for which I am very grateful. The synergy of these sorts of influences led me to the ideas which developed into this dissertation. Although Bob and Chris were my advisers at different times—they share many traits: They are both incredible researchers and teachers. More importantly, they are extremely good people, who always take time out for their students, regardless of what else they have to do. I cannot imagine having had better advisers.

VITA

1966	Born in Cleveland, Ohio
1988	B.S. in Business-Economics, Miami University
1996	M.A. in Linguistics, University of Wisconsin-Madison

PUBLICATIONS

Davis, Paul C. and Chris Brew. 2002. Stone soup translation. In *Proceedings of the 9th Conference on Theoretical and Methodological Issues in Machine Translation (TMI-02)*, 31–41, Keihanna, Japan.

Davis, Paul C. 2000. Presupposition resolution with discourse information structures. In *Varia*. Ohio State University Working Papers in Linguistics, ed. by Jennifer S. Muller, Tsan Huang, and Craige Roberts, vol. 54: 25–58.

Kasper, Robert T., Paul C. Davis, and Craige Roberts. 1999. An Integrated Approach to Reference and Presupposition Resolution. In *Proceedings of the ACL'99 Workshop on the Relationship Between Discourse/Dialogue Structure and Reference*, 1–10, College Park, Maryland.

Kasper, Robert T., Mike Calcagno, and Paul C. Davis. 1998. Know When to Hold 'Em: Shuffling Deterministically in a Parser for Nonconcatenative Grammars. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pp. 663–669, Montreal, Canada.

FIELDS OF STUDY

Major field: Linguistics

Specialization: Computational Linguistics

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

> Perhaps I risk hyperbole but it seems to me that computer translation ought to rank as one of the noblest of human undertakings, as in its broadest aspects it attempts to understand, systematize and predict not just one aspect of life but all human understanding itself. Measured against such a goal, even its shortcomings have a great deal to tell us. Perhaps one day it will succeed in such a quest and lead us all out of the jungle of language and into some better place, although for all the reasons I have mentioned this appears somewhat unlikely. (Gross 1992:123)

The automated translation of one language to another is known as *machine translation*. Quality machine translation (MT) remains one of the quintessential goals of computational linguistics, as a supremely challenging task bringing together theoretical linguistics and computer science. MT approaches garner a variety of techniques from both fields, and span differing domains in terms of pairs and types of languages handled, the amount of language covered, as well as the breadth of world knowledge encoded in the system. Like most niches of Natural Language Processing (NLP) research, MT systems range from practical programs used in real-world applications, to theoretically newborn systems which explore new frontiers. Everpresent in these efforts is the knowledge that in a task that bilingual humans accomplish with great fluency, MT systems have never yielded high-quality translations in unlimited domains. This state-of-affairs should not be surprising, given the difficulty linguists face in modeling one language; machine translation requires the successful modeling of two languages, as well as modeling a translation relation between them.

In this thesis a new model for MT is introduced. The *linked automata* model brings together very simple technologies, technologies which in isolation should not be adequate for translation, to explore how far they can be pushed as a translation system. Previous finite-state translation approaches have centered on transducers, well-understood but limited finite-state machines, which tie together two languages in a single device. Transducers can perform well in very restricted language domains, but do not have the ordering flexibility to directly model the word-ordering differences often found between languages.

The linked automata model breaks the strict coupling of languages found in transducers into two separate models, using one automaton for each language, and an alignment function to link words in the languages. The goal for this augmented probabilistic finite-state model is to more directly encode the correspondence between sequences of words in two languages than can be accomplished with transducers alone. As such, the model aims to be a more appropriate translation approach which still utilizes many of the advantages of finite state devices. This dissertation thoroughly explores the linked automata MT system from the ground up, detailing its motivation, construction, use, performance, and the necessary extensions and heuristics so that it can generalize beyond its modest architecture, as a beginning investigation of its properties as an automatic translation tool. The model and its exploration should be viewed as a skeleton around which more ambitious translation models can be built. The linked automata model is an empirical one, automatically constructed exclusively from parallel texts, which neither fully encodes syntactic nor semantic structure, much less knowledge from context or culture. In this work I hope to show that it offers a promising base for future development.

1.1 Thesis Overview

This thesis is organized into seven chapters. In the remainder of this chapter, I give a brief history of machine translation, in section 1.2, and discuss some of the major difficulties machine translation efforts are likely to face, in section 1.3. In Chapter 2, the MT literature most relevant to the linked automata model is reviewed, with special emphasis on statistical machine translation and finite-state machine translation. Chapter 3 details all the preliminaries to the model, including the preparation of texts used in the experiments. An essential assumption of the approach is that training texts have been aligned. Chapter 3 details several word alignment approaches, and introduces a word alignment evaluation method for comparing the results to a hand-aligned gold standard. In Chapter 4, the linked automata model is described in detail. Beginning with the motivation for the model, I describe the two language models and the alignment model, and detail the use of probabilities. The preliminary algorithms for construction and translation are also given, and finally the model is compared to some of the other MT approaches described in Chapter 2. In Chapter 5, the evaluation of the system begins with a discussion of evaluation for MT in general and identification of the appropriate evaluation methods for the linked automata model. The model is then evaluated on the two data sets built in Chapter 3. In its basic form, the linked automata system can only process sentences on which it was trained. In Chapter 6, the model is extended with several generalization techniques, the most important of which is merging in the automata. In addition to describing merging and its ramifications, I give several other increased-coverage heuristics, and reevaluate the system. In Chapter 7, the final chapter of the dissertation, I describe some of the research directions the project may take. There are two main threads here, one being using more linguistic information, and the other being extending the model's coverage. In addition to identifying certain of these areas likely to prove fruitful, two final experiments are presented: the use of part-of-speech tags, and the use of discontinuous word alignments. Lastly, at the end of Chapter 7, I provide an assessment of where the linked automata model stands and its future in the world of MT research.

1.2 A Brief History of Machine Translation

Before embarking on a description of a new approach to machine translation, a brief survey of the history of MT may be useful. This mini-history is not intended to be exhaustive, nor to give details on specific algorithms, but rather to give some of the highlights (and lowlights) of the field over the last 50 years, and to point interested readers to further resources.[1]

Some researchers point as far back as the 1600s, to Leibniz, Descartes, and others, for proposals of numerical codes to relate the words of different languages to each other. None of these proposals were for the construction of actual machines,

[1]For an excellent and thorough history of the early years of MT, see Hutchins (1986), and for first-hand accounts from some of the early MT pioneers, see Hutchins (2000a).

however. Such ideas were not made explicit until the twentieth century, when the precursors to the computer were being invented (Hutchins 1986). There is some debate as to who first proposed mechanical translation between natural languages, but most histories point to conversations between Warren Weaver and Andrew Booth, in the late 1940s (see Arnold *et al.* (1994) and Booth & Locke (1955)), and in particular to a memorandum sent in 1949 by Warren Weaver to some 200 scientists who he thought might be interested in the notion of machine translation (Hutchins 1986).

The Weaver memorandum, sent on July 15th, 1949 (see Weaver (1955) for a reprinted version), focused on long-term strategies for MT, made possible by the invention of the computer. Weaver was well aware of electronic computing capabilities due to his work with computers during Word War II. His memo cited four key issues for the translation of natural language, and gave preliminary ideas for how to work with them: 1) The problem of multiple meaning, 2) the (at least occasional) logical nature of language 3) the use of insights from cryptography (Weaver followed the work of Claude Shannon, and the two would later co-author a book), and 4) the possibility of linguistic universals. Weaver thought that context could be used to disambiguate meanings (presaging many of the statistical methods for MT to come later, see Chapter 2), and that computers ought to be able to process those elements in language which were written in a logical style (e.g., scientific articles, as opposed to literary works). Weaver employed an example of relating translation to cryptography, in viewing a Chinese text as just English in a Chinese code (an idea to be implemented in Brown *et al.* (1993)). Last, Weaver saw linguistic universals as one of the most promising traits: "the most promising approach of all is one . . . that goes so deeply into the structure of languages as to come down to the level where they exhibit common traits" (Weaver 1955:23).

Weaver's memorandum generated widespread interest in the potential of automatic translation (Booth & Locke 1955), and in the years that followed many research projects were initiated, especially in the United States, with important work also emanating from England (particularly that of Booth) and the Soviet Union. The next decade was a time of fruitful research. Yehoshua Bar-Hillel became the first full-time paid MT researcher (appointed to a position at MIT in 1951), and he organized the first MT conference in 1952 (Booth & Locke 1955). Thus began what might be viewed as the first golden-age of MT research.

Some work during this decade focused naturally on some of (what would be viewed today as) the more mundane I/O and hardware issues of the time: Storage and speed issues were a constant reality of the early computing era. But work also focused on many of the core issues that remain today: how best to represent lexicons and the grammars of languages, how to map between them (directly or via an interlingua), and how to deal with ambiguities and disparate syntactic structures between languages. The Cold War led to much interest in MT between Russian and English, and a modest MT experiment was given at Georgetown in 1954. The Georgetown-IBM experiment consisted of translating a small set of highly constrained Russian sentences into English. The program applied word lookup (it had a vocabulary of 250 words), some morphological analysis, and used six grammatical rules (see Hutchins & Somers (1992) and Vasconcellos (2000)). The experiment was successful enough that money began to flow to MT in earnest (Hutchins 1986).

At the same time, some researchers (especially Bar-Hillel) were beginning to become skeptical about the possibility of high-quality automatic translation. In 1951 Bar-Hillel already thought it obvious that automatic translation was only possible at the expense of accuracy, and he suggested that MT might work best when assisted by humans, a mixed MT; a human partner might be placed on both ends of the translation process (as a sort of pre- and post-editor) (Hutchins 2000c). Bar-Hillel's criticisms would grow by 1959 to a complete loss of confidence in the prospect of high-quality MT (Hutchins 1986). Meanwhile, even at the same university, while acknowledging MT's shortcomings, many researchers continued on enthusiastically. Victor Yngve led projects at MIT. In 1955 he made a program which searched text left-to-right, looking for longest matches of word and phrase classes which had been earlier defined. The approach appeared promising, but it failed to account for different word ordering in the language being translated to (Yngve 2000). During the later 1950s, several more MT conferences were held, and Yngve (along with William Locke) started the first machine translation journal, *MT*, which would continue to publish until 1970 (Yngve 2000).

Notwithstanding the enthusiasm some held, the criticisms of Bar-Hillel and others would eventually spell the end of much of the funding for MT research. According to Hutchins (1986), Bar-Hillel's 1959 report stated that not only was high-quality automatic machine translation unattainable in the present, but altogether an impossibility! Bar-Hillel's criticism essentially rested on the utter impossibility of a computer being able to have enough world-knowledge to resolve ambiguities unresolvable from context alone, citing the example:

(1) *The box was in the pen.*

in the context of:

Little John was looking for his toy box. Finally, he found it. The box was in the pen. John was very happy.

Bar-Hillel claimed that the ambiguity between the two meanings of *pen* could only be resolved by understanding the relative sizes of objects (i.e., a box could fit into playpen but not into an ink pen). Hutchins (1986) views some of Bar-Hillel's criticisms as too strong, given the field's infancy and Bar-Hillel's own withdrawal from MT work, but concedes that "Bar-Hillel's case ... convinced many not involved in MT research that MT as such was doomed to failure, and it has continued to represent a challenge and point of departure for arguments about MT to the present day" (Hutchins 1986:155).

The growing dissent within the field led funding agencies to question their priorities. In 1963, the CIA withdrew funding for the Georgetown MT group, and in 1964, a committee known as ALPAC (Automatic Language Processing Advisory Committee) was formed by the NSF in order to investigate the state of MT research in the U.S., and to determine if funding should be continued (Hutchins 2000b). The conclusions of the committee, published in 1966 (Pierce *et al.* 1966), seemed predestined. The committee consisted of seven members, of which only two had worked on MT, David Hays and Anthony Oettinger. Neither believed in a continuation of MT research. As Oettinger (2000:83) states, "I knew that I was probably going to end up by taking my own research field 'down the drain' but I already had the firm conviction that MT was not going anywhere and that it made no sense to perpetuate a fraudulent belief that something might be achieved." To make matters worse, the two linguists on the committee, Eric Hamp and Charles Hockett, and the one AI researcher, Alan Perlis, were all, in one form or another, skeptical of mathematical and computational linguistics (Hutchins 1986).

The ALPAC report recommended that funding for MT research cease, but that funding for non-MT computational linguistics as well as linguistics in general should be continued. The committee found that MT quality was too low to be useful, and that there was indeed no need for any additional non-human translators: "The supply of translators greatly exceeds the demand .... The Committee is puzzled by a rationale for spending substantial sums of money on the mechanization of a small and already economically depressed industry" (Pierce *et al.* 1966:11-12). Arnold *et al.* (1994) find some of these views particularly absurd, given the need for translation from Russian during the Cold War, and the fact that at the time of the report at least three MT systems were in use (at the Wright Patterson Air Force base, at Oak Ridge Laboratory, and at the EURATOM Centre at Ispra, Italy). These systems provided translations which would need to be post-edited, but still saved translators time; but the committee seemed focused only on fully-automatic, high quality translation.

Funding in the United States for MT effectively ceased after the report's publication, and the funding death spread to Europe. In the USSR the effect on MT work was equally devastating (Mel'čuk 2000). There were some notable exceptions, such as the TAUM group in Canada, who would develop the METEO system, which has been in daily use since 1977 at the Canadian Meteorological Center in Montreal (Arnold *et al.* 1994). Research in MT became somewhat stigmatized and *The Association for Machine Translation and Computational Linguistics*, first formed in 1962, became know as *The Association for Computational Linguistics* in 1968 (Hutchins 1986).

In the late 1970s, MT began to make a bit of a comeback (Arnold *et al.* 1994). There was a significant increase of MT activity in Japan, and in Europe an indirect descendent of the Georgetown system, known as SYSTRAN, was bought by the Commission of the European Communities, for translation from English to French. SYSTRAN today remains arguably the most successful commercial MT company. MT systems also developed along a number of different theoretical lines, from those which were direct (source language to target language) systems, to those which went by some means of transfer, such as a syntactic mapping or even an interlingua (Hutchins 1986).[2] In general, research also became more sophisticated linguistically, resulting in a number of approaches which sought to better capture the grammars of the languages involved, rather than improve translation algorithms.[3]

These systems spawned a number of different theoretical approaches, many of which continue to be developed today, and gradually, funding for MT returned to the U.S., although it lagged behind the research efforts in Japan and Europe. What most of these systems shared, whether direct or transfer; word, syntax, semantics, or interlingua based; fully-automatic or human-aided; is that most aspects of the translation models were hand-coded. That situation changed greatly in the 1990s,

[2]Detailed descriptions of some of the varying techniques can be found in Hutchins (1986), and case studies of some of the projects in Hutchins & Somers (1992).

[3]Although there was a perception, at least in the 1980s, that many of the more linguistic systems did not function very well (see section 1.3.1).

with the growth of data-driven approaches, such as statistical MT and example-based MT. These types of empirical approaches will comprise the main part of the literature review in Chapter 2.

1.3 Why MT Is Difficult

No one well-acquainted with machine translation doubts that it is a very difficult enterprise. Prior to presenting the linked automata model, it is important to examine some of the reasons why MT is difficult, especially those bigger picture issues relevant to all systems, regardless of the approach taken. Of course, the problems likely to arise in an MT project could fill volumes in themselves, so here I try to give a brief survey of the types of problems an MT researcher will either have to remedy or tolerate.

The difficulties found in MT can be divided into two groups: 1) programmatic or meta issues, i.e., how MT research is done, and the more important, 2) aspects of the MT task that make it difficult. I begin with the former, and focus more closely on the latter.

1.3.1 Programmatic Issues

There are critics of MT, and then there are critics of the way that MT is done. One of the most respected voices among those questioning the MT research process itself has been Martin Kay (Kay 1982; Kay 1997a; Kay 1997b). Kay's chief criticisms seem to be that there is too much focus on the linguistic parts of the task, that not enough attention is paid to how humans go about translating, and, perhaps most damningly, that these mistakes are repeated over and over: "the determination of those working in the field doggedly to pursue old questions with old methods is apparently without bounds" (Kay 1997a:38). Kay's desire for more research on human translation techniques is echoed by others, such as Gross (1992), but his views on the over-focus on linguistics are somewhat more controversial.[4] Nevertheless, he makes a compelling argument:

> [S]ystems that are made up entirely of linguistic components actually degrade as the linguistics on which they are based improves. This should not be surprising. It is the job of the linguist to describe and account for the meaning potential of words, phrases, sentences, text and dialogues. It is not the job of the linguist to say what part of this meaning potential is being exercised in any given circumstance. Simply put, the better the linguistics, the greater the number and the subtlety of the ambiguities that will be unearthed and the greater the resulting strain on the nonlinguistic components of the system whose job it is to resolve them. Previous systems had little or nothing in the way of non-linguistic components and essentially none are proposed for those being built today. It is for this reason that Systran remains the best MT system available, and that Verbmobil will, with sickening surety, fail to produce anything of the slightest interest. (Kay 1997a:36)

Others, such as Gross (1992) view MT as essentially a linguistic problem, but one that may be intractable, citing familiar arguments that humans simply do not have the capacity to define a mathematical solution to a problem involving translation, since any language used to express the solution would itself be a subset of the language(s) of the larger problem. Gross (1992) also sees cultural differences reflected in languages as being very prevalent, so much so that claims of linguistic universals being an aid to translation are unfounded.

[4]Hutchins (1986) too seems to share this view, that 'perfective' linguistics systems have never been successful, and for the most part never built beyond toy systems.

Many of the researchers critical of MT still favor further MT research, but they want it to be realistic, focusing on areas where it is most likely to be successful, such as fast, lower-quality translation, machine-aided translation, or translation in limited domains (sublanguages or controlled languages). Others, such as Melby (1997), are less pessimistic, and while agreeing with the difficulties posed by MT research methods of the past, point to a number of MT successes, as well as vast improvement in engineering and in linguistics, which have the potential to yield much better systems.

I think that one of the main points a researcher should take from these criticisms is that MT is not purely a linguistic problem, to be solved by linguistics alone. It is a translation problem, requiring insights from human translation, linguistics, computer science, and statistics. Perhaps more importantly, like all seemingly intractable problems, acceptable solutions may be found via methods to approximate; thus MT makes a most compelling case for a synergy between the symbolic and the statistical, and between the linguistic and the engineering approaches. Additionally, like any large-scale research project, MT development must be well planned and designed, and able to accommodate changes in theory and the addition of new components.[5] Regardless of one's view of how MT research should be done, the task-specific problems identified in the next section need to be dealt with.

[5]One of the criticisms of Systran is that it treats every phenomenon as a special case, and that therefore potential improvements are just as likely to degrade accuracy as to increase it (see Kay *et al.* (1994); Wilks (1992)).

1.3.2 MT Task-Specific Issues

The difficulties identified in the preceding section might more properly be called criticisms of the MT process. In this section, we turn to those problems pervasive in the MT task that make it so difficult. The places where a machine translation system is likely to encounter the most difficulty might best be divided into two (not altogether unrelated) groups: 1) ambiguity/indeterminacy and 2) lexical/structural mismatches.

1.3.2.1 Ambiguity

Language is ambiguous. This should not come as a surprise, because language needs to be flexible enough to be used in differing circumstances:

> A language makes available to its users, words and sentences that are flexible, or vague enough, so that they can fit a variety of situations. When placed in those situations, on the other hand, they acquire a precision that no grammar or lexicon could possibly have provided for (Kay *et al.* 1994:20).

I use ambiguity here as a cover term for all the sorts of phenomena which need to be interpreted with regard to a context, be that context local (within a phrase or sentence), global (within a text), or even nonlinguistic (within a genre, subject specific, or cultural); the term thus includes lexical and syntactic ambiguities, as well as other sorts of indeterminacies and underspecifications.

This notion of interpretation relative to a context is an important one, and Kay *et al.* (1994) single it out as the biggest problem for interpretation. They refer to language being *situated*, in that the meaning of a word or phrase is determined by the situation in which it occurs, and that the meaning of larger units, such as texts, is determined by the situations in which they are used. When these situations (or *contexts*) are not taken into account, ambiguities arise.

Arnold *et al.* (1994) identify a number of different types of ambiguities an MT system will face, beginning with lexical ambiguity. Consider the well-worn example of the English word *bank* below:

(2) I have to get some money out. I can meet you by the bank.

To correctly translate (2) into a language which does not have the same lexical ambiguity for *bank* (i.e., financial institution or land bordering water), an MT system will have to refer to the context of the sentence, since the preceding sentence can be used to disambiguate between the two meanings. And suppose it so happened that the target language did possess this exact same lexical ambiguity. Then an MT system could translate the sentence without considering the ambiguity, but how would it know? That is, to take advantage of the parallel ambiguity, a system would either have to not be aware of the ambiguity at all, or be aware of the ambiguities in both source and target languages.

Consider another lexically ambiguous example from Kay *et al.* (1994), using the word *open*. *Open* means something different if shown on a milk carton, on a store window, or flashing on a microwave oven. Again, the key here is context, in this case the world context. Resolving these types of ambiguities may be possible from textual context, but often it is not, and requires world knowledge for disambiguation, such as Bar-Hillel's *pen* example, (1), given earlier. Many MT researchers view this type of problem as one of the most difficult to solve (see, for example, Rosetta (1994)).

There can also be syntactic ambiguities, such as PP attachment ambiguity in the example below (Arnold *et al.* 1994):

(3) Connect the printer to a word processor package with a Postscript interface.

This sentence is syntactically ambiguous in at least two ways. First, it could mean that the printer should be connected to a word processor package and that word processor package has a Postscript interface. The second meaning is that the printer should be connected to a word processor via the word processor's Postscript interface. What is interesting in this sentence, as Arnold *et al.* (1994) point out, is that if one understands something about Postscript interfaces, namely that they are software and cannot be used to make physical connections, only the first meaning is available. Thus, this seemingly ambiguous sentence is not ambiguous, given sufficient knowledge (knowledge which is not available from context). Once again, an MT system which does not have access to this sort of knowledge (which is likely to be the case), will have difficulty translating such sentences.

There are often, of course, cases of genuine syntactic ambiguity. Those sentences which people find ambiguous in at least some contexts would be expected to be difficult for any MT system, such as (4) below:

(4) The girl saw the man in the store.

Such structural ambiguities need not always be resolved in a translation, since the ambiguities can sometimes be retained in the target language sentence. The ability to make such translation decisions with confidence, however, requires that the MT system be aware of the ambiguities in both languages.

There are many, many types of phenomena which fall under this broad category of things indeterminate, all of which pose challenges for MT systems, such as pronoun resolution, ellipsis, and the like, which humans process fluently, but which can cause an MT system to falter. And for MT systems, such difficulties are especially likely, given that misanalyses on either side of the translation relation can cause problems down the road. Should we take these sorts of difficulties to mean that translation should not even be tried? The answer, for my part, is no. What it does mean is that we need to be realistic in our expectations of what MT systems can handle, and we need to leave open means for these types of ambiguities to be resolved, should such technology become available.

1.3.2.2 Mismatches

An equally broad category of potential MT difficulties could be called *mismatches*, covering mismatches on the lexical, structural, and cultural levels. This category is not wholly independent from the ambiguities, but the crucial difference is that the mismatches are a property of a divergence between two languages, as opposed to an indeterminacy in one.

We begin with divergences on the lexical level. Consider the English words *door* and *gate*. In Spanish, both these words are translated as *puerta*. Thus, we might say that we have a conceptual mismatch at the lexical level. Translating from English to Spanish, this poses no problem: We always choose *puerta*. But what is the correct translation in the other direction? Again, as in the ambiguity cases, context may provide the necessary clues.

Another type of lexical mismatch is termed a *lexical hole* (Arnold *et al.* 1994). A lexical hole occurs where one language has a concept lexicalized but the other does not, such the French word *ignorer*, which in English could be translated as *to not know* or *to be ignorant of*, but not with a single word. These types of mismatches will confound translation systems which only map single words or compounds to each other.

Moving away from lexical mismatches, we find the very related phenomena of dealing with collocations and idioms—multiword units which may be noncontiguous. To translate English idioms like *to pass away* or *to kick the bucket* to French, one must know that they idiomatically mean *to die*, thus a good translation might be *mourir*, as opposed to a literal translation of the sequence of words: *to kick the bucket.* But, to handle these types of cases, an MT system must be able to differentiate idioms from syntactically analogous phrases, such as *to kick the chair.* In addition, although they might be handled as special cases, some idioms also vary their form (e.g., *kicking the bucket, will kick the bucket,* etc.), meaning the number of special cases will multiply (Arnold *et al.* 1994). Collocations are similar to idioms, but are more compositional (i.e., the meaning is more easily determined from the words). The problem with collocations is that certain word choices are preferred (Arnold *et al.* 1994):

(5) He made {*had, *took} an attempt

Collocations may be able to be handled with context, special lexical entries (like idioms), or sometimes by taking selectional restrictions into account, but they tend to pose problems for MT systems, especially for generating fluent target language text.

Other (multi-) word units, such as compounds, must also be handled. As Kay *et al.* (1994) state, the mapping from words in one language to another can be anything but direct, since word meanings vary in the circumstances in which they are used. They cite the example of the English *health insurance* and the German *Krankenversicherung*, which literally means *sickness insurance*. Although such mismatches are fixed and might be stored as such in a lexicon, not all mismatches can be, especially when we begin to consider what I will call *structural mismatches*. Given the Spanish sentence:

(6) Ha salido
has left
'He/She departed'

How should it be translated into English, with the pronoun *he* or *she*? These types of structural mismatches (which could be also called indeterminacies, hence the overlap with the previous section) are abundant in translation, and the usual sorts of tactics might be applied (e.g., looking to context for resolution). Other familiar examples include the lack of distinction for gender in Finnish pronouns, or the lack of required determiners in Japanese. Both these instances will cause problems when translating to a language which requires such information. As is the usual story, MT systems which have a better sense of the translation situation, of the context and the meaning of the text, may be able to resolve these types of mismatches.

The most difficult mismatches, like the most difficult ambiguities, are those which relate to culture. In Spanish, when answering questions posed in the negative with *sí* (meaning *yes*), the English translation should be *no*, and vice versa (Kay *et al.* 1994). These types of differences cannot simply be encoded into a lexicon.

This identification of potential translation difficulties, both of the ambiguous and the mismatch varieties, is not intended to be exhaustive, but it should give a sense of why translation, even beyond the large scope of the problem, is such a difficult task. Most translation systems will not be able to account for most of these problems. This will particularly be true for new systems, such as the linked automata model presented in this dissertation, which attempt to exploit simple architectures, and which possess no special components for handling, for example, idioms or grammatical differences, or for storing world knowledge. The system to be proposed will, however, show some potential strengths in this area, because it allows for noncontiguous sequences of words to be related, and was designed with the notion of folding additional components into the system in mind. Additionally, its statistical nature may allow it to learn many of the collocational patterns found in translations. Nevertheless, it would be naive to expect such a system to fare well with such difficult test cases as the examples of this section.

Thus, even beyond the sheer scope of the undertaking, the challenges of automatic translation should not be underestimated. Should we take these difficulties as a sign that translation is impossible? I think not, but we should always bear them in mind, as aspects that will need to be addressed. Systems must be also be adaptable, since those filled with special cases are destined to be limited. It may be instructive to look at what were identified as some of the major obstacles for translation in the past. In 1967, Yngve identified some of the major hurdles MT researchers needed to overcome (as of 1965). One obstacle was the lack of a proper programming language. At MIT, Yngve and his team came up with COMIT, a language designed to make life easier for the linguist as programmer. Viewed today, COMIT code looks a lot like assembly language. The higher level languages of today make development for MT researchers much easier, which is a significant benefit for dealing with the large scope of the MT problem. Yngve also notes problems in linguistics with the lack of sufficient morphological and syntactic theories to analyze source language sentences properly. Linguistic theories have made significant strides in these regards (Kay 1997a). Thus, in many regards, MT research has come a long way. Of course, another problem Yngve identified, which he called *the semantic barrier*, remains: The inherent difficulty in determining the meaning of sentences poses the biggest challenge in their translation.

CHAPTER 2

EMPIRICAL MT: A REVIEW OF THE LITERATURE

In 1988, at the Second TMI conference at Carnegie Mellon University, IBM's Peter Brown shocked the audience by presenting an approach to Machine Translation which was quite unlike anything that most of the audience had ever see or even dreamed of before . . . IBM's "purely statistical" approach, inspired by successes in speech processing, and characterized by the infamous statement "Every time I fire a linguist, my system's performance improves" flew in the face of all the received wisdom about how to do MT at that time, eschewing the rationalist linguistic approach in favour of an empirical corpus-based one. There followed something of a flood of "new" approaches to MT, few as overtly statistical as the IBM approach, but all having in common the use of a corpus of translation examples rather than linguistic rules as a significant component (Somers 1999:113).

2.1 Introduction

The automated translation of one natural language to another is known as *machine translation* (MT). All MT systems must in some fashion model the grammars of the two languages involved, and must also model the relationship between the languages. The form of this relationship, the mapping from the *source* language to the *target* language, has historically been used to categorize theories of machine translation. For example (see Figure 2.1), one of the simplest mappings between languages is word-to-word, where the words of the target language are substituted for the words of the source language. Such methods will in general also require some reordering of the target words. MT theories become more linguistically sophisticated as they move up the hierarchy. Mappings between the syntax of languages allows for structural generalizations and better accounts for word-ordering differences between languages. While syntactic mapping MT systems may appropriately match structures between the two languages, it is clearly preservation of meaning that is most important in translation. Thus, a mapping from meaning to meaning, a semantics-based transfer, encodes more relevant knowledge than does a syntax-based mapping. Another meaning-related possibility is to create an intermediary, an *interlingua*, to which the semantics of each language are connected, allowing for further generalizations as well as potentially decreasing the effort required when the translation of more than one language pair is desired (since the mapping from a given language to the interlingua or from the interlingua to a given language need only be constructed once; thus the mapping to interlingua is analogous to a compiler).

Of course, this characterization of MT theories is greatly simplified. Theories may also take into account contextual information, and thus be pragmatically based, and in addition may make use of cultural information. Similarly, on the lower end of the hierarchy, mappings may range anywhere from between morphemes to between (unparsed) examples of complete sentences. In general, as theories move up the hierarchy, their ability to make linguistic generalizations increases, but so do the knowledge requirements for implementing them.

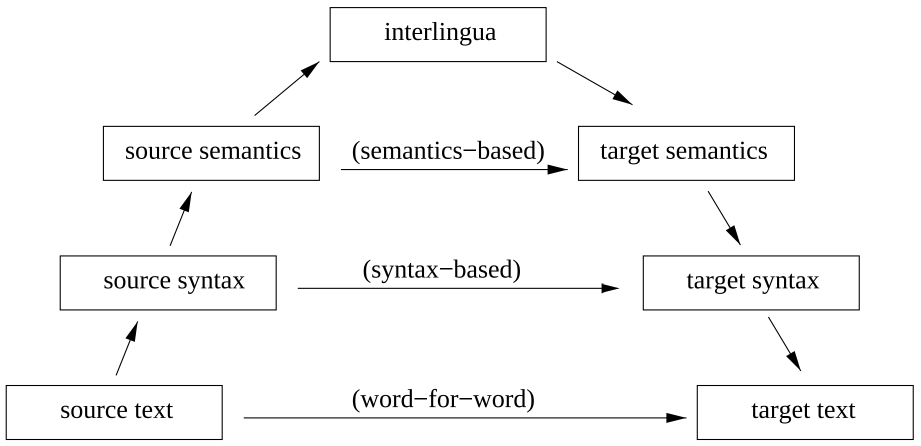


Figure 2.1: Different types of MT models (adapted from Knight (1997))

There are also other dimensions along which one may subdivide theories of MT. MT theories can be classified according to the method of transfer (direct or through some level of linguistic processing), or the domain of possible translations required (e.g., the domain may be restricted to a *sublanguage* relevant to specific tasks), or even the 'amount' of the language involved (e.g., unambiguous subsets of the languages, as specified by PACE, the controlled English of the UK engineering company Perkins Engines; see Arnold *et al.* (1994)). Nevertheless, the hierarchical subdivisions between language-to-language mappings shown in Figure 2.1 capture the essence of what all MT theories must encompass.

The next question for MT theories is how these models of grammar and inter-language relationships should come about. In an ideal situation, the grammars of the two languages would be fully specified by a linguistic theory, and the world knowledge requirements would also be adequately covered. Under such a scenario, successful MT systems would only require implementing the linguistics and AI theories that had been fully explicated. Early MT approaches were based on this ideal. Grammars for the languages were hand-crafted, along with rules for language transfer. In fact, these approaches have dominated theoretical work in MT until only recently, and form the basis for most successful commercial MT systems existing today (e.g., SYSTRAN, see Hutchins & Somers (1992) for a system overview). These approaches have the dual benefit of taking advantage of the theoretical linguistic work already accomplished, and of not requiring examples of translations. In addition, such systems will ideally be able to capture generalizations about the grammars of languages which are not learnable simply from viewing (positive) examples of language data alone (Gold 1967).

However, hand-crafted systems (used in isolation) have a number of drawbacks. First, because the modeling of grammar remains an open research area in linguistics, coverage of such systems will be incomplete. Incomplete grammars lead to a lack of robustness, since otherwise translatable sentences may be rejected as unprocessable because of an incomplete or incorrect grammar. Secondly, the purely symbolic approaches usually associated with hand-crafted systems may be intolerant of the noise and disfluencies inherent in actual natural language use. Finally, perhaps the chief difficulty for hand-crafted approaches is the human effort required. In order to achieve success, grammars must be painstakingly created for each language (ignoring those parts of grammar which one might take to be universal). One could argue that given the complexity and changing nature of natural language, as well as the (non-linguistic) knowledge requirements for translation, hand-crafted approaches alone may never be sufficient for MT.

The recent proliferation of electronic texts of many of the world's more widely-spoken languages offers assistance and alternatives to the hand-crafted approach. Most important for translation are the availability of parallel texts (the same textual content in more than one language). Recent MT research has focused on exploiting these texts (see Melamed (2001)), especially using statistical and example-based methods. Such methods, because of their reliance on data, are known as *empirical* or *data-driven* methods. What such methods offer are increased robustness and decreased human effort. Typically, empirical methods learn automatically from a large training set of parallel texts, with each pair of sentences (or some other subdivision) serving as an example. The decrease in human effort occurs in two ways: The language and translation models for the language pair are no longer hand-crafted, and the given data-driven approach may be applied to other language pairs (in an ideal scenario), without changes.

This is not to say that all data-driven methods are fully automatic. Many data-driven approaches rely on linguistically annotated data, such as part-of-speech tags, word alignments, or syntactic annotations. It is often the case that some of these annotations involve human labor, or that the tools used to make them require a degree of hand-crafted training data. There also exist fully-automatic means to obtain such annotated parallel corpora, but typically the quality of output increases when annotations are inspected and corrected by humans. Often, the availability of such tools for different language pairs will affect the choice of which empirical methods can be used, since different models make different assumptions in terms of the types of annotation required.

Empirical methods seek to learn generalizations from the data they encounter, rather than from a linguistic theory (that is not to say that linguistic theories do not inform how they learn and what they look at). Because of their general stochastic nature, these systems are better equipped to make choices when no single, clear alternative stands out, and thus better handle parts of the translation not covered by an existing grammar. Additionally, they are generally more easily adapted to handle new data.

The system presented in this dissertation (as will be described in Chapter 4) is such a data-driven system, using a combination of probabilistic and finite-state methods. As such, statistical systems in general and statistical finite-state systems in particular will be the main focus of the literature review, but another important empirical area, *example-based machine translation* (EBMT), will also be briefly covered. While all of these empirical methods have promise for MT theories, it may turn out that the best theories incorporate the best parts of empirical and non-empirical approaches, allowing the automated learning techniques to incorporate knowledge from linguistic research. Although discussion of the model to be described in Chapter 4 will be centered on its data-driven core, extensions identified in the dissertation research will involve some hybrid approaches between empirical and non-empirical techniques.

2.2 Statistical Machine Translation

2.2.1 Pure Statistical Methods

It is likely not an exaggeration to say that all of the important work in *statistical machine translation* (SMT) grew out of research efforts of one group at IBM during the late 1980s and early 1990s. The work might be viewed as a partial fulfillment of some of the ideas that Weaver (1955) first proposed, namely using local context for disambiguation, and viewing translation as an encoded signal (see section 1.2). This research culminated in a seminal article by Brown *et al.* (1993). As early as 1988, the research had a profound effect on the MT community, when results from a purely data-driven method were shown to be comparable to human-coded systems, and sparked a bit of a revolution toward empirical approaches (see the quotation at the beginning of this chapter). By 1993, the system described, know as *Candide*, performed as well as the best commercial systems in the translation of French text to English text (Knight 1997).

Thus, because of its importance in the field, and because it remains the best example of 'pure' statistical machine translation, I focus on it here and describe it in some detail. The IBM approach, like all data-driven MT methods, begins with parallel texts. These texts are assumed to be broken up into *bitexts*, pairs of source and target sentences which are taken to be translations of one another (thus the parallel texts are *sentence aligned*; many excellent algorithms exist for sentence alignment, see for example Gale & Church (1993)). The basic idea for the IBM approach is to get the best *word alignments* (the mapping between source and target words;[1] see Figure 2.2 for an example alignment) possible, and to use these alignments, along with models of the source language, as the basis for the probabilistically chosen best translation (as such, the IBM model can be viewed as being at the bottom of the hierarchy shown earlier in Figure 2.1).

[1]Word alignment is discussed in detail in Chapter 3.

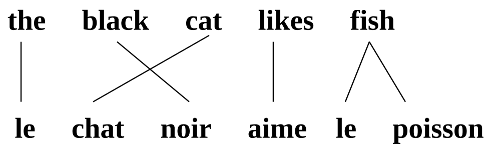


Figure 2.2: An example English and French word alignment

Berger *et al.* (1994) describe their approach in terms of the *noisy channel* model (see Figure 2.3; the communication channel idea is based on the pioneering work of Shannon (1948)), where a given French sentence is to be translated to English. The noisy channel model assumes that the French sentence, *f*, was originally given as its English equivalent, *e*, and must be decoded back to its best approximation in English, $\hat{e}$. The use of this model helps to make clear why the conditional

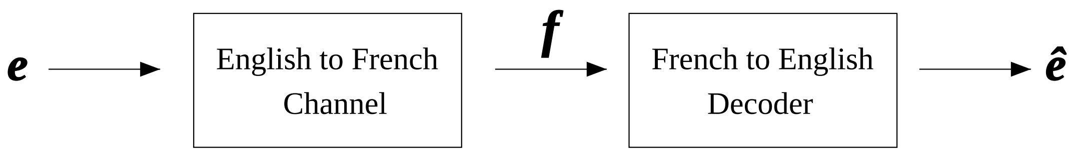


Figure 2.3: The noisy channel model for translation (Berger, *et al.* 1994)

probabilities (described below) used for estimation make sense. This model also explains why SMT researchers, in a translation from *f* to *e*, conventionally refer to *e* as the source and *f* as the target (see Knight & Al-Onaizan (1998)).[2]

[2]To be consistent with that convention, I will adopt the same often confusing practice in this section, although it is possible I will lapse to the more natural terminology. In any case, the translation setup should be clear from the context.

Given this model, the probability of a French-to-English translation can be written as: $P(e|f)$. This conditional probability represents the chance that the English e was the original source of the French f. Thus, given a French sentence f, the problem of finding the best translation becomes finding the English sentence, $\hat{e}$, which maximizes $P(e|f)$:

(7) $\hat{e} = argmax_e P(e|f)$

Employing Bayes' theorem, we get:

(8) $\hat{e} = argmax_e P(e|f) = argmax_e P(f|e)P(e)$

where the rightmost side of the equations gives us more useful terms to work with, since $P(f|e)$, the translation model, and $P(e)$, the language model, allow us to independently model the meaning of the translation and its fluency, respectively.

> The translation model ensures that the words of e express the ideas of f, and the language model ensures that e is a grammatical sentence. Candide selects as its translation the e that maximizes their product (Berger *et al.* 1994:157).

This is, in essence, the entire IBM SMT model, and is therefore most central in understanding the approach. What remains and is rather complicated is how these two models, the translation model and language model, are arrived at (as well as how they are used), which will be described next, where we concentrate on the more complex of the two, the translation model.

The IBM approach uses five translation models of increasing complexity, Models 1-5. Not only are these models used to successively better formalize the translation process, but, additionally, each model is used as a parameter setting tool for the model which follows it in the training process (this is especially true for Models 1-3, which will be the main focus of the discussion).[3]

Model 1, the simplest model, is the word translation model (Berger *et al.* 1994). The parameters of this model are all of the form $t(f_i|e_i)$, which is the probability that French word f_i is the translation of the English word e_i. As stated earlier, the IBM models are alignment models, thus the translation model $P(f|e)$ can be reformulated in terms of $P(f, a|e)$, where a is the series of values representing the connections between the words of e and f (i.e., a is the alignment). Since there are many ways that the same French sentence and English sentence can be aligned, to get $P(f|e)$ we need to sum over all the possible alignments:

(9) $P(f|e) = \sum_a P(f, a|e)$

Each $P(f, a|e)$, in turn, is given by the product of the probabilities of the individual word translations which make it up, i.e.:

(10) $P(f, a|e) = c \prod_{j=1}^{m} t(f_j|e_{a_j})$

where m is the length of the target sentence (French), e_{a_j} is the English word that French word f_j is aligned with under alignment a, and c is a normalization factor related to the target sentence length.

[3]In addition to Brown *et al.* (1993) and Berger *et al.* (1994), much of the discussion in this section relies on the very clear explanations given in Knight & Al-Onaizan (1998) and Knight (1999).

The goal therefore is to produce the best alignments in order to maximize $P(f|e)$, i.e., to maximize (after some simplification):

(11) $\sum_{a_1=0}^{l} \cdots \sum_{a_m=0}^{l} c \prod_{j=1}^{m} t(f_j|e_{a_j}) = c \prod_{j=1}^{m} \sum_{i=0}^{l} t(f_i|e_i)$
(This is Brown *et al.* (1993) formula 16.)

This is estimated by using the EM (estimation maximization) algorithm (Baum 1972). The EM algorithm, given reasonable initial parameter values, proceeds iteratively until a desired degree of convergence is obtained. Thus, Model 1 yields a relatively rough estimation of the best word alignments.[4]

Model 2 is like Model 1 except that it takes into account *distortion* probabilities, which are probabilities related to the (possible) preference for words to be connected based on their relative positions in the source and target sentences (e.g., a target word at the beginning of the target sentence might be more likely aligned with a source word near the beginning of the source sentence than with one near the end of the source sentence). So, $P(f,a|e)$ is now written as:

(12) $c \prod_{j=1}^{m} t(f_j|e_{a_j}) a(i|j,m,l)$ (Knight & Al-Onaizan 1998)

where l is the length of the source sentence, and a is no longer an alignment but rather the probability of the alignment in terms of the word positions and sentence lengths only. The new formula to maximize, replacing (11), is:

(13) $P(f|e) = c \prod_{j=1}^{m} \sum_{i=0}^{l} t(f_j|e_i) a(i|j,m,l)$
(This is Brown *et al.* (1993) formula 26.)

Model 2 is constructed directly from Model 1, using Model 1's word translation probabilities, and training proceeds similarly via the EM algorithm.

[4]I will briefly discuss the *use* of these models in translation, as opposed to their training, at the end of this section.

Model 3 introduces the concept of *fertility*—that a single English word may connect to 0, 1, or more than 1 French words. Fertility thus allows for much more natural alignments. The fertility $\phi(e_i)$ is the number of words e_i generates in the translations (Berger *et al.* 1994), and $n(\phi|e_i)$ is the probability that e_i generates n words (Brown *et al.* 1993). In addition to translation probabilities and fertility probabilities, Model 3 replaces Model 2's alignment probabilities with *distortion* probabilities (which are roughly the same thing, just running in the opposite direction), of the form: $d(j|i, m, l)$. It is at this point that the formula for $P(f, a|e)$ becomes quite complex, because once fertilities are involved, there are many more possibilities to consider (including alignment with the NULL word). p_1 is introduced as the NULL word parameter, as well as its opposite, $p_0 = 1 - p_1$, and, without going into too much detail, we finally arrive at the formula for the translation model of Model 3:

(14) $P(f|e) = \sum_{a_1=0}^{l} \cdots \sum_{a_m=0}^{l} [\binom{m-\phi_0}{\phi_0} p_0^{m-2\phi_0} p_1^{\phi_0} \prod_{i=1}^{l} \phi_i! n(\phi_i|e_i) \prod_{j=1}^{m} t(f_j|e_{a_j}) d(j|a_j, m, l)]$
(This is Brown *et al.* (1993) formula 32.[5])

Here we can think of the terms as follows: $\binom{m-\phi_0}{\phi_0}$ is the number of ways we can put ϕ_0 spurious words into $m - \phi_0$ slots, $p_0^{m-2\phi_0}$ is the cost for not adding spurious words after real words, $p_1^{\phi_0}$ is a cost for adding spurious words after real words, and, as mentioned, $n(\cdots)$ is a fertility probability, $t(\cdots)$ is a translation probability, and $d(\cdots)$ is a distortion probability. Thus ends the presentation of the formulas for the various IBM models.

[5]Knight & Al-Onaizan (1998) add an additional term to this formula, i.e., inside the square brackets, $1/\phi_i!$, apparently as a scaling factor.

In terms of training, Model 3 begins with some of the parameter values supplied by Models 1 and 2, but because of the greater number of possible alignments, its training process uses a heuristic search guided by the best alignments from Model 2, to reduce the search space. One problem with Model 3 is that it is *deficient* (Brown *et al.* 1993). This means that it leaves some probability in its probability distribution for impossible strings—strings where more than one word is in the same position.

Finally, we arrive at the last two models. In Model 4, the distortion parameters are changed so that alignments can be expressed in terms of word classes. This adjustment better allows for the alignment of phrases as units. Model 5 makes the final improvement of eliminating the deficiency of Model 3, by disallowing spurious alignments (i.e., those that are impossible).

Returning to the larger SMT picture, we now have a trained translation model, $P(f|e)$. We still need a language model $P(e)$ (see (8)). The IBM team does not discuss this language model in much detail. However, it can be described as something close to a trigram model which is smoothed by the technique of deleted interpolation (in reality they use a *link grammar*, which is better capable of accounting for long range connections than is a trigram model (Berger *et al.* 1994)).

Given these translation and language models, the final issue is the translation process itself, which is known as *decoding*. Again, the decoding process is only discussed in passing in the literature, and is of course implementation dependent (for example, in Knight & Al-Onaizan's (1998) implementation of the IBM approach, decoding reduces to transducer composition). In Berger *et al.* (1994), decoding (also referred to as *transfer*) begins with a French string, *f*, and using a stack decoding algorithm in conjunction with a beam search, keeps the best alternative translation(s) for each French word as the process proceeds from left to right. Each set of possibilities is a called a hypothesis, h. In the second decoding stage, these hypotheses are ranked according to the product of $P(f|e')P(e')$, where e' is the decoded English of the hypothesis, the translation model is Model 5, and the language model is a smoothed trigram model. In the final stage, they add a *perturbation search*, in which deletion, insertion, and replacement of words are allowed (this procedure is not further specified). This step enlarges the number of hypotheses, which are reranked, and the highest scoring English sentence is the translation. Recently, Germann *et al.* (2001) have also described two new decoders for IBM model 4, including a slow but optimal decoder, and a fast greedy decoder which performs only slightly worse than a stack decoder, but is at least an order of magnitude faster.

In summary, using only parallel texts—and no hand-crafted knowledge—the IBM approach was able to achieve MT results comparable to the best non-statistical approaches. The significance of this result should not be understated, since (for languages where parallel texts exist) it suggests methods for more easily attained translation systems and quite possibly suggests directions for more accurate translations. The keys to this approach are the translation model, $P(f|e)$, and the language model, $P(e)$. While such models can never be perfect, they can offer surprisingly good results even if both are somewhat flawed, because each serves to disallow some of the poorer outputs of the other (Knight & Al-Onaizan 1998). There do exist other purely statistical MT approaches (e.g., the HMM models and others as described in Vogel *et al.* (2000)), but the IBM approach remains the prototypical example, and serves as a useful foundation for discussion of other data-driven approaches.

Before beginning a discussion of models which are both statistical and have a finite-state framework, one other very recent statistical work should be briefly mentioned. Yamada & Knight (2001) is noteworthy because it takes the IBM work to what might be viewed as its logical next step: It takes syntactic structure into account, in a statistical fashion. While the IBM models are based on string-to-string mappings, the Yamada & Knight (2001) model is based on a parse-tree-to-string mapping, i.e., an input sentence is first parsed, then operations are performed on each node of the parse tree, which may include reordering of nodes, inserting of words at nodes, and translation of the words at the leaves of the tree. This model clearly outperforms the IBM models, as measured by a human evaluation of the word alignments produced. The model thus serves as a good example of what can be achieved as more linguistic information is taken into account, even if such information is acquired automatically (i.e., in a data-driven framework). We now move to other approaches which also attempt to represent some degree of syntactic structure (in some cases just linear order), but do so in a finite-state framework. The linked automata model to be presented in Chapter 4 is most closely related to these approaches, and as such their discussion forms the heart of the literature review.

2.2.2 Finite-State Methods

In contrast with the purely statistical nature of the IBM models described in the previous section, there exist a number of data-driven approaches which, in addition to being statistical, have a finite-state framework. In general, the motivation for these approaches is to use more linguistic information than can be captured from focusing on word frequencies alone. In particular, these probabilistic finite-state methods attempt to use the syntactic information present in the bitexts. Much of the syntactic information exploited by these methods, however, is phenogrammatical (i.e., information about linear order) rather than tectogrammatical (hierarchical information), since finite-state technology is particularly well-suited for representing linear ordering phenomena. By making use of this syntactic information, the MT models move one step up the hierarchy shown earlier in Figure 2.1, since the translation involves a syntax-based transfer (although in all cases, the models are also heavily word-based).

These finite-state methods thus employ more linguistic knowledge than does pure SMT, but interestingly, they do so in an automatic way. The finite-state machines or grammars used are constructed directly from the data. In this manner, linguistic knowledge is beneficially employed, but at the outset of the process, via the design of the models. It is the researchers' knowledge that finite-state devices can be appropriate in certain contexts to approximate natural language syntax (see for example Pereira & Wright (1997)) that allows for this informed setup.[6] This is analogous to one of the most important concepts in Machine Learning in general: the appropriate selection of features. In large, open-ended AI tasks, such as machine translation, algorithms must be directed to features (such as words and word orderings) liable to be important, rather than blindly attempting a possibly infinite

[6]Some take this even a step further, and claim that a finite-state framework may be fully adequate to represent natural language syntax, because the (memory) bounds of the human processor constrain those same natural language constructions typically used as evidence for the need for more powerful grammars (Yngve 2000).

set of features and feature combinations. Thus, by employing finite-state technology, the MT models make use of additional linguistic knowledge (and as such may be thought of at least as nominally hybrid), and still avoid relying on the creation of hand-crafted rules. In this section I will describe several general methodologies, in varying detail, which will highlight the important aspects of probabilistic finite-state MT and lead to the discussion of the model to be presented in Chapter 4.

2.2.2.1 Stochastic Inversion Transduction Grammars

We begin with a model (Wu 1997) that is neither truly a translation model, nor finite-state, but that nevertheless serves as a good starting point, both because it is an excellent example of methods for modeling syntactic information from bilingual corpora and because it highlights some of the issues that such methods face. Wu (1997) introduces the concept of (stochastic) *inversion transduction grammars*, a surprisingly flexible technique for representing the grammar of two languages at once. The idea begins with *simple transduction grammars*, which are sets of rules used to generate (or parse) two languages at once, so long as the symbols generated by the right-hand side (rhs) of a rule are concatenated in the same linear order. For example, the following simple grammar can generate the bitexts ⟨*the dog ate / le chien à mangé*⟩ and ⟨*the cat ate / le chat à mangé*⟩:

(15)
$$\begin{array}{lll}
\text{S} & \rightarrow & \text{NP VP} \\
\text{NP} & \rightarrow & \text{Det N} \\
\text{VP} & \rightarrow & \text{V} \\
\text{N} & \rightarrow & \text{dog/chien} \mid \text{cat/chat} \\
\text{V} & \rightarrow & \text{ate/à mangé} \\
\text{Det} & \rightarrow & \text{the/le}
\end{array}$$

Here the symbols on the left side of a "/" indicate the productions for language L1 (in this case English), symbols to the right of the "/" indicate the productions for language L2 (in this case French), and the lack of a "/" means the L1 and L2 share the same nonterminal or terminal. Of course, such grammars are insufficient for translation, because they require the two languages to share the same exact structure.

Inversion transduction grammar (ITG) extends the power of simple transduction grammar by allowing two possible orientations for rule production: *straight* and *inverted*. A straight orientation, notated in rules by "[ ]" around the rhs, means that the symbols for both L1 and L2 are emitted in a left-to-right order, whereas an inverted orientation, notated in rules by "$\langle\ \rangle$" around the rhs, means that the symbols of L1 are emitted left-to-right but the symbols of L2 are emitted right-to-left.[7] With these additions to the formalism we can extend the grammar in (15) to cover $\langle$*the black cat ate / le chat noir à mangé*$\rangle$ by adding/amending rules as follows:

	AdjP	$\rightarrow$	black/noir
(16)	NP	$\rightarrow$	[Det N']
	N'	$\rightarrow$	N | $\langle$AdjP N$\rangle$

These rules can apply at any point in derivation (i.e., not just to leaves of a tree), allowing the orders of large constituents of the two languages to vary. Note that this formalism is context-free, and not finite-state, but its importance here is its shared goal with the finite-state methods to be discussed—to account for the different word ordering facts between (fixed word order) languages. ITGs do not allow for all possible word orderings, i.e., there is a crossing constraint: Matching between subtrees is allowed only if the parents are also matched. This constraint serves to limit the number of possible matches (alignments) in a linguistically plausible way,

[7]In cases where just one symbol is emitted, brackets are of course unnecessary.

which is important in terms of computational complexity. Again, this is a problem that all the probabilistic finite-state methods will have to deal with. Of course, this formalism is not adequate for modeling (pairs of) more freely ordered languages (Wu 1997).

ITGs can form useful grammars for what allowable word alignments should be (see Wu (1997) Figure 5 for a detailed example of alignments which can be represented with two four symbol strings). This is important, since many MT algorithms reduce to models of word alignment. Another relevant step in Wu (1997), beyond modeling a bilingual grammar, is the conversion of the formalism to a stochastic one, as well as the presentation of efficient algorithms for processing (in this case parsing, but steps taken are relevant to translation as well). The rules are made probabilistic in the usual way (i.e., each rule has an associated probability). This stochastic step allows for efficient processing (e.g., by the use of n-best heuristics, since the probabilities allow a means to choose between alternatives).

Wu (1997) also points out that bilingual grammars like ITGs should in general be more appropriate than *parse-parse-match* strategies (i.e., those MT training strategies which parse the source and target strings first separately), since: 1) monolingual grammars may not be available; 2) grammars may be incompatible (i.e., different types of constituents); and 3) selection between different structures may be arbitrary (and thus harder to decide in isolation). This is an important point. Given parallel texts, there is important information inherent in each bitext that may be obscured or lost by processing the bitext's sentences separately.

2.2.2.2 Composed Transducers

As mentioned previously, Knight & Al-Onaizan (1998) implement the purely statistical IBM Model 3 (Brown *et al.* 1993) via composed transducers. In terms of translation output, the results are theoretically identical (i.e., this was one of the goals of the research). As such, (Knight & Al-Onaizan 1998) does not represent an advance in terms of coverage, but I briefly detail it here to identify the insights it offers in terms of design, by showing how statistical MT can be implemented using finite-state devices. The motivation for making a move to finite-state technology is straightforward: 1) Finite-state devices are arguably more easily understood and used than purely statistical methods; 2) finite-state models move away from the pure word alignment model for translation, allowing word-by-word processing which may be more reflective of syntactic considerations; 3) there are very efficient algorithms and techniques for using finite-state devices, especially due to work in speech processing; and 4) finite-state devices are easily composed, allowing for a more modular (and therefore more simple) partitioning of the various tasks involved with the IBM SMT method (i.e., word translation, fertilization, distortion, etc.).

Knight & Al-Onaizan (1998) use weighted finite-state transducers, which are simply state transition diagrams where labels on transitions take the form $a : b/w$, where a is an input token, b is an output token, and w is the numerical weight. To use the transducer as a translation model, $P(f|e)$, for a given source string, e', and target string, f', one finds all paths through the transducer which accept e' and output f', multiplying the weights of the transitions for each path. The highest scoring path allows selection of the best transduction, which is, for this model, taken to represent the best translation.

Since transducers can be composed, Knight & Al-Onaizan (1998) implement the different parts of the IBM Model 3 as separate transducers. Namely, for the translation process they use six composed finite-state devices (actually four composed transducers, along with two acceptors). Figure 2.4 shows the relevant steps of the model for the translation of *le chat noir aime le poisson* (the alignment for this example was shown in earlier Figure 2.2). At the top of this chain is a finite-state

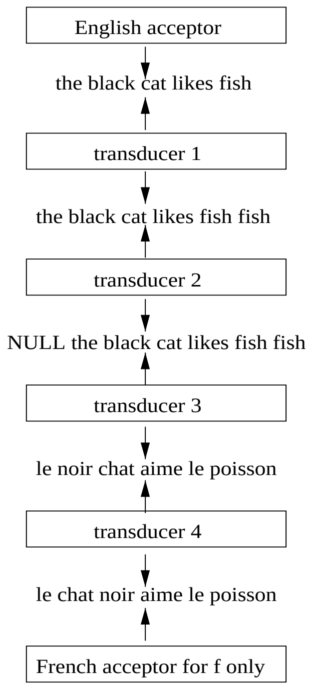

English acceptor
the black cat likes fish
transducer 1
the black cat likes fish fish
transducer 2
NULL the black cat likes fish fish
transducer 3
le noir chat aime le poisson
transducer 4
le chat noir aime le poisson
French acceptor for f only

Figure 2.4: SMT as a cascade of finite-state devices

acceptor (an automaton) of the source string, e. This acceptor simply accepts or rejects the string. Next is transducer 1, which determines fertilities. If an English word x has a fertility n, then the transducer outputs x n times. The next transducer,

transducer 2, simply distributes NULL words into the output, with (relatively low) probabilities (i.e., as in Model 3, with the NULL word probability of $p1$; for this translation example no NULL words are actually necessary, although one is shown but not used). Transducer 3 is the word translation transducer. It substitutes target words for source words, one-for-one. The last transducer is transducer 4, which permutes the target (French) words into their proper order, so that they can be accepted by the permutation acceptor at the end of the cascade.[8] This acceptor must contain all possible orderings for the target string, but avoids growing too large because it is built only for the particular target sentence (recall that even though it is called the target sentence, the French sentence is the one given to be translated, so it takes no magic to build an acceptor for it, nor are any probabilities necessary). Actual translation (decoding) works its way up from the bottom from the French acceptor all the way to the English acceptor.

There is also an additional transducer representing the language model P(e) (the acceptor at the top of the chain simply accepts or rejects strings, but does not assign a probability). Like Brown *et al.* (1993), Knight & Al-Onaizan (1998) use a trigram model smoothed by deleted interpolation, implemented as a weighted finite-state transducer. Finally, note that just as in the IBM case, since more than one alignment may correspond to the same translation, the probability of each path through the composed devices for a given translation must be added, to get a probability for the translation.

[8]It is not clear in Knight & Al-Onaizan (1998) if transducer 4 and the permutation acceptor are actually a single device, so this description and Figure 2.4 may be slightly inaccurate.

Knight & Al-Onaizan (1998) also discuss several additions as well. A transducer for distortion probabilities could be inserted, where source words are followed by a position indicator, e.g., for *fish*: *source-pos5*, which are later converted to target position indicators, e.g., *targ-pos6*. They also discuss marking identical source words that have different translations as separate, so that they can have distinct fertilities, and grouping words as phrases (e.g., *le poisson*). These sorts of extensions will correlate with some ideas for extending the linked automata model (see Chapters 6 and 7).

The most important open question for this finite-state replication of IBM Model 3 is whether it will scale-up reasonably. Knight & Al-Onaizan (1998) leave this as a topic for future research. The technique most likely to achieve success in this area is to use known "pruning and lazy composition methods that find approximate best paths in the face of overwhelming possibilities" (Knight & Al-Onaizan 1998:436). These steps, in conjunction with n-best heuristic search techniques, could make the model scale reasonably both in terms of space and time complexity. A final positive point for the research area is that the composition and decoding algorithms are used in other applications beyond MT, lending credence to the idea that they will be efficiently engineered and easily understood.

2.2.2.3 Subsequential Transducers

We next turn to a very interesting data-driven finite-state translation approach which uses *subsequential transducers*. In a series of papers extending from the early 1990s to the present (see, for example, Oncina *et al.* (1993), Castellanos *et al.* (1994), Oncina & Varó (1996), Amengual & Vidal (1996), and especially Vilar *et al.* (1999) and Amengual *et al.* (2000)), a number of Spanish researchers have developed quite effective methods for using subsequential transducers for limited domain, restricted-syntax MT tasks. Subsequential transducers (SSTs) are transducers where for any given state there is at most one outgoing transition for any input label (Roche & Schabes 1997). More formally, for a transducer $H =< X, Y, Q, q_0, E >$, where X is the input alphabet, Y the output alphabet, Q a finite set of states, $q_0 \in Q$ the start state, and $E \subseteq Q \times X \times Y^* \times Q$ is a set of edges (i.e., these are the transitions), then H is subsequential if and only if for any $< p, x, y, q > \in E$ and $< p, x, y', q' > \in E$, then $y = y'$ and $q = q'$. Additionally, final states in subsequential transducers can emit output symbols (that is what makes them final). This behavior is defined by a (partial) function $\sigma : Q \rightarrow Y^*$ called the *state emission function.* An SST which can translate the following four French phrases: *le chat, le chien, le chat noir, le chien noir* into the following English equivalents: *the cat, the dog, the black cat, the black dog* is shown in Figure 2.5.

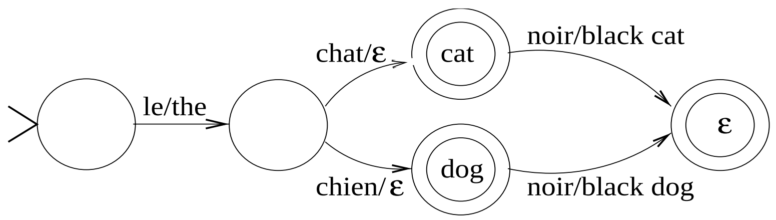


Figure 2.5: An example of a subsequential transducer (SST). The large arrow head points to the start state, ε represents the empty string, and final states are enclosed in double circles.

The main motivation for using SSTs is that they are highly accurate translators in restricted domains which potentially generalize well, and can become smaller by using appropriate generalization algorithms as the number of training examples increases. How SSTs fare in more realistic natural language translation remains an open research question.

I present the basic learning, generalization, and transduction ideas as described in Vilar *et al.* (1999). Given bitexts, a SST which is a compatible generalization of the bitexts can be learned via the Onward Subsequential Transducers Inference algorithm (OSTIA, (Oncina *et al.* 1993)). Here "compatible generalization" means that if the training data (i.e., the bitexts) are representative of a total subsequential function, for a sufficiently large number of training pairs the learning process converges, i.e., all proper translations will be learned, as well as many translations for unseen (and probably unlikely) input sentences. But this total subsequential function requirement is clearly not in general true of natural language translation (i.e., the same word may be translated differently in contexts where the word(s) preceding it are the same), hence the notion that SSTs are most appropriate for limited domains and restricted (ideally finite and unambiguous) syntax. That is not to say that the SST model is completely inadequate for unrestricted natural language translation, because techniques for approximation do exist. I will not outline the full algorithm here, but basically OSTIA takes the input sentences and represents them in a prefix tree, moving common prefixes towards the root by merging states as much as possible. Merging only takes place if the resulting transducer would be both subsequential and not in contradiction with the training examples. State merging allows for generalization, i.e., the transduction of input sentences on which the system was not trained.

OSTIA creates a good translation model, and has the added benefit of greatly reducing the size of the transducer that would be created absent generalization, but it over-generalizes. As such, it results in a poor language model for the input language.[9] Oncina & Varó (1996) attempt to rectify this with OSTIA-DR (where D is for domain, and R for range). OSTIA-DR uses domain and range models (models of the input and output languages) to restrict the generalization of the learned SSTs. The general idea is that the domain and range models can reject sentences, thus providing negative information, in an attempt to sidestep the limitations for language learning in the presence of only positive examples. The domain and range models can be learned from the same initial training examples (although better models may arise from unrelated data). This process is, of course, an approximation at best, but it can accelerate the learning process (Oncina & Varó 1996), although once again its utility for unrestricted language is unknown.

The next major issue for the SST model deals with asynchrony between the input and output languages (i.e., the different word orderings that occur). SSTs by their nature tend to be long strings of states with input/output labels being dominated to the left by input language constraints and to the right by output language constraints (or vice versa). This occurs because of the subsequential requirement. During construction, transitions will often be made with an input symbol, x, with an empty output symbol, ε, i.e., x/ε, because output must be delayed until enough input is seen to guarantee subsequentiality. This means there will be an excessive growth in the number of states and in the need for training data for convergence.

[9]Note that transducers in general can be viewed as two language models and a translation model combined.

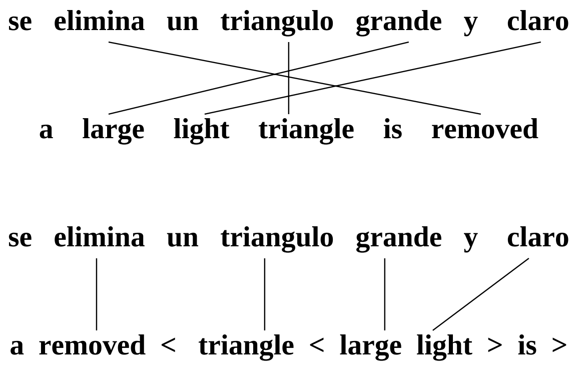

se elimina un triangulo grande y claro
a large light triangle is removed
se elimina un triangulo grande y claro
a removed < triangle < large light > is >

Figure 2.6: The reordering process. The original pair with alignments is shown on top, and the reordered result on bottom (not all alignments are shown).

Vilar *et al.* (1999) cleverly attempt to resolve this problem by using the special symbols $<$ and $>$. The symbols placed in an (output) sentence mean that the word preceding $<$ has to be placed after the paired $>$, when the sentence is rearranged to its original order. In other words, the special symbols are used to put the output words in an order that is more compatible with the order of the input string, and therefore allow for a drastic reduction in the size of the resulting transducer, yet still provide a means to recover the proper output string order. Borrowing an example from Vilar *et al.* (1999) with Spanish to English translation, consider the translation shown in Figure 2.6. We can put the English sentence in a more compatible word order (fewer crossing alignment arrows and fewer alignments spanning long distances), as shown in the bottom alignment pair. Of course, using such a technique means that the bitexts must first be word-aligned. Vilar *et al.* (1999) detail the overall steps for bracketing and later reordering the output sentences. An important consideration in this methodology is that the language defined by the “bracketed” sentences is context-free but not regular. Vilar *et al.* (1999:129) do not see this as a major problem because “the number of levels of bracketing can be assumed to be finite and not too large.” Another potential problem with this reordering technique is that it is not clear how well the transducer would function if used in the opposite direction (i.e., with the output language as the input).

A final important issue discussed by Vilar *et al.* (1999) is the problem of input which is noisy (contains errors, repetitions), of unknown vocabulary, or of unexpected syntax (constructions not covered by the model). SST performance, as with finite-state models in general, degrades rapidly in such cases, since the input strings cannot be processed at all, thus no translation is produced. Error correcting parsing (ECP, (Amengual & Vidal 1996)) is used to alleviate this problem. The approach is to create an error model (this can be attempted automatically by systematically distorting training data for the input language). The error model, E, should account for vocabulary variation, deletions, insertions, and so on. It can then be incorporated into the system probabilistically, where the input sentence $x \in E$ is assumed to be a corrupted version of some sentence $\hat{x} \in L$, where L is the input language. Parsing a source sentence is a matter of finding the sentence $\hat{x}$ which maximizes the probability that x is a corrupted version of $\hat{x}$ (note the similarity to the pure SMT approach):

(17) $\hat{x} = argmax_{x' \in L} P_L(x') P_E(x|x')$

Vilar *et al.* (1999) show that ECP greatly reduces the number of translation errors for a test set created with what is hoped to be realistic errors for the restricted language they use.

In summary, the SST methodology offers a finite-state framework for translation, especially in cases of limited domain, restricted languages. Vilar *et al.* (1999)

provide insights regarding generalization, better input models, size reduction, dealing with asynchrony, and coping with unexpected input during parsing. What is most interesting is that these are many of the important issues which the system to be presented in Chapter 4 must also deal with. Thus, the SST model will prove to be one of the most relevant points of comparison.

2.2.2.4 Weighted Head Transducers

The next data-driven finite-state MT model I consider is one of the most complex. Alshawi *et al.* (2000) and Alshawi & Douglas (2000) present an MT method which uses a collection of *weighted head transducers* to model the hierarchical structure of sentences via dependency trees. The motivation for this approach is straightforward: If, in addition to statistical word correlations, hierarchical syntactic information can be made available, it should prove to be of great use in the translation process. A second important motivation is a significant reduction in system size that can be achieved because the use of hierarchical information allows for recursive transduction techniques. Thus, the given phrases need only be stored once, rather than repeated as subparts of sentence paths which otherwise differ. Unlike pure SMT approaches such as Brown *et al.* (1993), the head transducer models have "non-uniform linguistically motivated structure" (Alshawi & Buchsbaum 1997:360). In order to move toward these more linguistically sophisticated structures, the head transducer MT model requires significantly more power than do other finite-state systems, such as SSTs.

A head transducer (HT) is a transducer which, instead of reading input and writing output from left to right, can read input from any position on the input tape, and write output to any position on the output tape. Thus, head transducers are quite powerful devices, which are not subject, for example, to limitations in terms of asynchrony, as are SSTs. Formally, a head transducer (as defined in Alshawi *et al.* (2000)), is a 5-tuple $H =< W, V, Q, F, T >$, where W is the input alphabet, V the output alphabet, Q a finite set of states, $F \subseteq Q$ a set of final states, and T a set of transitions. A transition from state q_1 to state q_2 takes the form $\langle q_1, q_2, w, v, \alpha, \beta, c \rangle$ where $w \in W$, $v \in V$, the integer α is the input position, the integer β is the output position, and c is the weight (the weight is the probabilistic part of the transition). Valid derivations occur after each input symbol has been read once, and a final state has been reached. The output string at the end of a derivation is the sequence of symbols on the output tape, ignoring any empty spaces on the tape. Head transducers so defined are much more expressive than traditional transducers. For example, they can reverse a string of arbitrary length, which traditional transducers cannot. A head transducer which can do this with the input and output alphabet $\{a, b\}$ is shown in Figure 2.7 (this example is from Alshawi *et al.* (2000)), where above each transition is the *input label:output label* and below is the *input position:output position.*[10] In principle, there are no word ordering differences in translation for which a head transducer cannot account.

Head transducers are so called because of their potential for middle-out transduction, as if starting with a syntactic head (akin to head-first parsing). Used as models of dependency trees, a syntactic formalism where the dependents from a head are connected to it via arrows, head transducers can read and write to

[10] -1 can be taken to mean the next unread or unwritten position to the left and $+1$ the next unread or unwritten position to the right.

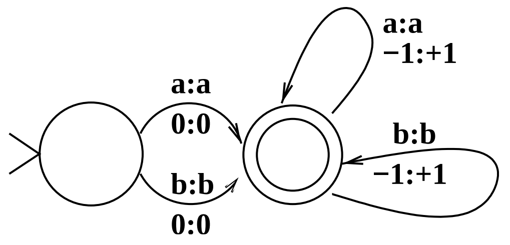


Figure 2.7: A head transducer which can reverse a string of arbitrary length in the alphabet {a,b}. 0 is taken to be the rightmost position of the input string and the leftmost position on the output tape.

the right or left, processing the daughters (i.e., the dependents). As a translation model, input to a head transducer is a string corresponding to a flattened source dependency tree, and the output is a string corresponding to a flattened target dependency tree. Head transducers are applied recursively to accomplish translation: For a given source input tree, each subtree is processed yielding a corresponding target subtree, until the entire original tree has been processed. The words of the flattened target tree are the translation.

Like all the data-driven methods discussed, the HT method begins with bitexts. And, like many of the other methods, the bitexts need to be aligned.[11] Because of the desired higher level of linguistic information (i.e., hierarchical syntactic information as modeled by dependency trees), word alignment is not sufficient. The dependency alignments must also be learned, and the heads must be identified. As earlier, I will not describe the alignment methods,[12] but the basic idea is to use a

[11]In some methods alignment *is* the model, but in most finite-state approaches alignment is a means to an end and not the full MT model in itself.

[12]Alignment is a subfield in its own right, but is not the subject of this research.

statistical measure of correlation among words. Unfortunately, the strategies for selecting heads and dependents are rather crude (Alshawi *et al.* 2000), and appear to be based on correlation strength alone; that is to say, source and target heads are identified as those words with the strongest correlation measure, and identification of dependents proceeds similarly. Such a strategy can sometimes lead to hierarchical structures that do not appear to be linguistically desirable (see for example Alshawi *et al.*'s (2000) Figure 4, and also Figure 2.8 in the next section), raising the question of how much more information has been gained as compared to word alignments alone. In any case, better alternatives to the head and dependent identification method are not readily apparent, and certainly the strategy of using more linguistic information seems well motivated.

The probabilistic weights of the transitions allow a means to select between alternative derivations (translations), of which there will be many since the model contains many small separate transducers (i.e., there are separate transducers for a head and its daughters, and transducers for the daughters themselves). Like other finite-state MT models, the head transducer model needs a way to generalize to unseen inputs. This is accomplished through the merging of states among the smaller transducers. This merging occurs often. In Alshawi *et al.* (2000), for any given word and translation pair, $w : v$, there is only one initial state and one final state (i.e., the same instance of a word translation is not repeated). Additionally, intermediate states (those not start or final states) are merged whenever their incoming transitions differ only in terms of the target position. In essence, this strategy creates a single, larger transducer capable of a large degree of generalization.

As with the SST model, the HT model must cope with situations where the source string cannot be parsed, i.e., there is no derivation. Alshawi *et al.* (2000) take the pragmatic approach and concatenate the shortest length sequence of partial derivations that has the highest probability.[13]

Performance for the HT translation model appears promising, as they achieve relatively good accuracy for translation for short natural language sentences (in a limited domain) from Spanish to English and from Japanese to English.[14] By using many small transducers which do not overlap (but may later be connected), Alshawi *et al.* (2000) avoid the explosion in the number of states that can occur in (ungeneralized) single transducer models. Additionally, the use of available syntactic information may provide knowledge which is beneficial for translation. Finally, Alshawi *et al.* (2000) also identify possible extensions to the HT model, such as techniques for identifying and using compounds, by treating groups of words as single entities, which can lead to more robust translations.

2.2.2.5 Two Models in One: Lexical Selection and Reordering

The final purely empirical finite-state MT approach which I will examine bears some relation to the head-transducer approach of Alshawi *et al.* (2000) (section 2.2.2.4), and grew out of the same research lab at AT&T. Unlike the head transducer approach, however, the Bangalore & Riccardi (2001) method breaks the translation problem into two parts, lexical selection and lexical reordering (LSLR), and uses

[13]It not clear to me whether these two criteria can conflict, but one would typically expect shorter derivation sequences to have higher probabilities, so the issue might never arise.

[14]Japanese to English is considered one of the more challenging MT language pairs and directions.

traditional transducers to map source language strings to target language strings. Bangalore & Riccardi (2001) shares with the head transducer approach its method of beginning with a hierarchical alignment of dependency trees, and thus begins its attempt to capture hierarchical structure. But rather than aligning these trees in a translation model, Bangalore & Riccardi's (2001) approach maps sequences of source words which represent phrases to sequences of target words which represent phrases.[15] The chief contribution of their method (in addition to modeling the syntactic structure) is to separate the target language model from the translation model, in a way which can be later combined, due to the composability of transducers.

The first step in the Bangalore & Riccardi (2001) method is to create the lexical selection model. This begins the same way as in the head-transducer approach, following (Alshawi *et al.* 1998), with the automatic alignment of bitexts, resulting in hierarchically aligned dependency trees. As mentioned in section 2.2.2.4, this can result in representations which appear to be linguistically unmotivated. For example, Figure 2.8 shows on the left an example of an induced dependency representation, from Bangalore & Riccardi (2001), and on the right, what might be taken to be the most plausible representation.[16] As can be seen in the figure, the automatically induced structure on the left deviates quite a bit from the structure on the right, with many implausible dependency relations, such as *I* being the head

[15]Their string transducer approximates a tree transducer (Bangalore & Riccardi 2001).

[16]Unlike traditional dependency trees, the arcs are shown pointing from dependent to head, following the formats of (Bangalore & Riccardi 2001; Alshawi *et al.* 2000). Also, in the tree on the right, *to* is taken to be the head of the verb phrase headed by *make*, rather than a dependent of *make*, following Mel'čuk (1988), and what is likely the most the traditional syntactic analysis.

of the entire sentence, and *make* being a dependent of *call*, which appear to be more a product high co-occurrence than of likely syntactic relations. This again raises the question of whether the hierarchical information will add more knowledge to the translation system than could be gained from employing word alignments alone.

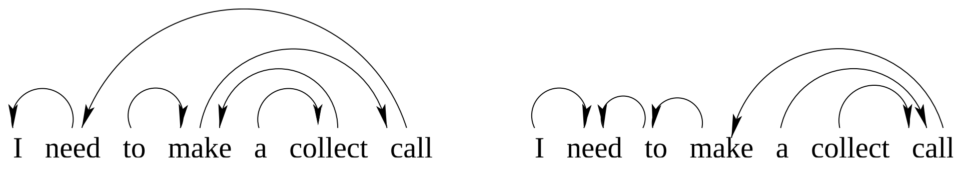


Figure 2.8: A dependency tree representation induced in Bangalore & Riccardi (2001) (left) and a more linguistically motivated representation (right), with dependents pointing to heads in both cases.

After the alignment stage of the training of the lexical selection model (which allows for 2:1, 1:2, 1:0, and 0:1 alignments of source to target word sequences), a bilanguage corpus is created, consisting of source to target sequences. From this corpus, Bangalore & Riccardi (2001) train a stochastic finite-state transducer (SFST). They then take a very interesting step to move from alignment of word sequences to alignment of phrases. This is accomplished by taking substrings (continuous sequences) from the bilingual corpus and finding those which have high mutual information. These strongly correlated substrings (where the correlation must hold on both the source and target sides, this being a bilingual corpus) are taken to be phrases. In the final step in the lexical selection model creation, the words in the aligned target phrases are reordered if necessary, to correspond to proper target language word order (this stage is called *local reordering* and should not confused with the lexical reordering stage to be discussed shortly). This leaves a set of source phrase to target phrase alignments from which a transducer can be constructed.

The lexical selection model outputs a sequence of target language phrases, each of which may be in the correct target language order, but with the entire target sentence (collection of phrases) not in the proper order. Thus, the need for the lexical reordering model arises. The lexical reordering model is created by constructing another stochastic finite-state transducer. This SFST is created from a corpus of source-ordered target language sentences and target-ordered target language sentences.

The entire translation model is then made by composing the two SFSTs: the lexical selection transducer and the lexical reordering transducer. A third transducer is also composed with this model, to ensure that the resulting target strings are well formed in terms of the brackets they contain. Bangalore & Riccardi (2001) use brackets to indicate the proper ordering of phrases, in a manner similar to Vilar *et al.* (1999), and view this step as a finite-state approximation of a "parenthesis context-free grammar upto a bounded depth." Decoding (i.e., translation) then is simply a matter of transducing a source language string into a target language string. This straightforward decoding, along with composability, is one of the main motivations for using transducers in translation systems.

Bangalore & Riccardi (2001) report promising results on a English to Japanese translation task (their accuracy numbers appear slightly lower than those of Alshawi *et al.* (2000), but the data seem to have been different, so it is not a valid comparison). Lexical reordering appears to significantly improve translation accuracy. Interestingly, however, the use of phrases (see Bangalore & Riccardi's (2001) Table 1) does not necessarily seem to improve accuracy, suggesting that the mutual information measure of substrings may not be the best measure for identifying phrases.

On the whole, this is a very interesting translation model which takes advantage of finite-state techniques to separate different parts of the translation process. The separation of translation model from language model is somewhat analogous to that which motivates the linked automata model, presented in Chapter 4. Like the head transducer translation model approaches, Bangalore & Riccardi (2001) take a number of steps to incorporate hierarchical syntactic information into the model, creating one of the more complex finite-state translation models which is purely data-driven.

2.2.2.6 A Hybrid Finite-State Model

The last finite-state model to be discussed is one which is not purely data-driven, but serves as a good example of how finite-state techniques can be combined with hand-coded knowledge to yield a hybrid system. I describe it briefly here, but some of the ideas and means for incorporating them may offer useful extensions to the linked automata model (extensions are discussed in Chapters 6 and 7).

Vogel & Ney (2000) employ cascaded transducers to perform translation for VERBMOBIL, a system whose goal is translate speech between English, German, and Japanese, where utterances mainly deal with scheduling meetings. In this domain, proper handling of names, dates, and times is imperative. Such a problem domain lends itself to hand-crafting, especially since entities such as dates and times have their own specialized syntax. Vogel & Ney (2000) use seven transducers (arranged in a hierarchy): 1) proper names, 2) spelling sequences (i.e., a person attempting to spell something out loud, such as “S P E double L”), 3) numbers, 4) simple time and date expressions, 5) compound time and date expressions, 6) part-of-speech (POS) tagging, and 7) a grammar transducer. Again, since transducers can be easily combined via composition, these specialized transducers taken together can be viewed as a translation system, but keeping them separate allows researchers the opportunity to focus on and improve individual aspects of the application.

Most of the transducers are completely hand-crafted (an exception is the POS transducer, which is mostly automatically constructed), including the grammar transducer, which is based on POS tags, and is the means by which word order differences between languages are captured. The system achieves generalization through the use of the POS tags, by allowing both tags and the translation of source words to filter down the cascade towards the translation. In this manner, the tags can be employed along with the translated source words to help guide the proper ordering of words in the output target sentence. One non-finite-state aspect of the system is the use of a separate target language model (a word-based trigram model) to help choose between different translations produced. Vogel & Ney (2000) experiment with a number of different weightings for this language model, relative to the translation model, and achieve their best results with an equal weighting between the two.[17] Vogel & Ney (2000) use an novel approach for

[17]The results of Vogel & Ney (2000) seem roughly on a par with other published results (such as those reported for Alshawi *et al.* (2000)), but again, cannot be truly compared since the testing and training data are different, as is the means of evaluation; this is a long-standing problem for machine translation evaluation, as will be discussed in Chapter 5.

approximate matching of input source sequences to training sequences. They use a weighted edit-distance, where the idea is: It is better to match an input string to a training sentence that is only slightly different (for example, just a few function word differences) from a training sentence, than to reject it completely. Both this technique of partial matching and the hand-constructing of small, special purpose devices may be useful for other finite-state translation systems.

2.3 Example-Based Machine Translation

The final empirical MT approach to be explored is example-based machine translation (EBMT). Because EBMT is so different from the statistical, finite-state methods of the model presented in Chapter 4, here only an overview of its main ideas is given. As a distant cousin of its other (mainly statistical) data-driven counterparts, EBMT's major similarity is its reliance on large amounts of parallel texts in order to learn. Thus, all of the data-driven methodologies can be seen as example-based, but EBMT is distinguished by its focus on matching inputs with suitable stored examples, as well as the process of recombination of the translations of these examples to form the final translation output. A secondary reason for this review of EBMT is that some of the techniques used to find partial matches of inputs, as well as the techniques to recombine translated subparts, may prove to be helpful for increasing the coverage of the linked automata system. Additionally, the insights from EBMT matching may be useful in the evaluation of MT results (see Chapter 5).[18]

[18]This section benefited from the excellent and thorough survey of EBMT research through the late 1990s provided by Somers (1999).

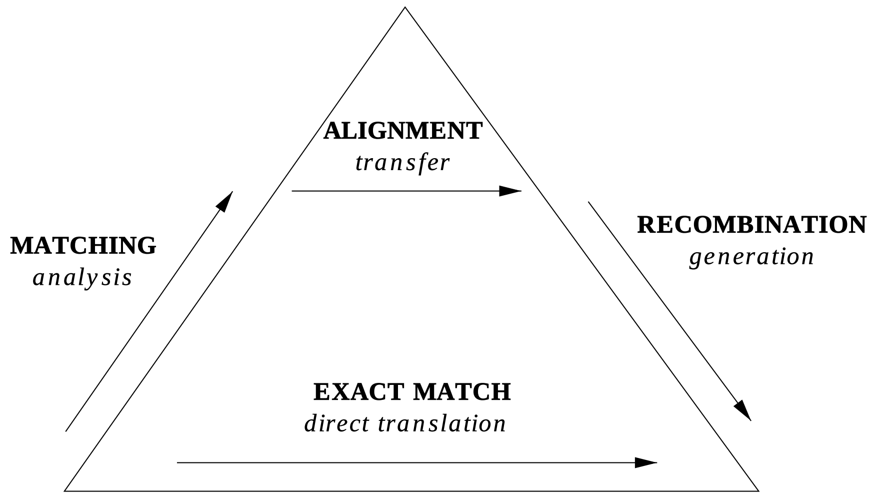


Figure 2.9: The "Vaquois pyramid" for machine translation, adapted for EBMT. Traditional rule-based labels shown in italics, EBMT labels are capitalized (from Somers (1999)).

EBMT's three main steps, *matching*, *analysis*, and *recombination*, can be compared with the more traditional rule-based MT approach of analysis, transfer, and generation (see Figure 2.9). An input text is first compared to the database of stored examples (matching). Once the best match is found, the corresponding target text for the matched text is found (analysis). In the last stage, the target texts may need to be recombined if only fragments were matched in the matching stage (recombination). EBMT bears a strong relationship to the nearest neighbor/exemplar-based approaches used in machine learning, although these connections are rarely made in the literature (Somers 1999).[19] There are several motivations for such an example-based approach, which has also been called analogy-based, case-based,

[19]One recent EBMT approach which explicitly uses machine learning techniques for matching is McTait (2001).

and experience-based. The first is an appeal to human translation and cognitive processing—that people do not use rules to translate but rather look for analogous translations:

> Man does not translate a simple sentence by doing deep linguistic analysis. Man does the translation, first, by properly decomposing an input sentence into certain fragmental phrases . . . then, by translating these fragmental phrases into other language phrases, and finally by properly composing these fragmental translations into one long sentence. The translation of each fragmental phrase will be done by the analogy translation principle with proper examples as its reference . . . (Nagao 1984:178–179).

Veale & Way (1997) see this as an appeal to modeling linguistic performance, rather than competence. Another motivation for EBMT is like that of all data-driven approaches: It is arguably more robust and requires little to no hand-coding, as compared with the traditional rule-based systems. An additional motivating factor is its potential to handle new data, where increasing the coverage is sometimes viewed as a matter of nothing more than adding new examples to the database.

Historically, EBMT can most directly be traced back to Nagao (1984). In a translation task between English and Japanese, Nagao (1984) sees no use for detailed linguistic analysis, because the structures of the two languages are so different. The goal is to see as wide a scope of context as possible in a sentence, and to make the translation between blocks of words. The general EBMT approach can be illustrated by the following example from Sato & Nagao (1990), translating (18) from English to Japanese:

(18) He buys a book on international politics.

(19) He buys a notebook.
Kare ha nouto wo kau.
He *topic* notebook *obj* buy.

(20) I read a book on international politics.
Watashi ha kokusai seiji nitsuite kakareta hon wo yomu.
I *topic* international politics about concerned book *obj* read.

(21) Kare ha kokusai seiji nitsuite kakareta hon wo kau.

Here *he buys* from (18) is matched with the same English phrase in (19), and *a book on international politics* is matched with the English phrase in (20). The analysis phase retrieves the corresponding Japanese translations (underlined), which are recombined to produce the translation in (21).

The first question that arises from such an example is: How are the appropriate matches between the source sentence fragment and example sentence fragments to be made?[20] With the assumption that an EBMT system is created from bitexts, as with the other data-driven systems, the matching process revolves around the issue of how long of a sentence fragment to use as a match (absent the case of a perfect match, in which case translation is trivial, see Figure 2.9). The longer the passage that is matched, the lower the probability that the match will be complete. The shorter the passage that is matched, the greater the possibility that the match will be ambiguous and that the resulting translation will be of low quality—since short matches lead to word-for-word-like translations (Nirenburg *et al.* 1993).

EBMT researchers generally look for the longest match possible, then continue the process recursively for the remaining parts of the sentence. Determining the

[20]Note that here I revert back to the more usual terminology of using *source* to refer to the language being translated from.

quality of a match means that an appropriate similarity metric is needed. The choice of such a metric is one of the key differentiating points between EBMT systems, and is related to the form in which the examples are stored (i.e., the *example representation*). In cases where examples are stored simply as strings of text, similarity metrics can be as simple as measuring the number of keystrokes necessary to convert the source string to an example (a process quite similar to calculating the edit-distance, the minimum number of insertions, deletions, or substitutions required to convert one sequence to another, see Kruskal (1999)), relative to the character length of the source string. This keystroke distance can be calculated as:

(22) $Distance = \frac{the\ number\ of\ keystrokes}{number\ of\ source\ sentence\ characters}$ (Nirenburg *et al.* 1993)

A similar method would be to measure difference in terms of words rather than characters. More elaborate metrics can also be used, such as comparing the similarity in terms of morphemes, POS tags, word senses, and the like, but these all require the ability to do this processing and do it reliably.

If examples are stored not simply as text, matching may be more sophisticated, and there may be potential for greater generalization, more accurate recombination, and a reduction in storage requirements. For example, suppose that examples are stored as trees, which of course necessitates additional linguistic processing, even if automatic (here, like in the SMT cases, the MT models benefit from hybrid techniques). Matching using trees allows for comparison of structure as well as words. Such structured storage also makes recombination smoother; it reduces what is called *boundary friction*, where translation fluency is worst where translated fragments are recombined, since structural considerations can be taken into account. This sort of approach was used in many early EBMT approaches (Somers 1999).

If examples are further processed, and stored as templates or specially tagged entries (Brown 1999) or as case frames (Jones 1996), the matching process yields more matches, but sometimes at the expense of quality. For example, to match:

(23) *the dog barks*

one could use a template such as:

(24) $[S \quad [NP_{+animate} \quad][VP_{intrans} \quad]]$

which will yield any number of useless matches such as:

(25) *a child slept*

Thus, matching procedures often involve a combination of such generalization strategies, along with bilingual dictionaries (often obtained from pre-existing outside sources, but sometimes induced automatically from corpora, see Brown (1997)). Moving to even more sophisticated representation of examples, the model of Jones (1996) involves the use of predicate frame structures, using detailed semantic information. For example, for the following sentence:

(26) *The goods were sent to you last Wednesday.*

Jones (1996:40) gives the following representation:

(27) $SEND(X1 : \text{human} : 0 : (X1)_{Agent}$
$(X2 : \text{non-human} : \text{“the goods”} : (X2))_{Goal}$
$(X2 : \text{human} : \text{“you”} : (X3))_{Recipient}$
$[S1 : \text{“last Wednesday”}]_{Time}$

The obvious issue here is that this sort of representation may require exactly the sort of hand-coded rules that data-driven systems seek to avoid, unless the rules too can be induced automatically.

As mentioned, example representation is a key issue not only for matching, but also in terms of storage, generalization, and recombination. Certainly the techniques such as templates and predicate frames reduce storage requirements (an oft-cited problem for EBMT approaches),[21] and not necessarily at the expense of accuracy. Using word-clustering techniques, Brown (2000) was able to reduce the number of examples needed by a factor of four to five, as well as decrease training time to reach a level of accuracy better than that achieved prior to clustering. Thus, as in the SST case (section 2.2.2), the issues of storage and generalization go hand in hand.

With regard to recombination, example representation again plays a central role. As mentioned earlier, representations which capture hierarchical information, such as trees, make recombination easier. Without such representation, it could be impossible to handle word order differences (i.e., the order of the input sentence would always be imposed on the translated fragment). In addition to example representation, the partitioning during the matching of inputs to examples may affect recombination. Partitioning (deciding where the fragment to match should begin and end) at phrasal boundaries may be most effective, but is hard to accomplish without syntactic processing. Veale & Way (1997) propose an interesting technique which involves using a closed-class of function words, called *markers*, to delineate phrasal boundaries for selecting fragments, and justify their approach based on psycholinguistic studies of human sentence processing. Most important is that better partitioning results in less boundary friction during recombination. Another

[21]Grefenstette (1999) explores the idea of using the internet as a source for EBMT examples.

potential method to decrease boundary friction is borrowed from SMT. Translated fragments could be combined, then smoothed via a target language model (Somers 1999).

In summary, EBMT, like other data-driven approaches, is motivated by its robustness, ability to generalize, and lack of a need for elaborate linguistic analysis. It is also highly adaptable to different domains and language pairs, since coverage can be increased or varied by adding or changing examples. One problem apparent in some of the EBMT literature is the lack of a probabilistic model for some of the decisions that need to be made during the matching, analysis, or recombination stages. For example, when gathering statistics for correlations between source language and target language words, Veale & Way's (1997) parameters do not sum to one (i.e., it is not a well-founded probability distribution). While these sorts of flaws may suggest areas where accuracy could be improved, they should not distract from the clear relationship between EBMT and statistical MT.

Although the strengths and the focuses of the two data-driven MT frameworks are different, both learn from parallel texts of examples, and both seek means to align the data so that it can be used to generalize to unseen inputs. In fact, it may turn out to be that combined EBMT and SMT approaches are the best empirical solution, since EBMT systems appear to be better suited for special handling of idioms and contextual, long-distance dependencies, while SMT systems better generalize to unseen sentences. In a provocative recent paper, Marcu (2001) presents a hybrid EBMT/SMT system which outperforms two (unnamed) commercial systems. These sorts of results, along with the potential benefits of incorporating hand-coded data (as suggested in Vogel & Ney (2000)), further suggest that MT architectures should be flexible. This is consistent with the overall plan for the linked automata MT architecture presented in this dissertation, in that it is viewed as an automatically-induced base around which more specialized MT systems can be built.

CHAPTER 3

PRELIMINARIES TO THE MODEL: CORPORA AND ALIGNMENT

> [B]uilding an MT system is an arduous and time consuming job, involving the construction of grammars and very large monolingual and bilingual dictionaries. There is no 'magic solution' to this. (Arnold *et al.* 1994:11).

As mentioned in Chapter 2, data-driven (or *empirical*) MT methods begin from parallel texts, i.e., texts in different languages representing the same content. The motivation for beginning with parallel texts is to use the translation knowledge implicit within them as a substitute for the hand-coding of grammars and dictionaries.

For the linked automata model presented in Chapter 4, these texts must be word-aligned,[1] where sequences of words representing the same content in source and target language sentences are linked. In this chapter, I detail the preparation process, beginning with the choice of parallel texts, then focusing on word alignment. I first introduce word alignment in general (section 3.2.1), then present a crude word alignment algorithm (section 3.2.2), which in turn motivates the use of

[1]Some empirical MT methods do more complex alignments, e.g., hierarchical syntactic alignments, see for example sections 2.2.2.4 and 2.2.2.5.

a better aligner (section 3.2.3) and a hand-aligned corpus (section 3.2.4). Last, I present a method for word alignment evaluation so that the different word alignment algorithms can be compared. This will then set the stage for presenting the linked automata MT model in Chapter 4.

3.1 The Parallel Texts and Their Preparation

Creating a translation system requires a large number of aligned bitexts. An aligned bitext consists of a source string (often, but not always, a sentence—smaller and larger units also occur), its target language equivalent (i.e., its translation), as well as a means of identifying which (possibly empty) sequences of words in the source string are aligned (i.e., correspond) with which (possibly empty) sequences of words in the target. For example, we might represent an English and French word-aligned bitext as in Figure 3.1 (shown earlier as Figure 2.2).

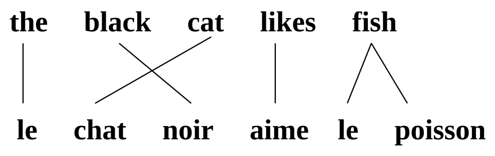


Figure 3.1: An English and French word-aligned bitext

Word alignments are also sometimes presented in the following parenthesized form, using word numbers to avoid ambiguity, with source word number sequences appearing to the left of the colon and target word number sequences appearing to the right for each alignment pair:

(28) the black cat likes fish
le chat noir aime le poisson
(1:1)(2:3)(3:2)(4:4)(5:5,6)

Thus, source (English) word 1 aligns with target (French) word 1, source 2 aligns with target 3, source 3 aligns with target 2, source 4 aligns with target 4, and source 5 aligns with both target 5 and 6.

Before creating a word-aligned bitext data set, a bilingual corpus must be selected. I chose to use parallel translations of the Judeo-Christian Bible.[2] I selected the Bible for several reasons. First, it was freely available. Second, religious works tend to be meticulously translated, which can ease the task, since successful training requires a high degree of correspondence between the source and target texts. Additionally, another benefit to using the Bible is that it allows for perfect sentence alignment, something which is unrealistic for most bilingual corpora, since we can match the texts verse for verse.[3]

Another choice which needs to be made is that of source and target languages. Since I wanted to give the linked automata system its best chance to succeed, at this early developmental stage, I chose two languages which should be relatively easy: I selected English as the source language and Spanish as the target language. These languages exhibit a degree of word-order variation (but not as much as we might

[2]These have been graciously made available by the University of Maryland Parallel Corpus Project, at http://benjamin.umd.edu/parallel/. My thanks to Philip Resnik and his collaborators for their generosity.

[3]The use of such religious works is obviously not without controversy. In this project not only do I manipulate sentences, but I also rearrange words in orders not seen before, creating new sentences. In doing so, it is certainly not my intent to offend anyone, and I apologize if I have done so. My decision to use these works was simply a practical one, and further research will likely make use of additional corpora.

see, for example, with English and German), and sentences are roughly the same length (Spanish tends to be slightly shorter, both in terms of words and characters), providing what should be a more straightforward translation task than we would get with two languages less closely related in terms of type, such as Finnish and Cantonese.[4]

On the English side, the King James Version was used. For the Spanish, the Bible version was the 1909 edition of the Reina-Valera. After downloading the two texts, they were prepared by eliminating any inconsistencies (e.g., missing or transposed verses, etc.) and removing the html tags as well as most (non-word-internal) punctuation, using various Perl scripts created for the process, to ready them for the alignment stage.

3.2 Alignment

When I embarked on the word and sentence alignment part of the project, there were two often conflicting goals: 1) to keep the effort spent on alignment to a minimum and 2) to have the best alignments possible. The reason for trying to minimize the resources spent on alignment was simply that word and sentence alignment are entire research domains in their own right, and were not the focus of this research. But, an important assumption to the entire translation model proposed was that the process begins with accurately aligned texts. Thus, I opted for a strategy that was

[4]It was also of no small benefit that English and Spanish are the author's two best languages.

as straightforward to implement as possible, but that also allowed for a reasonable degree of accuracy. Needless to say, changes in alignment strategy will continue to have ramifications for this work, and it is a possible area for further exploration.

While it is perhaps likely that alignment may be most efficient and most accurate when sentence and word alignment take place concurrently (or at least when a model for word correspondences is generated concurrently with sentence alignment, see Chen (1993)), I chose to first sentence-align then word-align the two texts. I began with an implementation of Gale & Church (1993), a very efficient sentence alignment algorithm based on the number of characters in sentences.[5] While the results were fairly good, it was clear that the overall translation system performance could be only as good as the overall word alignments, which in turn would be only as good as the overall sentence alignments. Thus, as already mentioned, I decided to use the verses in the Bible for sentences, yielding perfect sentence alignment, which I hoped would make it easier to evaluate the overall translation model (i.e., the interest of this research was not so much in perfecting sentence alignment; rather, I assumed an accurate alignment, and wanted to see how far I could go with it in the translation model). To this end, I simply first checked the texts to make sure their verses agreed, eliminated any passages where there were discrepancies, and produced the verse-aligned bitexts.

[5]My thanks to Chris Brew for providing me with his Lisp code implementing the Gale and Church sentence alignment algorithm. For the results reported in this dissertation, the algorithm was not used, however, since Biblical verses formed readily alignable units.

3.2.1 Word Alignment

After sentence alignment comes the obviously more challenging task of word alignment. One possible approach to word alignment, given already aligned sentences, would be to again use a Gale and Church-like approach, i.e., given an appropriate metric and an appropriate dynamic programming algorithm, we could attempt to align words just as we do sentences. Word alignment, however, has complexities not seen in sentence alignment. Certainly, suitable metrics do exist, but word alignment based on the number of characters or phonemes in words is not appropriate, simply because these metrics do not sufficiently distinguish one word from another with enough accuracy for the task.

Before moving to the step of creating a word-aligner, let us first examine some of the types of alignments that can occur. As is also the case for sentence alignment, in word alignment we can see one-to-one, one-to-none and none-to-one (I will refer to these last two collectively as *null* alignments), many-to-one, one-to-many, and many-to-many alignments. A template, showing some typical alignment configurations, is shown below in Figure 3.2.

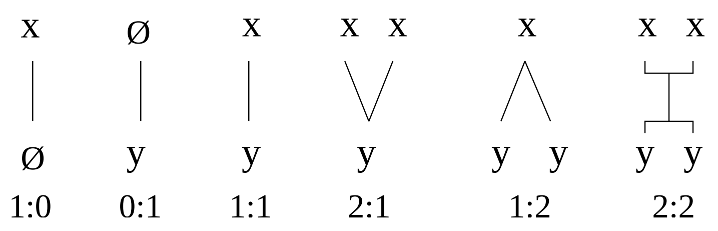


Figure 3.2: Some typical alignments between source (x) and target (y) words

A word alignment has typically been defined as a mapping between the words of a source language string and the words of a target language string (Brown *et al.* 1993).[6] However, because the subparts of word alignments (e.g., the correspondence between *fish* and *le poisson* in Figure 3.1) can also be viewed as mappings between words of source and target strings, there sometimes exists confusion in terminology, specifically, if such 'subparts' should also be referred to as word alignments (or *links, connections, correspondences*, etc.). In this and later chapters, I will attempt to stick with what appears to be the most standard practice in the literature, even though it may be slightly confusing, and refer to these subparts as *alignments*, and reserve the term *word alignment* for the set of all such correspondences in a bitext.[7] For example, in Figure 3.1 and example (28), there are five alignments, which together comprise the word alignment of the bitext (notice that four of the alignments are 1:1 and one is 1:2).

Some other researchers (such as Ahrenberg *et al.* (2000)), use both *alignment* and *link* interchangeably to refer to these subparts (i.e., what I have defined as alignments). I will reserve the term *link* to refer to the individual correspondences within an alignment, following the practice of Melamed (1998). A link is a 1:1 correspondence between words, i.e., a pair (u, v), where u is a source word and v is a target word (one might think of links graphically as the individual lines of an

[6]Brown *et al.* (1993:266) actually describe an alignment "between a pair of strings as an object indicating for each word in the French string that word in the English string from which it arose," thus our notion is somewhat more general.

[7]In many cases, a distinction between the two will not be necessary; but having the two terms should help in the event the meaning cannot be determined from the context alone. *Alignment* and *word alignment*, of course, may also refer to the process of computing a word alignment—in these instances no distinction is needed.

alignment). So, a 1:1 alignment consists of one link, a 2:1 alignment consists of two links, and a 3:2 alignment consists of six links (thus the number of links is equal to the number of relationships between words—this leads directly to the idea of using the cartesian product to get the links for non 1:1 alignments, in section 3.3). Finally, I will sometimes refer to the words of an individual alignment as *anchors*, using *source anchor* for source words and *target anchor* for target words. Having the word alignment terminology squared away, we now return to the discussion.

In a word-aligned bitext, each word in the bitext must be accounted for exactly once. In addition to the different types of alignments shown in Figure 3.2, there are also different types of word alignments, such as a *swapping* word alignment. A swapping word alignment occurs when the links (sometimes these will also be called *branches*) of different alignments cross each other (see Figure 3.3; swapping typically refers to the crossing of two different 1:1 alignments, and often only when the anchors are all adjacent, but I use it to refer to any sort of word alignment where some of the branches cross). Referring again to Figure 3.1, we see a swapping word alignment, where *black* is aligned with *noir* and *cat* is aligned with *chat*, creating the crossing branches.

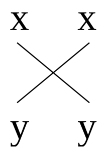


Figure 3.3: A swapping word alignment

While in sentence alignment we might expect to see some of these same sorts of alignments (e.g., many-to-one, one-to-many, as well as one-to-one, two-to-two, and null alignments),[8] the dynamic programming techniques typically used in sentence alignment do not lend themselves to swapping. For example, we might choose to align the first source language sentence with the fourth target language sentence, and also to align the third source language sentence with the second target language sentence. For most applications, disallowing this sort of swapping sentence alignment will not produce poor results. But for word alignment, as we have already seen, allowing such swapping is a must for any pair of languages where the word order is not always the same. In the next section, a very simple word-aligner, which allows for a number of different alignments, including swapping, is developed. One type of alignment which this algorithm will not account for, however, is a discontinuous alignment (see Figure 3.4). A discontinuous alignment is one in which either the source anchors or target anchors in one alignment are not contiguous (i.e., all the words are not adjacent to each other). I will discuss why the algorithm does not allow for such alignments in the next section, and return to the subject of discontinuity in depth in Chapter 7.

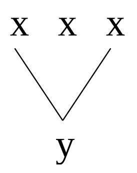


Figure 3.4: A discontinuous alignment

[8]Of course, the possibilities here are endless, but these alignments are some of the most common.

3.2.2 A Simple Word-Aligner

I previously noted that the number of characters or phonemes was not a suitable metric for word alignment. Toward finding a suitable metric, I again looked for a straightforward approach, and chose to base our measures upon word frequency. The basic idea here is that if words co-occur more often than we would expect based on their frequencies in the data, given some threshold, we hypothesize that they are good candidates for alignment. The first step is to gather frequency data as follows: For source word A and target word B, calculate the information in the following table, where k1 is the number of times A and B co-occur in the bitexts; k2 is the number of times B occurs in bitexts where A does not occur; k3 is the number of times A occurs in bitexts without B; and finally k4 is the number of source and target word co-occurrences in all bitexts, where the source word is not A and the target word is not B:

(29)

$k1 : (AB)$	$k2 : (\neg AB)$
$k3 : (A\neg B)$	$k4 : (\neg A\neg B)$

We also use the following counts and probabilities:

(30)

counts:	$n1 = k1 + k3$	$n2 = k2 + k4$	
probabilities:	$p1 = k1/n1$	$p2 = k2/n2$	$p = (k1 + k2)/(n1 + n2)$

Next, knowing that for a task such as word alignment data-sparseness is a predominant problem, we choose as the most appropriate measure Dunning's (1993) *g-score* (also known as *log-likelihood ratio*, *likelihood ratio*, or more accurately *log of the likelihood ratio*), which is defined as follows:

(31) $2 * [logL(p1, k1, n1) + logL(p2, k2, n2) - logL(p, k1, n1) - logL(p, k2, n2)]$

$logL(p, k, n)$ is defined as:

(32) $k * log(p) + (n - k)log(1 - p)$

Returning to the numbers needed for the g-score calculation, there are a number of possibilities. Again, for a bitext of n source words and m target words, for each word of the source s_i and each word of the target t_j we could count the number of times they co-occur in the given bitext. Doing this for all the bitexts, we could construct a table, giving us not only a number for s_i and t_j co-occurrence, but our other numbers needed as well. But what does *co-occurrence* mean? Do we count all words equally? Do we care if the words are in the same relative place in the respective sentence with regard to the length of the sentence? Are we interested in the co-occurrence of word sequences with word sequences, and the like? And perhaps the most practical question, for any corpus, won't such tables become too large to use without the aid of heuristics for shortening them?

As elsewhere, I try to be as practical as possible. First, we count all co-occurrences equally (I save the question of relative distance of one word from another in their respective sentences as a separate measure). The question of word-sequences is answered by considering what sort of alignments we want to allow. Since the most typical alignments (as ascertained by hand-aligning a small portion of the texts) are 1:1 (i.e., 1 source word to 1 target word), 2:1, 2:2, 1:2, 1:0, and 0:1 (see Figure 3.2), we can concentrate our frequency gathering on the 1:1, 1:2, 2:1, and 2:2 cases (leaving the 1:0 and 0:1 cases, known as *deletion* and *insertion*, respectively, to be determined by appropriate penalties during the actual alignment process).

Given these alignments, the counts needed are clear. For example, for the 2:1 case, we count how many times a pair of adjacent source words occurs in the same bitext as a given target word. Note that while we will allow for swapping word alignments, as mentioned, we do not allow for discontinuous alignments (so we would not allow source words 1 and 3 to be aligned with a target word, since they are not adjacent; see Figure 3.4). We did this mainly because we thought that such alignments were rare (this turns out to be an incorrect assumption, see Chapter 7). Additionally, allowing discontinuous alignments would cause the computational complexity of the alignment algorithm, not to mention the size of the frequency tables, to balloon.

As for the heuristic for reducing the size of the tables, we choose to only count co-occurrences that occur within a certain distance window. This coincides with the idea that in a long sentence, even for languages where word order varies, the words that are somehow related will tend to occur in similar relative positions in their respective sentences. For example, even though an English subject might be towards the beginning of the clause in which it occurs (which itself may be embedded in some larger sentence), and a corresponding German subject may occur toward the middle of the relevant clause (inside some larger sentence), the two words still will generally occur within a window of a size much smaller than the length of the respective sentences. For the data that we had, we inspected our results with various window sizes (and checked the sizes of our tables) and chose to use a window size of five.[9] Having made these design decisions, we implemented the frequency

[9]Clearly, this window size is something I could vary to improve the alignment results. However, in the future work on this project, given the dependence on accurate alignment, I will likely end up using different and more effective algorithms altogether, such as the aligner discussed

calculation algorithms in Lisp, and got data for the various alignments from the Old-Testament books Genesis–Numbers, with the idea that we would use these frequencies for aligning sentences from Genesis (i.e., the first book of the Bible).[10]

3.2.2.1 Overview of Word Alignment Algorithm

Our raw materials now in place, there remained the question of the appropriate alignment algorithm to allow for swapping. Again, note that typically dynamic programming algorithms do not allow for swapping, and in any case the increase of possibilities in the search space can be quite expensive. One can think of allowing swapping as the equivalent of allowing crossing alignment branches (see again Figure 3.1), or as the idea that the word sequences of one string float freely over the other, which remains fixed. Alignment consists of finding a means to account for each source and target word, but crucially using each word only once. Our solution was to create a version of the CYK algorithm (the Cocke-Younger-Kasami algorithm, see (Aho & Ullman 1972) for a good explanation) which is pre-seeded with the various alignments that make swapping possible. We borrow an idea from Johnston (1998) (note that I use an analogous approach during the actual translation process, described later in section 4.5.3). Johnston's idea is given in a chart-parsing

in section 3.2.3, and as such did not want to spend too much of my time focusing on the best possible alignment approaches. Also, the proper window size might best be viewed as a parameter to be adjusted depending on the pair of languages involved.

[10]Note that this is not an instance of the well-known machine-learning error of training on the test set, since one might expect when aligning an unseen corpus of bitexts to first go through the corpus and get the frequency data, and use the data to help in the alignment, by either adding it to existing tables or using it to create new ones. It simply means two passes through the corpus instead of one.

context, where the aim is to combine constituents in a way that allows overlapping. He uses what he calls a *multichart*, where edges which can be combined are identified by sets which record the atomic edges from which they were generated. Edges may only be combined when the intersection of these sets is empty. Our adaptation of this technique in the word alignment context works as follows: Suppose we use the target sentence as the base (think of this as if we are trying to parse the target sentence) in our m word CYK dynamic-programming matrix, thus the matrix will have $m * m$ cells. Parsing the target sentence amounts to putting its constituents together in all ways possible, keeping the linear order fixed (note that here we let the source sentence *float*, and keep the target sentenced fixed; we could do the opposite or let both float and would still get the same results). To each array cell, we add a number of what we term *sas-triples*, where *sas* stands for *source-alignment-set*. Each triple represents the source words that the target words spanning the given array indices could be aligned with. As the target is parsed, a new edge can only be put into a cell if the sas-triples of its daughters have a null intersection. The new edge has an sas-triple value of the union of the sas-triples of its daughters. In actuality, we also record the score at each cell, as our alignment algorithm is probabilistic, and record a link back to our alignments (these three entities, the store of source words, the score, and the alignment link, comprise the triple).

For brevity, we will not give our seeding algorithm here, but the basic idea is to seed each array cell with sas-triples, consistent with the alignment possibilities we allow (e.g., since we allow 2:2 alignments, we need to seed all the array cells representing constituents of length 2, but we need not seed any cells which represent constituents of length greater than 2, since we do not allow alignments or more than 2 pairs of words). We do this seeding for all possible source word sequences *within the allowable alignment window.* This last point is important. If, for example, we only allow source and target words within 10 words of each other to be aligned (where relative distance might be measured from the beginning of the sentence), there is no point in seeding the array with sas-triples for source words that are more than 10 words away from the given target constituent. The number of possible alignments increases dramatically with no constraint on the window size, since anything in the source can align with anything in the target, which even if we only allowed 1:1 alignments would mean $n * m$ possible alignments for every bitext. To allow for the 0:1 cases, we also seed the appropriate cells with sas-triples with empty source word stores.[11] The beauty of this seeding approach is that once completed, we can simply parse the target sentence as if it exists on its own. The sas-triples can be viewed as non-terminals, and the intersection test can be viewed as the only constraint on their being put together.

We can best illustrate the seeding process with an example, using the following bitext:

(33) *the cat sleeps*
el gato duerme

In this example, we assume an alignment window of 1, which means that target word sequences can be aligned with source word sequences if the difference in their

[11]I handle the 1:0 cases only in the *last* cell, i.e., the cell containing complete parses, by simply adding any new edges which could be created from existing edges with the addition of aligning a source word with nothing in the target, so long as that source word has not been already used. This is an invisible operation in terms of the parse, since it is independent of the target words.

	j→	el	gato	duerme
i↓		1	2	3
el	1	⟨⟩,⟨*the*⟩,⟨*cat*⟩ ⟨*the cat*⟩,⟨*cat sleeps*⟩	⟨*the*⟩,⟨*cat*⟩ ⟨*the cat*⟩,⟨*cat sleeps*⟩	
gato	2	⟨⟩,⟨*the*⟩,⟨*cat*⟩,⟨*sleeps*⟩ ⟨*the cat*⟩,⟨*cat sleeps*⟩	⟨*the*⟩,⟨*cat*⟩,⟨*sleeps*⟩ ⟨*the cat*⟩,⟨*cat sleeps*⟩	
duerme	3	⟨⟩,⟨*cat*⟩,⟨*sleeps*⟩,⟨*cat sleeps*⟩		

Table 3.1: Seeded CYK array for bitext ⟨*the cat sleeps; el gato duerme*⟩, with alignment window of length 1 (showing only source word stores of each sas-triple)

starting points is less than or equal to 1. Given this constraint, and assuming the alignment possibilities shown in figure 3.2, as well as allowing swapping, we seed the CYK array as shown in table 3.1. Array cell $< i, j >$ stores alignments for target word sequences beginning at i that are j words long. For instance, cell $< 1, 2 >$ is initialized with all the permitted alignments for the 2 target word sequence *el gato*. These alignments include the 1:2 alignments: $<$ *the* $>$ and $<$ *cat* $>$, but not $<$ *sleeps* $>$, because $<$ *sleeps* $>$ is outside of the alignment window; and the 2:2 alignments: $<$ *the cat* $>$ and $<$ *cat sleeps* $>$.[12]

Having properly initialized the CYK array, we parse in the normal way, attempting to combine the sas-triples for adjacent target edges, until we have spanned the

[12]Note that in the figure I only show the source word stores, and not the actual alignments, simply to save space. But during parsing, there are of course other ways to get these same stores, since one can cover the same two source words in the same array cell by using different alignments. For example, a pair of 1:1 alignments, once combined into a single sas-triple, may have the same source word store as a single 2:2 alignment sas-triple. Note also that the implementation uses bit vectors to represent the source word stores, i.e., a bit in the nth position is used the track the presence of the nth source word.

entire target sentence, getting complete parses from the $m * m$th cell. There are two additional considerations, however. We mentioned that the word alignment process is probabilistic, i.e., each sas-triple has a score. How we calculate that score is the first consideration. The second consideration is the very large number of alignment possibilities, notwithstanding the use of a window of allowable alignment distances. We need to use heuristics to keep the computation manageable.

3.2.2.2 Calculating the Word Alignment Scores

In terms of the score calculation, there are a number of factors. First and foremost (and notice that all initial scores are set during seeding; during parsing, scores of the daughter constituents are simply added) is the aforementioned g-score for the alignments as stored in the sas-triple. Probably the next most important factor is the distance of an alignment, where again distance is defined as the difference between the relative starting points of the source word sequence and target word sequence under consideration. Keeping with the intuition that alignments where the distance is great should be less favored, we experimented with a number of different measures and penalties, some which measured distance from the beginning of the relevant sentence and some from the middle. For the results reported later in this dissertation, we simply measured the distance as the difference between the relative starting points when compared to the start of the sentence, then multiplied this distance by a weight. The g-score, as we have defined it, gives us a positive number. If two items (two word sequences, in our application) A and B have a high g-score, they are good candidates for alignment. So, we seek to get alignments which have the highest score. Bearing this in mind, we then subtract the distance score (we call it a *distance penalty*) to get a score for the alignment.

We also adjusted the scores (lowered them) in cases of *deletion* (i.e., where a source word sequence aligns with nothing in the target; thus, it is as if a source word gets deleted during translation), in cases of *insertion* (i.e., where a target word sequence aligns with nothing in the source), and, for alignments which we considered to be less desirable than 1:1, namely 2:1, 1:2, and 2:2, we also gave penalties. Thus, from an initial g-score (0 in the case of insertions and deletions), we subtracted the various penalties, each of which were multiplied by weights. We hand-tuned the weights to a point where alignment appeared stable, and within a range of acceptability. Even before we had created a hand-aligned set for comparison (see section 3.2.4), however, we could see that the alignments produced were unlikely to be good enough for translation model testing (this intuition is borne out with word alignment evaluation, see section 3.3). Clearly, if we were to continue to develop this word alignment algorithm, it could benefit from more principled parameter tuning.

There are also a number of other relatively simple techniques which would likely greatly improve the performance. For example, issues of data-sparseness can be alleviated by using lemmatization—thus we might align lemmas rather than words. Of course, such a step would require the existence of an appropriate lemmatizer. Similarly, one might have special treatment for function words, assuming such high-frequency words could be easily identified. Additionally, similar metrics to the g-score measure could be employed to identify word collocations, which left unidentified could lead to translation errors later on. Again, as mentioned in Chapter 2, the degree to which such techniques are employed may affect how automatic the alignment methods, and consequently the MT methods, are.

3.2.2.3 Reducing the Word Alignment Search Space

The remaining problem we faced was the still very large number of alignment possibilities that the CYK algorithm faced, even after the constraints on allowable alignments we had imposed. Given that the verses were often over 20 words long, the dynamic programming array cells would quite quickly fill with alternative word alignments, since we allowed alignments of differing ratios, swapping, and insertions and deletions. The solution was simply to decrease the search space, using a hill-climbing strategy. At each array cell, we limited the number of sas-triples, by keeping only the n-best scoring triples. We determined a good value for n empirically, one which was large enough so that it seemed to have no adverse effect on the alignment, and small enough so that the alignments could proceed in a reasonable amount of time.[13] We arrived at an n calculated as the lesser of twice the number of words in the source sentence plus five, or 40, since numbers over 40 tended to bog down the system.

Having created a rather informally designed word alignment system, but one with all the alignment possibilities we initially thought required, including swapping, we went ahead and aligned the first four books of the Bible (i.e., Genesis, Exodus, Leviticus, and Numbers), yielding nearly 5,000 word-aligned bitext triples, each with an average of 50 combined source and target words.

3.2.3 Using a Better Word-Aligner

Although the creation of the word aligner described in section 3.2.2 (we will refer to this as the CYK aligner) allowed for a word-aligned corpus to be quickly constructed,

[13]I do not report alignment times here since it was an off-line task for the project.

manual inspection of the word alignments produced showed that they were less than ideal (a more quantitative evaluation will be presented in section 3.3). Thus, once the earliest development stages for the linked automata model were completed, I began to look for good word alignment implementations which were freely available, so that I could quickly make other data sets for experimentation with the model.[14]

It turns out that a good word-aligner was very close at hand. Section 2.2.1 introduced the IBM statistical MT model (Brown *et al.* 1993), which is a word alignment model. The IBM model was well-known for its (relatively) good performance, but although an overview of the mathematics behind the model were provided in Brown *et al.* (1993), the presentation was rather difficult, and researchers had difficulty implementing the complicated model. Partially because of this difficulty, a number of leading statistical MT researchers got together for a workshop in the summer of 1999, at Johns Hopkins University (Al-Onaizan *et al.* 1999). These researchers implemented most of the IBM model (up to Model 3), with the exception of producing a decoder (thus the implementation could produce word alignments, but still lacked the decoder needed for translation). They called this system *Giza*, and made it freely available to other MT researchers. One of the researchers, Franz Josef Och, continued to develop the system, including adding the code necessary to implement IBM Models 4 and 5, as well as the use of word-classes. This, system, called Giza^{++} (Och & Ney 2000), would be expected to yield better word alignments than Giza, and was also made freely available for research.

[14]Dan Melamed was instrumental in pointing me to word alignment resources, and Franz Josef Och kindly answered a number of my questions regarding the use of his word aligner, Giza^{++} (Och & Ney 2000).

The implemented IBM models, used as a word aligners, are not without limitations. For example, alignments show a high degree of directionality, in that only certain types of alignments are permitted: All of the alignments are of the form 1 source word to any number of target words. This means that certain types of alignments are ruled out, such as 2:1, 2:2, etc. (i.e., any alignment with more than one source word), but other types of interesting alignments are permitted, including 1-to-many alignments, where the target words could be discontinuous (i.e., non-adjacent, see Figure 3.4). Most important, because the parameters of the IBM model are tuned using maximum likelihood estimation, as opposed to hand-tuned as in the case of the CYK word-aligner, the Giza^{++} implementation of the IBM model should produce better word alignments. This intuition was supported by the testing against a gold-standard data set (see sections 3.2.4 and 3.3), and allowed for a second data set to be created. In the results in this dissertation, this set will be referred to as the Giza (or Giza^{++}) data set. It should be emphasized that these word alignment results were still less than ideal, and showed a number of systematic errors, but the addition of a second data set proved beneficial for the overall research program.

3.2.4 Hand Aligning

As mentioned in the previous two sections, I wanted to have a gold standard set of word alignments, so that I could assess how well different automatic aligners were performing. Knowledge of the word alignment accuracy was important in terms of determining if the MT model would produce more accurate translations with more accurate word alignments, as would be expected. Assuming that this was the case, accuracy could then be used to determine which word alignment algorithms should be utilized for training. A gold standard of hand-aligned data allows for the automatic evaluation of any number of word alignment algorithms. A second benefit of a gold standard is that, once created, it can be be used as a resource for a number of tasks in linguistics, such as word-sense disambiguation, translation, or searching for cross linguistic lexicalization patterns (Melamed 1998). But, of course, the creation of such a gold standard is an extremely tedious and time-consuming process.

The problems associated with creating a gold standard word-aligned data set are many. In addition to the effort required, tools must be created for the process, bitexts must be selected and prepared, and most important, clear standards for the annotation must be set and adhered to. Although it is true that there is not always one correct word alignment for a bitext, the situation is not nearly as difficult as in machine translation (where there can be many correct translations). There is much less ambiguity in word alignment because the words in the bitext are fixed; only the mapping between them can vary. Once certain conventions for aligning are agreed upon (for example, whether to allow discontinuous alignments, how to treat the alignment of pronouns and verbs in one language with pro-dropped verbs in another, i.e., should the pronoun be aligned with the verb or not with anything, etc.), annotators will typically agree to a strong degree what the correct alignment is (for example, in Melamed (1998), inter-annotator agreement in the word alignment of content words was typically over 90%).

Probably the best-known and most ambitious word alignment gold standard project is the Blinker (bilingual linker) project of Dan Melamed (Melamed 1998).

Melamed paid seven annotators to word align various parts of 250 verses of English and French versions of the Bible (he selected the Bible for the same reasons it was chosen here, availability and ease of using verses as pre-aligned units). To make the process easier for the annotators, and thus increase his chances of having useful results, Melamed created a graphical alignment tool, as shown in Melamed (1998).[15]

For my research, I did not have funding to pay annotators, and I wanted to have a relatively large data set (over 1,000 bitexts). So, I wanted to make the process as smooth as possible, knowing that I would have to do the aligning myself. Dan Melamed graciously made the his graphical alignment tool available to me, but he suggested that its implementation was bug-prone, and hard to adapt, so that I might be better off creating my own. I took his advice, and with the help of Chris Brew (who made the graphical module of the tool), made what is essentially a copy of Melamed's Blinker tool, using Lisp instead of Java. A screen shot of the tool is shown in Figure 3.5.

The tool presents each bitext from a previously made file, where the file consists of parenthesized bitext triples, i.e., a list containing a source sentence, a target sentence, and a possibly empty list of numbers indicating the word alignment. The tool prints the two sentences in full at the top, and the words of the source on the left and the target on the right (see Figure 3.5). Words are aligned by selecting unaligned words from either column, then a user may either click “L,” “Link,” or type “l” on the keyboard, to draw the lines indicating the alignment. Clicking on the “0” on the left will draw lines from selected source words to the 0 bar, indicating

[15]Attempting to hand-align without a graphical tool, i.e., using a text-based tool where word numbers are selected, is a process too tedious to attempt, as evidenced by my own inability to continue in such a task after just a few bitexts.

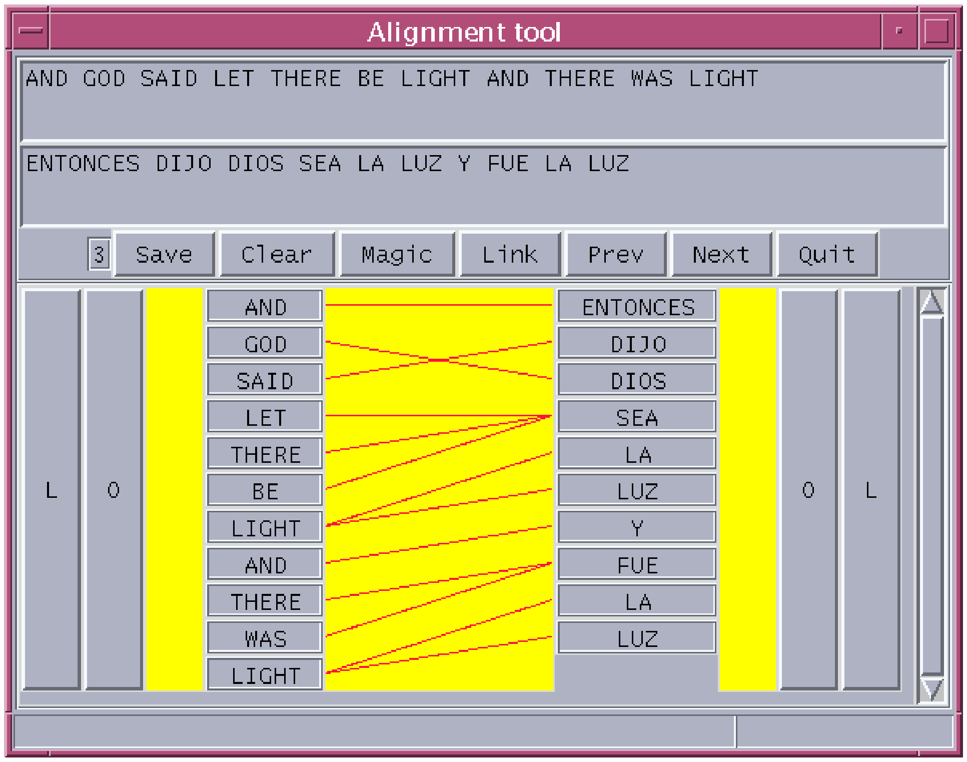


Figure 3.5: The word alignment tool

a source word null alignment, and clicking on the right “0” proceeds analogously for target word null alignments. Only unaligned words may be selected, and selecting a word changes the color of the word’s box. Users may also click on the lines which indicate alignments, which will remove all the lines associated with the alignment (and make the words selectable again). Alignments which are not 1:1 are indicated by the cartesian product of lines, (e.g., a 3:2 alignment would show six lines; clicking on any of these removes all six).

The alignment tool was designed as a forced-choice task. This means that users must account for each word before they can go to the next bitext (or return to an earlier one). This helps to avoid errors (i.e., neglecting to align words) and mimics the behavior that one would expect from a word alignment algorithm. More importantly, it avoids having to distinguish between the failure to make a link, versus no link (a null alignment), which can be a problem for word alignment systems (Ahrenberg *et al.* 2000). Most of the remaining buttons shown in Figure 3.5 should be self-explanatory, with the exception of the "Magic" button. The "Magic" button was added to allow a programmer to add a useful operation which the tool does not provide. In this instance, it was implemented as a semi-automatic aligner: Clicking on "Magic" causes the tool to make all the possible 1:1 alignments using only unselected words, which can be a tremendous time-saver. An alternative idea would have been to have "Magic" call an automatic word-aligner, if one was found of sufficient quality, so that such an automatic alignment would prove to be a time saver.

Although in retrospect it may have been more useful to have selected verses at random, and thus get a better sampling of the different types of texts in the Bible, I chose to do the word-aligning sequentially, from the beginning. I hand-aligned English and Spanish versions of the entire book of Genesis, which consisted of 1531 verses. My goal was to do the alignment as fast as possible, but in a consistent manner. I am a native speaker of English, and, although my Spanish is not fluent, typically the word alignments to make were clear; when in doubt I consulted with native speakers. The project took close to a month to complete, and I suspect that there are a number of errors in alignments (mainly places where the alignment style varied), given my hurried pace. The most important guideline that I followed was that no discontinuous alignments were allowed. To force this to be the case, I made such alignments impossible for the tool to accept (this requirement can, of course, be easily relaxed, for future projects). As was the case for the CYK automatic aligner (section 3.2.2), at the onset of the hand-aligning task, I thought that this was a reasonable thing to do. Such alignments seemed quite rare for English and Spanish, and I wanted the capabilities of the tool to be similar to automatic aligners, which I supposed did not make discontinuous alignments. I was wrong on both counts. Aligners such as Giza^{++} (section 3.2.3) allow for such alignments, and discontinuous alignments do indeed occur occasionally in the alignment of English and Spanish, as shown in a fragment from the task in Figure 3.6. Nevertheless, having begun the task with this style in mind, I stuck with it. To align in instances where discontinuity was clearly the best choice, I simply made a larger alignment, as in Figure 3.7, which gets the information correct, but is less specific.[16]

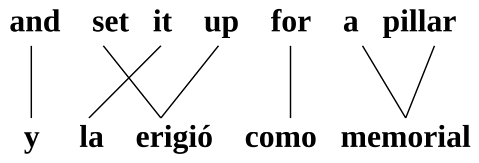


Figure 3.6: A discontinuous English to Spanish alignment from Genesis

In addition to this no-discontinuities alignment style, a number of other alignment style decisions were made, such as choosing to align an English pronoun and

[16]Such a strategy could be taken to the ridiculous extreme of selecting all the words in both sentences and linking them in one giant alignment. Such extremes never occurred in the hand-aligned corpus, and were generally limited to just a few words on either side.

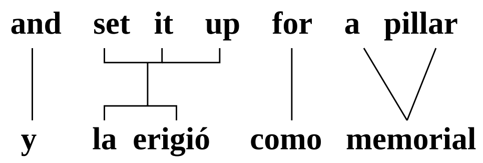


Figure 3.7: Coping with discontinuity when it is not permitted

verb with a Spanish pro-dropped verb, e.g., I would make a 2:1 alignment of *he talks* and *habla*, usually trying to let semantics be my guide (i.e., align those segments which carry the same meaning). As pointed out by others (Melamed 1998), translations in the Bible are sometimes rather far apart from each other, perhaps because they are translations from a third (or even fourth) original language, so at times passages were difficult to align. In the worst (very rare) cases, these would necessitate several null alignments. All in all, however, the alignments should be fairly consistent, and were much better than what one could expect from a automatic aligner, notwithstanding the discontinuity issue, and therefore could fulfill their purpose as a very low-cost gold-standard.

3.3 Word Alignment Evaluation

Word alignment evaluation is a surprisingly fuzzy area in the literature. With the notable exception of Melamed (1998), most word alignment evaluation (WAE) is either done by people (i.e., human judges, as in Och & Ney (2000)), or by automatic measures which appear to be somewhat ad hoc (see for example the techniques of Ahrenberg *et al.* (2000), to be described below). Human judging seems a very reasonable thing to do, but is typically limited in scope (e.g., Macklovitch & Hannan

(1998) limit themselves to 50 sentence pairs), hard to replicate without the funding for judges, and must be redone for each new test. Automatic methods usually involve comparing the results to a hand-annotated gold standard, such as that described in section 3.2.4, but many of the methods seem unclear.

Some of the confusion in WAE probably stems from the fact that word alignment is typically associated with machine translation, where evaluation is a notoriously difficult task (see the discussion in Chapter 5). The crucial difference between WAE and MT evaluation is that, given a fixed alignment style, there is typically something which is (at least arguably) the best alignment, while in MT there is generally no one translation which can be identified as best. The sources of problems in automatic WAE center on three main issues: 1) how to deal with missing alignments (i.e., when an aligner has not indicated how a given word should be aligned, sometimes also referred to as *missing links*), 2) how to appropriately weight and measure alignments which are not 1:1, i.e., which may be many-to-many, and 3) what overall measures to use (e.g., precision, recall, perplexity, etc.). In this section, I hope to put an end to this confusion, by establishing a principled WAE metric for automatic evaluation against a gold standard, after first describing how each of the three issues should be dealt with.

To deal with these issues, we will first review the terminology and notation. In this section, I will show word alignments as lists of pairs of integer sequences separated by colons, such as as shown in (28) earlier, and (34) below, where the left side of the pairs indicates source words and the right indicates target words, and where $\emptyset$ indicates a null alignment, i.e., that side of the alignment is empty. The integers unambiguously represent words at the indicated positions in the source and target sentences, respectively, and for the discussion in this section, no reference to the actual words is necessary.

(34) ((1:∅) (2,3:1,3) (4:2) (∅:4) (5:5) (6:∅) (7:6))

Continuing with the terminology laid out earlier, I will try to refer to the all the alignments (e.g., the entire list in (34)) as a word alignment, and the individual lists separated by colons as alignments. Also, I will again refer to the individual mappings between pairs of source and target word numbers as links. So, in the word alignment of (34), source words 2 and 3 are aligned with target words 1 and 3, and there is a link between source word 2 and target word 3 (and also between source 2 and target 1, source 3 and target 1, and source 3 and target 3, and so on for the other alignments; in the discussion, source word numbers are always given first).

When making word alignments, whether by hand or automatically, it only makes sense to align a given word once, i.e., no word can participate in more than one alignment. This should make intuitive sense: For a word to participate in more than one alignment, it would have to behave as if there was more than one instance of the word. A word alignment in which all of the words are used exactly once is said to be *conservative*. For example, using a hypothetical three word source sentence and two word target sentence, the following are examples of some possible conservative word alignments:

(35) ((1:1)(2,3:2))
((1,3:1,2)(2:$\emptyset$))
((1:$\emptyset$)(2:$\emptyset$)(3:1,2))

and the following word alignments are not conservative (*non-conservative*):[17]

(36) ((1:1)(1,2,3:2))
((1,3:1,2)(2:2))
(($\emptyset$:1)(1:$\emptyset$)(2:$\emptyset$)(3:1,2))
((1:1)(3:2))
((1:1)(2,2,3:2))

Armed with this notion of conservativity, we can deal with the first issue raised: How to deal with missing alignments (i.e., missing links). My answer is rather simple. I am only interested in conservative alignments. That is to say, any word aligner that is useful must account for every word once and only once. While it is true that some researchers might get use out of word aligners which produce non-conservative alignments (for example, perhaps only content words could be aligned; this could still prove useful for construction of a bilingual lexicon), and also true that humans when doing aligning are not always sure what choice to make (and may therefore not make a choice, or make more than one choice, if permitted), any practical word aligner, especially one meant to be central in an MT system, must be conservative. Certainly, I think most researchers would agree that no word

[17]Once might call non-conservative word alignments *deficient*, but that could create confusion with the use of the term by Brown *et al.* (1993), where deficiency refers to the generation of impossible strings in alignments (e.g., more than word in the same place), so I will stick with non-conservative.

token should be allowed to be used more than once in any word alignment.[18] The question of whether to allow missing alignments raises more questions. We might call word alignments which use each word no more than once but do not use all of the words, *incomplete*. To allow for a distinction between null alignments (a word is aligned with nothing) and missing alignments (the aligner did not make a choice) is a failure to complete the original task.

The goal of word alignment is to indicate which portions of bitexts are translations of each other, i.e., which subparts represent the same content. Something either is a translation of something else or it is not. While translations may be unclear, imprecise, or (fully or partially) inaccurate, allowing for missing alignments is tantamount to conflating the notion not being sure of the answer to a question with not being sure if a question was asked. In word alignment, the question is always asked, and therefore a choice must always be made, even when the correct answer is not clear. For this reason, most automatic aligners and most of the gold standards (e.g., Melamed (1998)) use a forced-choice task: Aligners must indicate if something was translated or not. One automatic aligner which does not always make such a choice is the PLUG aligner, and, as expected, this makes evaluation of the alignments rather difficult, at best (Ahrenberg *et al.* 2000). Thus, the WAE measures I will present shortly are intended for conservative word alignments. Notwithstanding my strong feelings on the subject, I will, however, also provide a measure which may be used for incomplete alignments.

[18]Note that this crucially different from the notion of *fertility* (Brown *et al.* 1993), which is related to generation of words, rather than their alignment. Word alignments produced by systems such as those in (Brown *et al.* 1993; Och & Ney 2000) still use each word exactly once.

Another related issue is how to treat alignments given with certain confidence levels. For example, one might create an automatic aligner which is conservative, but provides with each alignment a confidence level, say between 0 and 1. While I am unaware of automatic aligners which do this, certainly some human alignment projects have used similar set-ups. For example, in Och & Ney (2000), annotators had to specify either S (*sure*) or P (*possible*) for the alignments they gave. Such confidence levels can be viewed as added information available to WAE systems, if desired. The measures which I will give below can make use of confidence levels, simply by adjusting the weights for each alignment, to behave as if more or less words existed in the given bitext. This step should be clear once the WAE method is presented, and I will not pursue it further here since such confidence levels are not used in any of the alignment methods employed in this project.

Once we have made the decision to consider only conservative alignments, the second issue, how to weight and measure alignments which are not 1:1, becomes much easier. Let us begin by looking at how others have handled such alignments. An alignment may be partially correct. For example, suppose a correct alignment in a sentence is (1,2:1,2), and that an aligner returns as one of its alignments (1:1) (the question of how many words are in the source and target sentences is not important for this example). There is an intuitive sense that such an alignment is at least partially correct, in that (1,2:1,2) encodes the idea that source word 1 and target word 1 somehow correspond, but not as directly as does the alignment (1:1). A first approximation to a solution might be to count such alignments less than fully correct alignments. This is done in Ahrenberg *et al.* (1999), by multiplying the count of such partially correct alignments by .5 in a precision measure. Such a solution, while having the benefit of simplicity, takes into account neither how much of the alignment is correct, nor how many words the alignment accounts for.

Ahrenberg *et al.* (2000) attempt to remedy these problems with the following measures:

(37) $$Q = \frac{C_{src} + C_{trg}}{max(S_{src}, G_{src}) + max(S_{trg}, G_{trg})}$$

$$recall = \frac{\sum Q}{n(I) + n(P) + n(C) + n(M)}$$

$$precision = \frac{\sum Q}{n(I) + n(P) + n(C)}$$

Where $C_{src/trg}$ is the number of overlapping source (target) tokens in (partially) correct alignment proposals, $S_{src/trg}$ is the number of source (target) tokens proposed, $G_{src/trg}$ is the number source (target) tokens in the gold standard, and n(I/C/P/M) is the number of incorrect, correct, partially correct, and missing alignments, respectively.[19]

First of all, notice that the only difference between precision and recall is $n(M)$, the count of missing alignments. This will have a part to play in the discussion of the appropriate measure which will come shortly. Second, note that the number of missing alignments $n(M)$ and the number of incorrect alignments $n(I)$ (apparently) may overlap, i.e., if the correct alignment is (1:2), and (3:2) is proposed by the system, the target word numbered 2 is counted twice, once as part of a missing alignment and a second time as an incorrect alignment. So the recall measure is deficient (i.e., the probability mass is not correctly spread over the data which make it up). More importantly, the Q measure does not "distinguish between direct and

[19]Recall that for Ahrenberg *et al.* (2000) the word *link* means the same as our *alignment*; I have substituted alignment for link here, where necessary, to avoid confusion.

indirect links" (Ahrenberg *et al.* 2000:1259). This is a serious error, because it means that, given a two-word source sentence and a three-word target sentence, where the correct word alignment is:

(38) ((12:123))

Ahrenberg *et al.*'s (2000) precision and recall measure will give perfect scores to any of the following word alignments (plus others):

(39) ((1:1,2)(2:3)) ((1:3)(2:1,2)) ((1:2)(2:1,3))

In fact, Ahrenberg *et al.* (2000) give the following example of a reference English to Swedish 2:2 word alignment:

(40) ((dangerous goods: farligt gods))

where *farligt* means *dangerous*, and *gods* means *goods*, and point out that if a system proposed the following word alignment, it would be treated as perfect:

(41) ((dangerous:gods)(goods:farligt))

This is a fundamental error. Given the correct (40), an automatic WAE system has no way to distinguish, however, between the incorrect (41) and the much better (42) below. This is because an automatic system has no means to determine which partial matches make sense and which do not. It only has the means to determine which partial matches are possible. The WAE metric to be proposed will therefore count both (41) and (42) equally (each would get a score of .5), as neither perfect nor completely wrong alignments, as will be explained shortly.

(42) ((dangerous:farligt)(goods:gods))

What then is a correct way to weight many-to-many alignments, and to measure partial matches? Let us begin with the most basic case, a 1:1 alignment. A 1:1 alignment connects one source word to one target word. It therefore accounts for two words of the bitext. We can assign a weight to this link (remember that a link is equivalent to a 1:1 alignment) of 1, which seems straightforward enough; one basic link has a count of 1 (nothing hinges on this number being 1, it could be any number, so long as we weight other alignment types appropriately, relative to this basic type). If the link has a value (a weight) of 1, how much weight does each word contribute to it? The answer is .5 . This sort of weighting is appealing, because it is based wholly on the number of words and therefore the number of potential links. The important point to remember is that we want every conservative word alignment of a given bitext to have the same total weight. This means there can be no under or over-counting of the correctness of a given bitext, and also that a bitext will only contribute its appropriate share to a larger evaluation.

Suppose we have the following word alignment, for four word source and target sentences:

(43) ((1:1) (2:2) (3:3) (4:4))

There are 4 alignments (and 4 links, since all the alignments are 1:1), and 8 total words (4 source words and 4 target words). Each alignment contributes 1 unit of weight; therefore, the entire word alignment has a weight of: $1 + 1 + 1 + 1 = 4$, i.e., the total weight of the alignments is equal to the number of 1:1 links (in this case), and in all cases (as will be shown), equal to the number of source words (m) plus the number of target words (n) divided by 2, i.e., the total weight for a word alignment is:

(44) $total\ weight = \frac{m+n}{2}$

Each word contributes exactly .5 weight (e.g., 8 $words \times .5 = 4$). With these two notions, 1) that each word contributes .5 weight and 2) that the total weight for a word alignment (or for an individual alignment) is equal to the sum of the number of word tokens divided by two, we have all we need to handle all of the alignment types, as well as partial matches.

What does this mean for 1:0, 0:1, many-to-one, one-to-many, and many-to-many alignments? The thing to remember is that it is always the number of words that matters. Thus a 1:0 (or 0:1) alignment should contribute .5 weight only. The total weight in the word alignment below, with all null alignments, is still: $.5 + .5 + .5 + .5 + .5 + .5 + .5 + .5 = 4$, the same as in (43).

(45) ((∅:1) (∅:2) (∅:3) (∅:4) (1:∅) (2:∅) (3:∅) (4:∅))

To illustrate how things work with the larger alignment types, let's consider two different alignment cases of five total words, the first with 1 source word aligned with 4 target words, in (46) below, and the second with 2 source words aligned with 3 target words, in (47):[20] Since both alignments have a total of 5 words, they must each contribute 2.5 units of weight.

(46) (1:1,2,3,4)

(47) (1,2:2,3,4)

[20]Remember that these might be part of larger word alignments. The distinction is not so important though, since the properties regarding the distribution of weight are consistent at whatever level we look, alignment level, word-aligned bitext level, or entire parallel corpus level: Each word contributes .5 weight.

Now, let's look at the relationships in each alignment. In (46), the single source word is related to (i.e., *linked with*) four different target words, that is, in a graphic representation, we would see four lines drawn from the source word, one to each of the four target words. So, we might say there is the following set of links:

(48) {(1:1)(1:2)(1:3)(1:4)}

If we viewed (48) as a word alignment, however, it would be non-conservative, because the source word is participating in more than one alignment. If we could make this alignment behave like a conservative alignment somehow, we would have the partial matching problem solved, since it would amount to comparing 1:1 alignments, which is easy to do. Here is the trick: Assign to each link a portion of the total weight that the original alignment had (i.e., pretend that we are breaking the words up into pieces, giving each piece a little bit of the word's weight to give). How much weight each link should get is simply the total weight divided by the number of links, where the number of links is just the number of source words times the number of target words. So, if we again use m for the number of source words, and n for the number of target words, each link gets:

(49) $$weight\ for\ each\ link = \frac{total\ weight}{number\ of\ links} = \frac{\frac{m+n}{2}}{m \times n} = \frac{m+n}{2mn}$$

The formula in (49) will hold for all alignments, except for null alignments (where the denominator will be 0). For null alignments there is nothing to compute, since we have already stated that null alignments have a weight of .5 (i.e., they contribute just the weight of the one word they account for). Computing the links that exist is simply a matter of taking the cartesian product of the alignments. Thus, returning to the 2:3 alignment case of (47) earlier, we get the following six links, each with $\frac{m+n}{2mn} = \frac{2+3}{2\times2\times3} = \frac{5}{12}$ weight (which again adds up to 2.5 weight, as in the 1:4 case):

(50) {(1:2)(1:3)(1:4)(2:2)(2:3)(2:4)}

I call this operation of computing the cartesian product of an alignment to get the links and the weight for each link *flattening*. Flattening so as to distribute the weights evenly to 1:1 alignments (so that comparison is easy) is very similar to the technique used by Melamed (1998), although he goes through several steps of weighting and re-weighting links, and averaging agreement rates. The method given here is much simpler to compute. To my knowledge, (Melamed 1998) is the only other WAE effort which appropriately weights the individual links of alignments.

With flattening so defined, comparing any two conservative word alignments of the same bitext is trivial. We simply take each alignment, flatten it, so that all the remaining links are either null (i.e., 1:0 or 0:1) or 1:1, and see how much of the weight agrees. We can define a symmetric comparison measure, called *Word Alignment Agreement (WAA)*, as follows:

(51) $$WAA = \frac{total\ agreeing\ weight\ of\ word\ alignment}{total\ weight\ of\ word\ alignment}$$

Computing WAA is easy for any pair of conservative word alignments from the same bitext: First, if necessary, flatten the alignments (thus re-weighting them; I will show weights under each link for convenience below), optionally sort the flattened alignments in some canonical order (this is not necessary, but makes viewing the differences easier),[21] and count the weight that agrees (e.g., if one link of weight .25 matches another link with weight .67, then .25 of the weight agrees). From this total agreeing weight number, one can calculate a WAA score for the bitext by dividing by the bitext's weight. For the entire corpus, the sum of the agreement weights for each bitext divided by the total weight of the corpus gives a global WAA score for two sets of word-aligned bitexts.

To make things clear, here are some examples, beginning with five different word alignments for the same bitext of three source words and three target words:

(52) A:((1:1)(2:2)(3:3))
B:((1:2)(2:1)(3:3))
C:((1,2:1,2)(3:3))
D:((1,2:2,3)(3:1))
E:((1:∅)(2,3:1,2,3))

Next are the same five word alignments in the same order, but flattened, with the weights for each underneath (which each sum to three):

(53)

A:	{(1:1)	(2:2)	(3:3)}				
	1	1	1				
B:	{(1:2)	(2:1)	(3:3)}				
	1	1	1				
C:	{((1:1)	(1:2)	(2:1)	(2:2)	(3:3))}		
	1/2	1/2	1/2	1/2	1		
D:	{((1:2)	(1:3)	(2:2)	(2:3)	(3:1))}		
	1/2	1/2	1/2	1/2	1		
E:	{((1:∅)	(2:1)	(2:2)	(2:3)	(3:1)	(3:2)	(3:3)}
	1/2	5/12	5/12	5/12	5/12	5/12	5/12

[21] I have already been presenting alignments in a canonical order, ordered first by source number then by target number, where ∅ precedes any number. Ordering will prove useful for the metric I will mention for incomplete word alignments.

	A	B	C	D	E
A	1	1/3	2/3	1/6	5/18
B	1/3	1	2/3	1/6	5/18
C	2/3	2/3	1	1/3	5/12
D	1/6	1/6	1/3	1	5/12
E	5/18	5/18	5/12	5/12	1

Table 3.2: Word Alignment Agreement (WAA) scores for the five word alignments, A-E. 1 is perfect agreement, 0 is no agreement.

The WAA scores (i.e., the agreement rates) for each of the five word alignments are shown in Table 3.2.

In presenting WAA, we have now also covered the last WAE issue of what overall measure to use. WAA gives a number between 0 and 1 for comparing any two sets of word alignments, with 0 being least correct (i.e., assuming we are comparing against a gold standard) and 1 being perfect. The measure is symmetric (for conservative alignments), and easy to compute. While many other researchers doing automatic evaluation use precision and recall (e.g., Melamed (1998); Ahrenberg *et al.* (2000); Ahrenberg *et al.* (1999)), which are popular measures in information retrieval, word alignment evaluation with conservative word alignments only requires one measure, because precision and recall amount to the same thing (because all conservative alignments of the same bitext have the same total weight; thus the denominator in precision and recall measures, which is the only part where they differ, is identical).

I also mentioned that I would provide a measure for incomplete word alignments (more specifically, those where any missing alignments and null alignments have been removed from the calculation), even though I do not think they should be used. For conservative alignments, another, more elaborate way of calculating the WAA (i.e., rather than iterating through one list of links while checking the other for matches) would be to calculate, after flattening and sorting the alignments, the weighted edit-distance between them. In this case, insertions and deletions cost the weight of the given link, matches cost the difference in weights, and substitutions cost the total of the two weights, i.e., an insertion plus a deletion. Edit-distance is overkill here, because the two things being compared are not sequences, but rather aligned sequences where the order no longer matters. Since, however, they can be put in a canonical order, and no two items can repeat, edit-distance happens to yield the same numbers as WAA (which again can be calculated more easily), when we convert to an accuracy score as follows, called *wa-edit-agreement*:

(54) wa-edit-agreement$= 1 -$ (edit-distance/worst-possible-edit-distance)

The worst possible edit distance for a word alignment of m source words and n target words is $m + n$ (i.e., if the total weight is wrong, then the total weight must be substituted; $m + n$ is twice the total weight).

If we remove all the null alignments from two word alignments of the same bitext (thus avoiding having to distinguish between missing and null alignments, for those who have such word alignments), we can no longer use the WAA measure, because the two word alignments are no longer guaranteed to have the same total weight.[22] We can however, use the wa-edit-agreement measure, for each aligned bitext, and for an entire corpus (using the same measure in (54), with the total of the edit-distances and worst-possible-edit-distances). This measure gives a reliable indication of how much word alignments agree, even when one has not aligned all the words (because it will still cost the price of an insertion or a deletion to match a given link with a missing one). Again, I do not find this to be a necessary measure, but others may.

I would like to make one final comment on WAE. As will be shown to be true for machine translation evaluation as well (in Chapter 5), automatic WAE is just one type of word alignment evaluation. There are other aspects of a word alignment which one might want to evaluate, either automatically or by hand, and Ahrenberg *et al.* (2000) do a good job of identifying these aspects. For example, one might want to measure how well individual words are handled (i.e., how consistent their alignment is), or one might want to weight certain types of words higher or lower in alignments (for example, to decrease the impact of function words on scores). Another aspect which could be evaluated is how well the word alignment serves to create a bilingual lexicon (e.g., measuring both its accuracy and its size). These aspects, as well as aspects related to the efficiency of word aligners, would be interesting to evaluate, but are beyond the scope of this foray into automatic WAE.

[22]With different total weights, the WAA measure is no longer symmetric. Symmetry is a property desirable for agreement measures which edit-distance possesses.

3.3.1 Evaluating the Word Alignments Produced

So, having made it to a reasonable WAE metric, called Word Alignment Agreement (WAA), it is time to use it. As described in section 3.2.2, I used an automatic word-aligner (the CYK aligner) to align English and Spanish versions of Genesis, hypothesizing that the word alignments were not very good. I then used the better aligner described in section 3.2.3, Giza^{++} (Och & Ney 2000), on the same data. Last, (section 3.2.4) I hand-aligned all of Genesis. As mentioned in those sections, some of the alignment particulars varied, for example, neither the CYK aligner nor the hand-aligner (the "gold standard") allowed discontinuous alignments, and the types of alignments allowed in all three also varied. CYK allowed 1:0, 0:1, 1:1, 2:1, 1:2, and 2:2 alignments, with swapping over a limited distance; Giza^{++} allowed 1:0, 0:1, 1:1, and 1-to-many alignments, with swapping; and the gold standard allowed any alignment, so long as it was not discontinuous.

Additionally, as mentioned, the gold standard was not created with nearly the same care as in other gold standard data sets (e.g., as in (Melamed 1998) or (Och & Ney 2000)). Since there was only one annotator, annotation was not always consistent; the data was very uniform (i.e., it would have been better to use random verses from the Bible than verses in sequence); annotation was done as fast as possible; and, one might argue, there may have been bias since I did the annotation and knew in advance what I intended to measure it against. One could argue further that Giza^{++} would be at more of a disadvantage, since it permits a type of alignment which the gold standard does not (i.e., discontinuous). Nevertheless, I still hypothesized that the Giza^{++} data set would show a higher agreement rate (higher WAA score) with the gold standard than would the CYK set, and this indeed was the case.

	CYK	Giza^{++}	Gold
CYK	1.00	.50	.53
Giza^{++}	.50	1.00	.60
Gold	.53	.60	1.00

Table 3.3: Word Alignment Agreement (WAA) scores for the CYK and Giza^{++} word alignment sets with the hand-aligned gold standard.

In Table 3.3, I report the WAA scores for the two automatically word-aligned data sets on 1529 verses against the gold standard, and, for completeness, against each other. As can be seen in the table, Giza^{++} is significantly closer to the gold standard, showing an agreement rate of .60 with the gold standard, while the CYK data set has a .50 agreement with the gold standard. Of course, one might remark that both WAA scores are rather low. These scores will help inform the discussion of the evaluation of the translation system in Chapters 5 through 7.

Finally, in Table 3.4, I show the wa-edit-agreement results for the same data sets, with all of the null alignments removed, which shows an even more pronounced difference in the quality of the Giza^{++} and CYK alignments, suggesting that null alignments are more problematic for all the systems (this seems typical of word alignment systems in general).[23]

	CYK	Giza^{++}	Gold
CYK	1.00	.64	.57
Giza^{++}	.64	1.00	.78
Gold	.57	.78	1.00

Table 3.4: WA-Edit-Agreement scores for the CYK and Giza^{++} word alignment sets with the hand-aligned gold standard, after removing all null alignments.

[23]The results of the wa-edit-agreement on the conservative word alignments are not shown, since, as mentioned, these scores yield the same results as WAA scores for conservative word alignments, and thus are the same as those in Table 3.3.

CHAPTER 4

THE LINKED AUTOMATA MODEL

> A Manhattan project could produce an atomic bomb, and the heroic efforts of the sixties could put a man on the moon, but even an all-out effort on the scale of these would probably not solve the translation problem. (Kay 1982:74)

In Chapter 2, I presented an overview of some of the most important models of empirical machine translation in order to highlight the motivation for the models as well as their design. Chapter 2 also briefly covered some of the benefits and shortcomings of the translation models, in terms of both their training and use. It described statistical machine translation (SMT) approaches in general, focusing on the IBM approach (Brown *et al.* 1993), and placed particular emphasis on finite-state probabilistic MT models, such as Alshawi *et al.* (2000), Vilar *et al.* (1999), Knight & Al-Onaizan (1998), and Bangalore & Riccardi (2001). Lastly, Chapter 2 gave an overview of example-based machine translation (EBMT).

Having laid the groundwork in terms of previous approaches, in this chapter I present a new finite-state probabilistic MT model and explain the motivation for the approach. In section 4.2, I detail the architecture of the model, describing both the automata and the alignment table. Section 4.3 then gives a detailed description of the various probabilities in the model, their interaction, and the motivation for them. Sections 4.4 and 4.5 illustrate the training and translation processes for the model. Finally, in section 4.6 I situate the new model by comparing it to some of the other data-driven MT approaches presented in Chapter 2.

4.1 The Model and Its Motivation

As discussed in Chapter 2, data-driven translation models, those that learn automatically from bitexts, are favored for their robustness, ability to generalize, and lack of reliance on sophisticated human linguistic input. Certainly they are not the only reasonable approach to machine translation, and indeed it may in the end be that hybrid approaches which leave open the possibility for human input will perform the best. Within the empirical domain, however, there are particular features that make certain MT models more appealing than others, in terms of performance, simplicity (both in terms of understandability and ease of implementation), and plausibility as appropriate linguistic models for translation. Other important issues for all MT systems, in addition to their coverage, are their adaptability to new domains and language pairs and their ability to generalize to unseen inputs. Data-driven MT methods are an exciting and active area of research precisely because of the paucity of human resources they require. The goal of creating an automatic translation system with little to no human input has now passed from the realms of science fiction to the frontier of computational linguistics research. The ability to use linguistic knowledge to guide learning algorithms (without the need for hand-coded input), as well as the availability of electronic texts and increased computing power, have made such translation ventures possible.

Pure SMT approaches, such as the IBM *Candide* system (Berger *et al.* 1994), have performed surprisingly well, relying mainly on lexical information and using word alignment as the focus of the overall translation model. In conjunction with a language model, this translation model is the core of the SMT approach (see section 2.2.1). Finite-state systems (section 2.2.2) can be used to replicate SMT systems (e.g., Knight & Al-Onaizan (1998)), but they are better exploited by using the order capturing nature of finite-state devices. In this manner, finite-state devices can be used to make better language models, since probabilistic distributions are formed only over the orderings that are possible. In terms of their complexity and applicability, finite-state models vary from the most simple: large transducers used in isolation; to slightly more complex: subsequential transducers (SSTs) used with special generalization and error correcting techniques; to quite complex: devices applied recursively to account for hierarchical structure, such as head transducers (HTs). What all of these models have in common is their use of transducers, in one form or another.

The questions which the proposed model attempts to answer are: Exactly how complex need a probabilistic finite-state MT model be, what is the most appropriate finite-state translation model, and is there a more direct finite-state encoding of the translation relation than can be found in transducers? The aim of the research is to see exactly how far rather 'simple' finite-state devices can be pushed in the translation task, and to demonstrate that these simple approaches may perform as well as more powerful designs. More importantly, while such simple technology may prove easier to understand, easier to implement, potentially more efficient, and more widely applicable, the design of the proposed model also allows for a straightforward picture of the translation and language models, and the translation process. It does so by not conflating recursive use of the MT model with the model itself. In some senses the model's chief benefit is this compartmentalization of the data from its use. This design allows for more complex algorithms and computational techniques to be applied to the data to take advantage of its well laid-out design.

In contrast to the other finite-state models, this model moves away from transducers to *linked automata*, pairs of automata where the transitions are linked to one another by a weighted function (called an *alignment table*), for determining the best links. The first intuition here is that since in translation we are dealing with two languages, there must be two language models: the two automata. The second intuition is that while transducers can be used to represent a relationship between two languages, they impose the ordering constraints of one language on the other. In this sense, transducers are an inappropriate translation model, because they misrepresent the crucial relationships between words in the two languages, even when they get the larger relationship picture correct for entire strings. While the relationship between source and target strings is often viewed as one of alignment between words (see for example Figure 4.1 below, repeated from the earlier Figures 2.2 and 3.1), transducers must be elaborately coaxed to represent these alignments (see, for example, the SST treatment of asynchrony, in section 2.2.2).

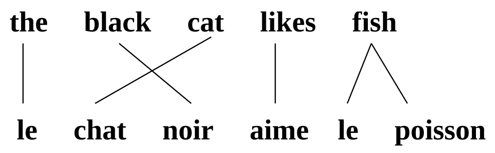


Figure 4.1: A word-aligned English and French bitext

Why not simply represent the alignments between the words of bitexts literally, while still representing the relative ordering of the source and target sentences? This is exactly what the linked automata approach does. It moves the word-aligned sentences directly into the model, with connections between source language automaton and target language automaton transitions representing the alignments between words. For example, the alignment of Figure 4.1 can be directly represented in the model as shown in Figure 4.2, where the source language automaton (English) is on top, the target language automaton (French) is on the bottom, and the alignments between transitions are shown graphically as dotted arrows. Thus, the linked automata model consists of two automata, one for each language, and a function (the alignment table) to connect them.

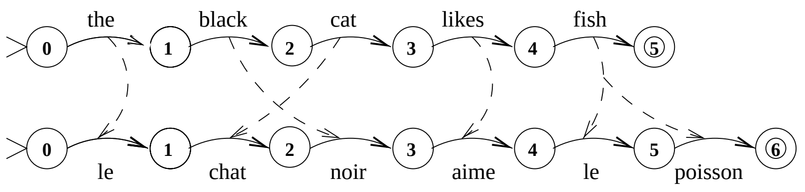


Figure 4.2: A bitext as represented in the linked automata MT model

This model design arguably better captures the relationship between the bitexts than do traditional transducers, and does so in an extremely economical way (i.e., there is no need for extra states in the model to handle ordering asynchrony). The relationship between the transitions is implemented as function from (sequences of) source automaton transitions to sets of (sequences of) target automaton transitions, known as the alignment table. The model is constructed directly from word-aligned bitexts, and translation is simply of matter of parsing the source sentence with the source automaton, and then using the linked (or *activated*, both terms will be used here interchangeably) transitions in the target to form a target language string. These transitions are forced to conform to the ordering constraints of the target language model.

This linked automata model is quite simple, and as such might be viewed as not fully capable of representing the translation process, since, for example, automata can only represent regular languages and not context-free languages, while natural languages are standardly assumed to be at least context-free. But, as mentioned in section 2.2.2, there exist methods for approximating natural language syntax in finite-state devices (such as described in Pereira & Wright (1997)). Additionally, it is important to bear in mind that the sorts of data typically used to illustrate the context-free nature of natural language (for example from Swiss German) are always bounded, i.e., they are always necessarily of a finite length. In fact, it is unusual to see these context-free effects (of center-embeddings, for instance) occur in units of greater than four (rarely would we see more than four embeddings). Thus, the memory constraints of the human processor may constrain natural language constructions such that the power offered by finite-state devices may be sufficient (Yngve 2000).

Still, it might seem from its modest finite-state set up, as well as a lack of cyclicity (see section 4.2.1), and the lack of an accounting for hierarchical syntactic structure, much less semantics in a traditional sense, that the linked automata would have no hope of coping with real word natural language tasks. In isolation, and in its purest form, this may be true. By using techniques for approximation and generalization, and heuristics for working with partial results, however, the model may indeed prove to be adequate for translation tasks, especially in limited domains. More importantly, I hope to show that the architecture of the model provides a suitable base from which more ambitious translation systems can be developed, and to begin the process of demonstrating how far this augmented finite-state architecture can be extended in the field of MT.

This is the reason why I sometimes refer to the use of the model as *Stone Soup Translation*. In the folktale, a man arrives in a village and claims that he can make soup out of nothing more than a stone and some water. One by one, however, he convinces the villagers to supply him with additional ingredients to make the soup. He adds carrots and potatoes and meat and so on. He tells the villagers that while the stone alone is sufficient, the addition of each next ingredient will make the soup taste just a little better. Much to the villagers' surprise, the finished soup turns out to be delicious—even though it was made out of 'only' a stone. So too functions the translation model. In its purest from, it may make not so savory a translation soup. But it serves as a sensible foundation from which more able translation systems can be constructed. We next begin to view the model in detail.

4.2 A Look Inside the Linked Automata Model

4.2.1 The Automata

The linked automata model contains two automata, the source automaton, a model of the source language, and the target automaton, a model of the target language. Both of the automata are standard. They consist of a finite set of states, Q, a start state q_0 (usually numbered 0 in the figures and shown with a wide arrowhead pointing to it, as in Figure 4.2), a set of final states, $F \subseteq Q$ (shown as double circles), and a set of transitions, T. Each of the states is uniquely numbered so that we have a means to identify them.

Transitions between states take the following form $< q_b, q_e, w, p >$, where q_b is the *begin state*, q_e is the *end state*, w is the label (w is for *word*), which may be empty (i.e., an epsilon-transition), and p is a probability associated with the transition. A string is recognized in the usual manner (i.e., we can recognize a string of words if there exists a path of transitions labeled with those words in the same order from the start-state to a final state). The probabilities on the transitions are available to select a single option (if desired) during recognition, in cases where there is more than one path by which the same sentence can be recognized, by choosing the most likely path.[1]

We require that the automata be acyclic.[2] Constructed purely from natural language sentences, acyclicity is guaranteed, since the sentences must always have a finite length, and cycles in an automaton allow for sentences of infinite length. So, an automaton created to fit all and only the sentences of a training set should be able to recognize all and only those sentences. But, if, for example, in an attempt to minimize space (decrease the number of states and transitions) or increase coverage,

[1]The automata in the model do not have to be deterministic, but only deterministic automata are used for the project described in the dissertation (at least prior to generalization). The probabilities are typically most useful for selecting between *fragments* of complete paths, when working with partial results. The transition probabilities (of the target automaton) also play a role after recognition. See sections 4.3 and 4.5.

[2]This is a rather strong constraint, and might be relaxed to allow some loops, such as those which go only from a state to itself, and through no others. At this stage of the project however, no cycles are permitted.

we were tempted to do something like have just one transition per word, we would create cycles. Continuing with this scenario, suppose that when constructing a system from the following training bitexts the construction algorithm allowed only one transition per word:

(55) ⟨*john loves mary/jean aime marie*⟩
⟨*mary loves john/marie aime jean*⟩

We can see the immediate problem that would develop in Figure 4.3. Cyclicity does not preserve the intended translation word ordering.

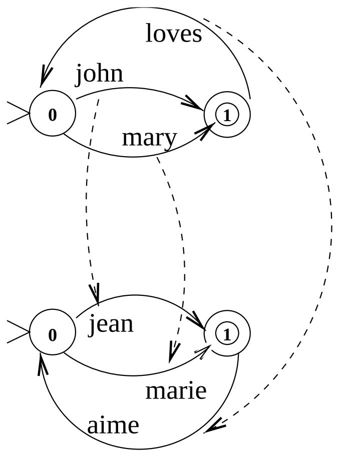


Figure 4.3: Linked automata model of one transition per word, illustrating the problem of cyclicity

For example, recognition of the sentence *john loves mary* in the source automaton would activate the transitions labeled *marie*, *jean*, and *aime* in the target. However, it would not be clear if the translation of the sentence should be *jean aime marie* or the incorrect *marie aime jean*, or, for that matter, any of an infinite number of other translations. Note that in and of itself, this increased coverage (e.g., the source automaton now recognizes *john loves john*; and longer sentences, such as *john loves mary loves john* could also be recognized and potentially translated) is something which should be desired, so long as the translations are accurate. This idea will be discussed more in Chapter 6.

I also plan to experiment with allowing cycles to a great degree (e.g., one might only preclude cycles from including the start state), knowing that this could make determining the order of target words more difficult. This may be sensible since fully-connected cycles are unlikely to arise; thus the proper ordering may still be determinable. The benefits of removing the acyclicity requirement will be increased generalization (see Chapter 6); much smaller system size (a side-effect of the main generalization strategy, merging); and the generalization process will be much faster, since expensive cycle-checks will be avoided. Thus the question of cyclicity is closely tied to the question of scalability for the model.

4.2.2 The (Simple) Transition Alignment Table

The alignment table may be implemented in a number of different ways. In this section, I describe what will be called a *simple* table. In Chapter 7, a new table architecture will be introduced which extends the coverage of the model.

The simple table is nothing more than a (partial) function from the power set of source transitions to a set of sets of pairs of target transitions and probabilities.[3] For example, using the simple linked automata translation system shown in Figure 4.2,

[3]We could equivalently view the table as a relation between the power set of source transitions and the power set of target transitions paired with probabilities.

the source transition $< 2, 3, cat >$[4] would be aligned in the table with the target transition $< 1, 2, chat >$. Suppose that this alignment comprised 17 percent of the overall alignments, and further that $< 2, 3, cat >$ was never aligned with anything else during the training process. Then we would expect to see the following table entry for this alignment:

(56) $(< 2, 3, cat >) \rightarrow \{((< 1, 2, chat >)\ .17)\}$

The probabilities in these table alignments sum to one (I discuss how I arrive at the probabilities and the rationale for doing so in section 4.3), and take the general form (where st_i is a source transition, tt_j a target transition, and $a_{x,y}$ an alignment probability):

(57) $< st_1, st_2, ..., st_n > \rightarrow \{(< tt_1, ..., tt_n >\ \ a_{1,n})...(< tt_1, ..., tt_m >\ \ a_{1,m})\}$

If we ignore the probabilities in the alignments, the translation model simply nondeterministically provides a set of translations for any input string which can be recognized. It is the alignments that need to be trained as the model is constructed (thus, the model requires word-aligned bitexts from which to train).[5]

4.3 Making the Model Probabilistic

There are three major considerations regarding probabilities in the model: (1) the probabilities of the language models (the two automata); (2) the probabilities in

[4]Where not necessary, probabilities in transitions are not shown.

[5]Note that as presented in the table description and figures, the alignments from source transitions to target transitions may give a misleading impression. The alignments are, in fact, bidirectional, in the sense that the table is easily inverted, allowing for translation in either direction. I represent them as unidirectional for simplicity.

the table (the alignment probabilities); and (3) the interaction between the two. In this section, I first cover each of these topics in detail (in sections 4.3.1, 4.3.2, and 4.3.3), and then turn to the handling of some of the subtler probabilistic issues for the linked automata model, the probabilities of fragments (section 4.3.4) and probabilities for what will be called *empty transitions*, those that arise from 1:0 alignments (section 4.3.5).

4.3.1 Automata Probabilities

In 4.2.1 transitions were depicted as having one slot for transition probabilities. This is a slight simplification. In the implementation of the prototype linked automata system, a transition minimally has the following form $< q_b, q_e, w, c, p >$,[6] where c, the new item, is a count of the number of times the given transition was used during training (i.e., how often the transition has been traversed during the construction of the automaton). Keeping both a slot for the raw counts as well as one for probability gives us some flexibility in terms of how the probabilities are calculated (this flexibility will be taken advantage of in section 4.3.4).

Now, if we focus only on the counts of transitions, each count (once normalized by the total of all the transitions' counts) gives an indication of how likely the word which labels the transition is in the training data, within the context of the words which precede and follow it. This probabilistic model is not, however, the most reasonable one to select for a language model. A better probabilistic model would focus on paths, i.e., sequences of words, rather than words themselves. This is

[6]Additional slots may also be used during translation to store the source-word-store and the distance, but they are not relevant to the probability discussion; see section 4.5.

because we want the language model to give an idea of how likely a sequence of words is, rather than how likely the words that make up the sequence are. The approach described above comes close to this ideal, but it does not take into account the likelihood of a sequence beginning or ending, and the normalization step is wrong, because it is normalized by the number of word tokens without fully taking into account the different paths in which they occur.

One common way to implement the more accurate notion is to have two types of probabilities. Following Vilar *et al.* (1999), first, for any given state, q, we need to know the probability that it is final: $P_{is\_final}(q)$. The second probability (the transition probability, i.e., the p in transitions) is like the (normalized) count we already have in transitions, but it is defined so that for every state q, the sum of the probabilities of the transitions departing from q and the probability that q is final is equal to one. So, for a transition departing from state q with count c, where the total count on all transitions departing from q is tc, the transition probability would be:

(58) $(1 - P_{is\_final}(q)) \times (c/tc)$

Using these two probabilities together, the probability of a string being recognized by the path of transitions (where q_i is a state number and $p_{i,j}$ is the transition probability):

(59) $< q_0, q_1, w, c_{0,1}, p_{0,1} >, < q_1, q_2, w, c_{1,2}, p_{1,2} >, ..., < q_{n-1}, q_n, w, c_{n-1,n}, p_{n-1,n} >$

is defined as:

(60) $P_{is\_final}(q_n) \prod_{i=1}^{n} p_{i-1,i}$

Both the state probabilities (i.e., the probability that a state is final) and the transition probabilities are easily estimated by using the frequencies of transitions (i.e., the counts) and of final states during training. The probabilities could be calculated on the fly, from frequency data, during translation, but in practice are typically calculated after training, so that the results can be stored and therefore not recomputed unless the automata are modified. Having both count and probability slots in the transitions (rather than, for example, replacing the counts with probabilities) allows for recalculation as information changes, as well as for different probabilistic models to be used without changing the data from which they were created (see section 4.3.4). Both of the automata probabilities are implemented in the logarithm domain, to avoid the problem of underflow.

4.3.2 Table Probabilities

We next turn to probabilities in the alignment table. Now, every (possibly empty) transition sequence in the source automaton is aligned with a set of (possibly empty) transition sequences in the target automaton. These alignments are based fully on the alignments of bitexts from a training corpus. An alignment between source transition sequences and target transition sequences thus represents a straightforward alignment between sequences of words in two sentences. One important simplifying assumption then is that the alignments of the sequences of words in sentences is strongly correlated with the alignment (i.e., best translation) of sentences themselves. The goal in the table is to best represent this connection with probabilities, since sequences of transitions can be aligned in more than way. To do this, we count. That is, during the automata and table construction process, every time a sequence in the source automaton is aligned with a sequence in the target, we increment the number stored in the table for this alignment by 1. For example, with table entries having the form where a source transition sequence (STS) aligned with any number of target-transition-sequence (TTS) and probability pairs (see earlier examples (56) and (57)), a *hit* between STS_i and TTS_k would result in changing the pair (where c is a count) $< TTS_k, c >$ to $< TTS_k, c+1 >$ (provided this pair is already in the set of alignments for STS_i). So after the new hit the table entry for STS_i might look as follows:

(61) $(STS_i) \rightarrow \{< \textit{somepair} >< \textit{somepair} >< TTS_k, c+1 >< \textit{somepair} > ...\}$

These hits then give us unnormalized probabilities for the alignments. One option for normalizing the probabilities could be to assume a uniform distribution for the alignments, where any alignment of words is equally likely. But this does not seem very intelligent, since we know some alignments are much more likely than others.[7] Note that with the simple table, this discussion assumes we are only interested in the alignment of substrings (i.e., contiguous sequences). This assumption will be removed when we move to all sequences, including discontinuous ones, in Chapter 7, so the number of possible alignments could be quite large. In any case, a more appropriate normalization strategy is to take the total

[7]For example, a 2 to 1 alignment is much more likely than a 17 to 1 alignment, and discontinuous alignments will typically only span limited distances and have a limited number of discontinuities.

number of hits (for the whole table, call this number N) and normalize by it, yielding a well-founded probability distribution.[8] In this manner we normalize by the number of alignments that actually occur.[9]

4.3.3 Combining Probabilities

The next aspect to consider is the interaction of alignment probabilities and transition probabilities. As will be described fully in section 4.5, translation consists in part of the transitions of the source automaton "activating" the transitions of the target automaton, via the links in the table. Looking more closely at this process, a source language sentence is recognized, yielding a single, most probable source transition sequence (STS). Once the STS has been chosen, the part of the source transition probabilities in the translation process is finished. The STS is then used to select sequences of target automaton transitions, by using all of the links from subsequences of the STS to target transition sequences which are found in the table (I will describe two methods for making this selection, first in section 4.5 and then in Chapter 7). The final stage of translation is to choose the best target transition sequence from the activated target transitions.

If all of the alignments in the table were equally likely, we could rely on the target automaton's transition probabilities alone to choose among competing sequences of

[8]In practice, I once again take the logarithms of the normalized probabilities to avoid underflow in later calculations.

[9]In truth, the normalization strategy is largely irrelevant, and could be removed, if the alignment probabilities were used in isolation, because (as in any normalization scenario) the counts could be used as unnormalized probabilities. Normalization, however, has the benefit of making the probabilities between 0 and 1 (prior to logarithm operations), which makes sensible weighting of the various probabilities as they interact much more straightforward.

transitions when looking for the most probable translation. Since, however, alignment probabilities vary, we rely on them to tell us how much to weight each target transition probability, i.e., for any given activated target transition, we multiply its probability by the probability of the alignment which generated it.[10] In this manner, each transition sequence in the target may be evaluated both in terms of its likelihood in the target language model and as having been generated via the alignment model. For example, a target transition sequence:
$< q_0, q_1, w_1, c_{0,1}, p_{0,1} >, < q_1, q_2, w_2, c_{1,2}, p_{1,2} >, \ldots, < q_{n-1}, q_n, w_n, c_{n-1,n}, p_{n-1,n} >$
with alignment probabilities $a_1, a_2, \ldots, a_n$, for each transition, respectively, will have the overall translation probability:

(62) $P_{is\_final}(q_n) \prod_{i=1}^{n} a_i p_{i-1,i}$

This ends the discussion of the three most important issues regarding probabilities in the linked automata model: transition probabilities, alignment probabilities, and the interaction between the two during translation. We next turn to some of the finer points regarding probabilities in the model.

4.3.4 Fragments

The automata probabilities as defined in section 4.3.1 can be viewed as a best-case scenario: We get the probability of a sequence of transitions from the automaton's

[10]Multiplication is not the only option here, but has the merit of simplicity. An additional option would be to add a parameter so that the effect of the alignment probability could be adjusted. Such a parameter could be tuned experimentally, although appropriate values would likely vary for different data sets.

start-state to a final state. For any string not recognized by the automaton, a zero probability is assigned.[11] In this idealized scenario, it makes perfect sense to use state probabilities along with transition probabilities.

When dealing with fragments (sequences of transitions in an automaton which do not begin at the start-state and end at a final state), however, using state probabilities is problematic. This is because the probabilities as defined in section 4.3.1 place importance on whether the last state used in a transition sequence is final, but for fragments, the "finality" of a last state is irrelevant. Also, the transition probabilities for fragments will be skewed, because they are calculated relative to the final-state status of the state from which they begin.

To illustrate this situation more clearly, let's look at a situation when fragments might come into play. When attempting to recognize a string of words in an automaton, we always try to recognize the string from the start-state to a final state. If a sequence of transitions is found matching this criteria, then recognition is successful, and clearly, state probabilities should be used, because they tell us how likely this sequence of words is, and not just how likely the words themselves are (see section 4.3.1). Suppose, however, that recognition fails, e.g., for some string of words, "A B C," there is not a path in the automaton beginning at the start-state that is labeled with A then B and then C, where the transition labeled with C ends at a final state. The automaton will do exactly what it was designed to do, and assign a zero probability to ABC.

[11]This is a separate issue from whether it assigns a nonzero probability to strings on which it was not trained; post-generalization, it leaves some probability mass for any string which it can recognize, including many on which it was not trained, see Chapter 6.

Part of the motivation for using the automata as language models, however, is that they can also be used for extracting partial results (as will be described in Chapter 6). That is to say, just because we cannot recognize ABC, we are not prepared to give up, since it may be the case that the string exists in the automaton, just not between the start-state and a final state, and could therefore yield a perfectly reasonable translation. Moving to a slightly worse case, it might be that the sequence ABC cannot be recognized in the automaton, but that AB or BC could be recognized, as well as C and A in isolation, so that these parts could be used in a translation. In any of these cases, we are not concerned with the probability that the last state in a sequence is final, since we are looking at subsequences of valid sequences.

Now, suppose then that we are indeed trying to recognize ABC, as a fragment, and we find AB and then look for C. Next, suppose that we find two different transitions labeled with C, which we will call C_1 and C_2, which we must choose between. Let us suppose further that C_1 has a count of 5000, and C_2 has a count of 3. So, ideally, without the aid of any contextual information, we would want to select C_1, because, being so frequent, it is likely to give us more reliable translation information. Now, assume that the state which begins C_2 is never final, and that C_2 is the only transition which leaves this state, and assume also that the state beginning C_1 is sometimes final, with a probability of .1, and that there are a number of other transitions beginning at the same state with high counts, totaling 20,000. Then, using the probabilities defined in section 4.3.1, we will always select C_2, because it will have a transition probability of 1 (i.e., $(1-0) \times (3/3)$), while the transition probability for C_1 will be much lower, at .18 (i.e., $(1-.1) \times (5000/25000)$).

Thus C_2 is being selected over C_1 not because it is the most likely transition with the label C, but because the state that begins the transition happens to never be final and not be part of a very frequently crossed path. C_2 is exactly the sort of transition we do not want to choose when working with fragments.[12]

There is an obvious solution to this problem. The question we are asking is what is the most probable subsequence of transitions that has a given label. The answer is to rely on counts exclusively, but again, only in the case of working with fragments. Thus, given the same C_1 and C_2, and a total count for the entire automaton of N, then the probability of each transition is the count divided by the total, i.e., $|C_1|/N$ and $|C_2|/N$, respectively, which would be $5000/N$ and $3/N$ in the example, meaning that we would make the correct choice. We can make this probabilistic choice because we are optimistic that the start-state and final state information is no longer relevant when looking at fragments; we believe this a reasonable thing to assume.

The flexibility mentioned earlier in section 4.3.1 makes this technique easy to implement, because the count information is still available in the transitions. Thus these "pure transition probabilities" (i.e., those where no state information is used) can be employed when needed for fragments, and the more traditional probabilities of section 4.3.1 can be used for normal recognition.

For larger sequences, the process is identical, and we just multiply the probabilities. Thus the probability of a sequence AB comprised of the connected transitions

[12]In the discussion of this hypothetical example, I ignore the question of the probability that the end states of the given transitions are final; this would come into play using the methods of section 4.3.1 and looking at the probability of this very short path (as defined in (60)), but the point made in the example would be the same if we just assumed that both transitions' end states had the same (nonzero) final-state probability.

A_1 and B_1 is $(|A_1|/N) \times (|B_1|/N)$.[13] This brings us to two of the most important points of the approach: We only use this strategy when traditional recognition has failed, and we always work from larger sequences to smaller ones. The reason for this latter point is that if we viewed small and large sequences as equals, a larger sequence could never beat a smaller sequence made up of the most probable smaller parts. This is because, using an example of AB, we might find a very probable connected sequence labeled AB, but there might be individual transitions labeled with A and B, respectively, which did not connect, but nevertheless had higher probabilities. Thus an algorithm which did not work from larger to smaller sequences would choose these unconnected A and B labeled transitions over what we would assume to be a better choice, the connected AB labeled transition sequence.

In summary, the fragment probability approach allows us to find the most frequently occurring transition sequences with the given labels, so that we can try to make the best choices (for finding alignments and other tasks) when traditional recognition from a start-state to a final state has failed.

4.3.5 Probabilities for Empty Transitions

The last point to cover concerning probabilities is how to deal with what I call *empty transitions*, those which arise from 1:0 word alignments. Recall that during translation, there is an interaction between alignment probabilities and transition probabilities: The alignment probabilities are used to weight the transition probabilities of activated target transitions (as explained in section 4.3.3). However,

[13] Again, as elsewhere, the normalization step could be omitted for sequences of the same length, but proves useful in terms of the interaction of probabilities.

when 1:0 word alignments occur during training, this means that a source word was aligned with null, i.e., no target word is associated with the source word, so while a source transition is created in the source automaton for the source word, no normal transition is created in the target automaton, since it would not be clear where the transition should be located (training is detailed in section 4.4).[14]

During translation, however, if there is a 1:0 alignment selected from the alignment table, a temporary empty target transition will be created, so that the source word's use in generating a translation can be tracked (i.e., we want to make sure each source word is used exactly once and only once in a translation; see section 4.5). Thus an empty transition is an odd transition, in that it has no state information and no label, only an indication of the source word which generated it and a probability. But what of this probability, what should it be?

If we wanted the empty transition probabilities to have no effect on translation, we might think to give them a zero probability, but that would have the unintended effect of zeroing out the alignment probability as well, when the two interact (i.e., when they are multiplied). A way to define the empty transition probability to have no effect would be to set it to one, rather than zero. That way, only the alignment probabilities would have an effect when the two interacted during translation. The problem with this approach, however, is that it would tend to bias translation towards the use of these null alignments, which is undesirable without an additional linguistic motivation.

What we want to find then is a realistic probability for empty transitions. One solution for assigning a probability to transitions with an unknown probability is to

[14]0:1 alignments pose no problem in this respect, since there is a target transition created.

look at other transitions. For example, we could take the average of the transition probabilities of the non-empty transitions in the target automaton, and use this as a transition probability for empty transitions. While this is not an unreasonable thing to do, it is not as principled as we would like, because it does not take into account the likelihood of empty transitions. The solution chosen for this project is to count how many times an empty transition is used in the target versus how many times it could have been used. This gives us a good estimate of how likely such transitions are. The calculation is shown in (63), where E is the sum of the number of times a 1:0 alignment is used in the alignment table, and C is the number of word tokens (i.e., the total of all the transition counts) in the target automaton.

(63) $$\text{empty-transition-probability} = \frac{E}{E+C}$$

Thus we have the same transition probability for all empty transitions which is based on how prevalent they are. One final point to consider is that these empty transition probabilities make the most sense when working with fragments, because they are based on the likelihood of empty transitions relative to all other transitions, i.e., their calculation is much the same as that used for the pure transition probabilities in fragments, in section 4.3.4. They do not take state information into account (i.e., final state probabilities) just as is done in non-fragment situations (see section 4.3.1).[15]

[15]Thus, perhaps a better solution would be to also calculate a default (average) final state probability, which could be used to weight the empty transition probability in the case of non-fragment target automaton processing situations (since otherwise the lack of state information could have a slight bias towards empty transitions, although the different normalization strategy in non-fragment cases would likely have the opposite effect, since the denominator in (63) is quite large compared to the total count on transitions leaving a given state); or again in these non-fragment situations, we could return to the idea of the average transition probabilities. In any case, for the current project, only the formula in (63) was used.

4.4 Constructing the Model: The Algorithms for Training

Before beginning the discussion of training (this section) and translation (the following section), I should briefly discuss what level of detail I will go into regarding the implementation of the linked automata model. In the next two sections, as well as in the remainder of this dissertation, I give the top-level algorithms for creating and using the translation system, but typically not the specific implementation details, since the focus of the dissertation is on the model, its motivation, properties, use, and evaluation, and not on the engineering aspects, of, for example, how to efficiently implement automata. I have attempted, in general, to provide algorithms and an overview of data structures which are efficient for the model, and which identify and bypass what otherwise could be the major bottlenecks of the main programming tasks: training, translation, and generalization.

That said, I will in certain sections give some implementation details where otherwise an explanation of certain efficient techniques would be unclear. For example, I will give some details on how automata can be implemented so as to very efficiently work with partial results, i.e., how fragments can be quickly recognized, in section 6.3.1.1; detail some of the means for working with empty transitions during translation (which without special treatment could cause a combinatoric explosion) in section 4.5.3.1; give some of the important steps to make state merging relatively efficient (section 6.2); and discuss a specific implementation of the alignment table so that discontinuous alignments can be efficiently processed in Chapter 7.

Returning to the topic at hand, I next discuss how the linked automata system is built. Training (also called *construction*) of the model is a very simple process.

As stated in Chapter 3, the linked automata model is created from a file of word-aligned bitexts. Each word-aligned bitext consists of a source language sentence, a target language sentence, and a numerical representation of the word alignments between them.

```
build-translation-system ($input\_bitexts\_file$)
{ let $< source\_fsa,\ target\_fsa,\ table >$ = initialize-translation-system()
  for each bitext triple $< source\_string,\ target\_string,\ aligned\_pairs >$
                              of $input\_bitexts\_file$
  { $source\_wordnum\_to\_transition\_hash$ = add-to-fsa($source\_fsa,\ source\_string$)
    $target\_wordnum\_to\_transition\_hash$ = add-to-fsa($target\_fsa,\ target\_string$)
    for each aligned pair $< source\_wordnums, target\_wordnums >$
    { add-to-alignment-table($table$,
        (get-trans-seq($source\_wordnums,\ source\_wordnum\_to\_transition\_hash$))
        (get-trans-seq($target\_wordnums,\ target\_wordnum\_to\_transition\_hash$)))
    }}}
```

Figure 4.4: Linked automata model construction algorithm from aligned bitexts

The basic algorithm for translation system construction, absent refinements such as merging (see Chapter 6), is as follows: Given a source automaton, a target automaton, and a table, simply read each bitext triple, add the appropriate states and transitions to the source automaton so that the source sentence can be recognized, perform the analogous additions to the target automaton with respect to the target sentence, and for each alignment pair of the form:

(64) $< (source\_tran_i ... source\_tran_j),(target\_tran_k ... target\_tran_l) >$

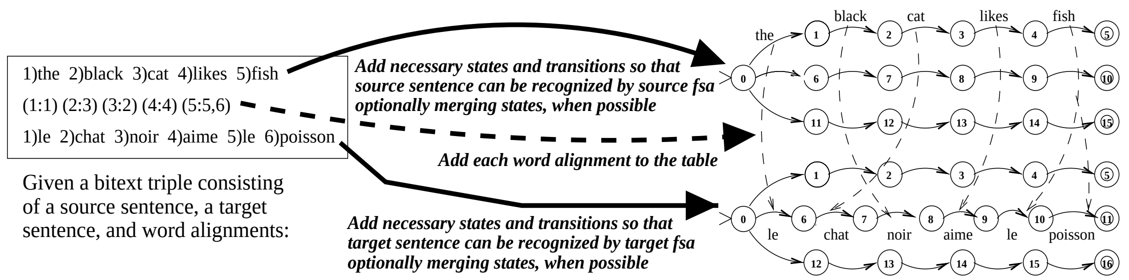
add an entry to the table with a count of 1, if one does not already exist. If such a table entry already exists, increment its count by 1. The top-level system construction algorithms are given more formally in Figures 4.4 and 4.5, and a graphical representation of the process for the aligned bitext previously shown in Figure 4.1 is shown in Figure 4.6.

```
add-to-fsa ($fsa$, $sentence$)
{ $m$ = number of words in sentence
  $prev\_state$ = get-start-state($fsa$)
  for($i$=1; $i > m$; $i^{++}$) {
       ;;; if transition already exists in right place, just adjust
       ;;; the value for next previous state (?Q2 is a variable
       ;;; over states, and w$_i$ is the i$^{th}$ word of sentence)
       if exist $< prev\_state, ?Q2, w_i >$
       {   $prev\_state$ = $?Q2$
           push ($i$, $< prev\_state, ?Q2, w_i >$) onto $word\_num\_trans\_hash$ }

       ;;; otherwise, make new transition and connect prev state to it
       else{ $< prev\_state, Qnew, w_i >$ = make-new-transition($fsa$)
                $prev\_state$ = $Qnew$
            push ($i$, $< prev\_state, Qnew, w_i >$) onto $word\_num\_trans\_hash$ }
  }
  ;;; make adjustments at sentence end
  pushnew $prev\_state$ onto final-states($fsa$)
  return $word\_num\_trans\_hash$    }
```

Figure 4.5: Automaton construction function for building from aligned bitexts

As can be seen in Figure 4.5, the automata construction is traditional, in that we do no minimization on the fly (save for not constructing new transition sequences when the path from the start-state to the present state already exists). The version



1)the 2)black 3)cat 4)likes 5)fish
(1:1) (2:3) (3:2) (4:4) (5:5,6)
1)le 2)chat 3)noir 4)aime 5)le 6)poisson
Given a bitext triple consisting of a source sentence, a target sentence, and word alignments:
Add necessary states and transitions so that source sentence can be recognized by source fsa optionally merging states, when possible
Add each word alignment to the table
Add necessary states and transitions so that target sentence can be recognized by target fsa optionally merging states, when possible
the
black
cat
likes
fish
le
chat
noir
aime
le
poisson

Figure 4.6: The construction process

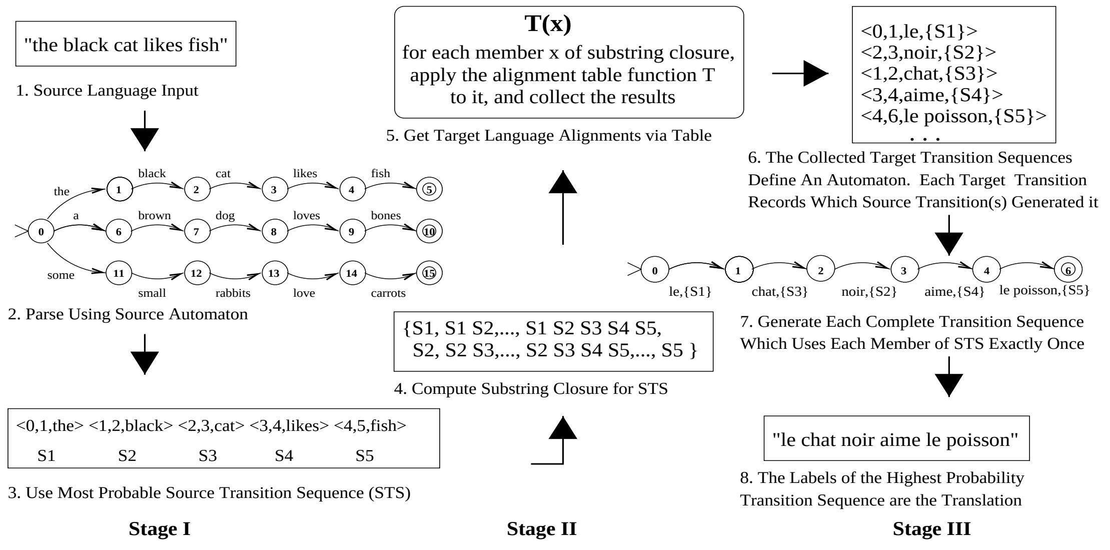
of the process shown in Figure 4.5 is somewhat simplified. In addition to adding the states and transitions necessary for each new sentence, the counts used for probabilities are also incremented on both transitions and final states. Transitions also store a measure for the distance they are from the start-state, which is most easily calculated during construction, since all paths constructed begin at the start-state.

As is shown in Figure 4.4, translation system construction requires one pass through the aligned bitext corpus. This task, like sentence and word alignment, is a one-time, off-line task. The probability calculations for the translation system are typically carried out at the end of the construction process (see section 4.3). In its most basic form (i.e., when no generalization is attempted during construction), construction is quite fast: Using a 500MHz Sun Blade-100, a Lisp implementation of the model took less than one minute to construct a translation system from 1529 bitexts consisting of verses from English and Spanish versions of the Bible.[16]

[16]Constructing while generalizing at the same time takes significantly longer, but is a more sensible ordering than doing all the construction, then attempting to generalize. See Chapter 6 for a detailed description.

4.5 Using the Model: The Algorithms for Translation

The translation process for the linked automata model (also known as *decoding*, in some of the statistical MT literature) consists of three major stages, each which will be dealt with in turn: (I) processing the source sentence (section 4.5.1); (II) retrieving the target transition sequences to be activated from the table (section 4.5.2); and (III) using these target transitions to find all allowable paths through the target automaton (i.e., all the translations) (section 4.5.3).



"the black cat likes fish"
1. Source Language Input
the
black
cat
likes
fish
a
brown
dog
loves
bones
some
small
rabbits
love
carrots
2. Parse Using Source Automaton
<0,1,the> <1,2,black> <2,3,cat> <3,4,likes> <4,5,fish>
S1 S2 S3 S4 S5
3. Use Most Probable Source Transition Sequence (STS)
Stage I
{S1, S1 S2,..., S1 S2 S3 S4 S5, S2, S2 S3,..., S2 S3 S4 S5,..., S5 }
4. Compute Substring Closure for STS
T(x)
for each member x of substring closure, apply the alignment table function T to it, and collect the results
5. Get Target Language Alignments via Table
Stage II
<0,1,le,{S1}>
<2,3,noir,{S2}>
<1,2,chat,{S3}>
<3,4,aime,{S4}>
<4,6,le poisson,{S5}>
. . .
6. The Collected Target Transition Sequences Define An Automaton. Each Target Transition Records Which Source Transition(s) Generated it
le,{S1}
chat,{S3}
noir,{S2}
aime,{S4}
le poisson,{S5}
7. Generate Each Complete Transition Sequence Which Uses Each Member of STS Exactly Once
"le chat noir aime le poisson"
8. The Labels of the Highest Probability Transition Sequence are the Translation
Stage III

Figure 4.7: The linked automata translation algorithm (simple table)

Continuing with the *the black cat likes fish* translation example (shown earlier as a portion of a linked automata translation model in figure 4.2), I give a graphical overview of the translation algorithm in Figure 4.7 (repeating the relevant portions in each subsection), and describe the process in much finer detail in the sections for each stage which follow. In the description, for the most part we will not discuss probabilities, except for those aspects relevant to translation which were not already presented in section 4.3 (where a detailed description of the interaction of probabilities during translation is given).

4.5.1 Translation Stage I: Parsing The Source Sentence

Translation begins with a string of source words, S, that (ideally) represents a sentence in the source language (i.e., S is a sequence of words, $sw_1, sw_2, ..., sw_n$ separated by spaces; this is step 1 in Figure 4.7 and Figure 4.8). The sequence of words is first checked to see if all of the words have been previously seen during training. If not, there is no point in continuing to attempt to translate the sentence, since it will not be able to be parsed, i.e., it will not be able to be recognized by the source language automaton. A heuristic for dealing with unknown words is given in section 6.3.2, which enables the system to process the parts of a sentence that are known, and leaves unknown words untranslated.

Next, if all the words are known, we attempt to recognize S in the source automaton (i.e., find a path from the start-state to a final state where the transitions are labeled with the words of S in the proper order; this is step 2 in Figure 4.8). If there is no successful recognition, we fail (steps for handling this situation are given in section 6.3.3, where the sentence is broken up into smaller parts, to see if these parts can be recognized and then used for translation). If recognition is successful, however, we choose the most probable recognition, i.e., the most probable source transition sequence (STS); this is step 3 in Figure 4.8.

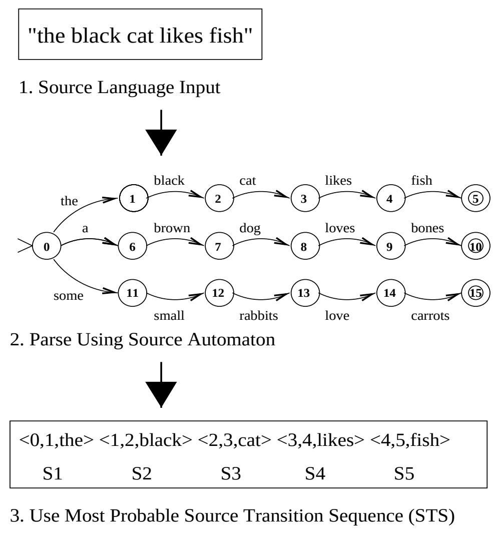

"the black cat likes fish"
1. Source Language Input
the
a
some
black
cat
likes
fish
brown
dog
loves
bones
small
rabbits
love
carrots
2. Parse Using Source Automaton
<0,1,the> <1,2,black> <2,3,cat> <3,4,likes> <4,5,fish>
S1 S2 S3 S4 S5
3. Use Most Probable Source Transition Sequence (STS)

Figure 4.8: Translation Stage I: Parse the source sentence

The choice of the best STS at this point is completely dependent on the source language automaton, i.e., it has nothing to do with the table alignments. This is an essential component of the architecture: We want language models (in this case the source language model) to handle the language-specific parts of the process, (e.g., recognizing source language sentences), and we want alignment models (i.e., the table) to handle the translation-specific parts of the process (i.e., associating words in the source language with words in the target language). One might ask at this point, why, if a source automaton could potentially yield more than one STS (i.e., more than one successful recognition of the sentence), do we use only the best one? Choosing a single best recognition is opted for because we want to decrease the search space as much as possible while retaining a good chance of getting the proper translation. Using all source recognition sequences would necessarily activate at least as many and probably more target transition sequences, and therefore yield more potential translations. Thus, our heuristic might cause us to miss certain possible translations. Nevertheless, we are willing to make this trade-off since the reduction in search space is potentially dramatic (this sort of n-best or beam search approach is used throughout the translation process).

4.5.2 Translation Stage II: Activating Linked Target Transition Sequences

The second stage of the translation process is to use the source transition sequence (again, the STS) from the first stage to select target transition sequences from the table. The discussion in this section assumes the use of a simple table, such as that described in section 4.2.2. The simple table was designed with the use of continuous alignments in mind (see section 3.2.1 for a discussion of different types of word alignments), and thus lends itself to specific processing techniques for such alignments. Although the simple table can be coerced to handle discontinuous alignments, in Chapter 7 we extend the table architecture to work with discontinuous alignments very efficiently, and thus the algorithms for this step in the translation process are adapted to accommodate the changes. We present a simplified graphical representation of the simple table in Figure 4.9, to remind the reader of its basic setup. Alignments are shown in the figure from sequences of source transitions (e.g. $< s1, s2 >$) to what are abbreviated as *TargVals*, which are the sets of target transition sequences and probability pairs which were described in section 4.2.2.

To use the STS is not simply a matter of getting the value for each source transition from the table, since alignments are between sequences of transitions. In principle, any continuous alignment type is allowed, e.g., 2 transitions to 1, 5 to

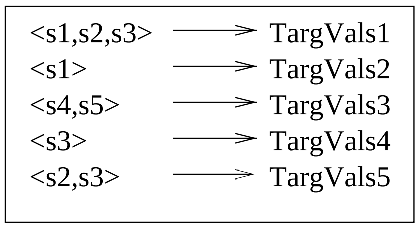


Figure 4.9: The basic idea of the simple table

6, etc. So, we first compute the *substring closure* for the STS (this is step 4 in Figure 4.10), which is simply the set of all substrings of a sequence, including the empty string. For a transition sequence of n transitions (typically from a sentence of n words), the substring closure has $n(n+1)/2$ substrings, plus the empty string.

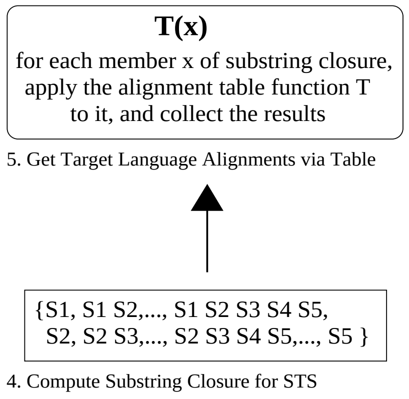

T(x)
for each member x of substring closure, apply the alignment table function T to it, and collect the results
5. Get Target Language Alignments via Table
{S1, S1 S2,..., S1 S2 S3 S4 S5, S2, S2 S3,..., S2 S3 S4 S5,..., S5 }
4. Compute Substring Closure for STS

Figure 4.10: Translation Stage II: Get links from table

Then, for each member of the substring closure, we get the value from the table (this is step 5 in Figure 4.10). If we think of the table as a function, T, we simply apply T to each member of the substring closure, and collect the results. The time needed to retrieve target transition sequences is quadratic with respect to the length of the source sentence. Given sentences of a typical length (< 50 words in our translation domain), which can be broken up into parts if necessary, and the fast operation of the target transition retrieving table function, the time needed comprises a negligible portion of the overall translation time. The collected results from applying the table function to the substring closure contain all the target transitions which will be activated, and lead us to the final translation stage.[17]

There are several points to note here. First, because T is applied to source transition sequences which may be aligned with more than one target transition sequence, and because it is possible that there is overlap in the source sequences used (e.g., see Figure 4.9, $s1$ participates in two different alignments), there may be more than one target transition that is activated by the same source word. This might appear problematic, in that it would suggest a given source word would seem to be occurring more than it actually is (i.e., one might expect to see two translations of the same word in the result). No problems are posed by this step, however, because in the final translation stage, we ensure that each source word is used exactly once in the resulting translation (see the discussion of the source-word-store (SWS) in section 4.5.3).

A second and related point is that here we do not use any heuristics to reduce the number of transitions activated, such as keeping only the most highly probable target transition sequence and probability pair, $TargVal_k$, for a given source

[17]The values of the function in Figure 4.7 (step 6) match those produced by following the dotted alignment arrows shown in Figure 4.2.

transition sequence, STS_i. The rationale is that it is quite possible that individual alignment probabilities in the best overall alignment scenario will not all be maximal in the sets in which they occur in the table, so we would like to keep all of the alignment possibilities open. If the number of activated target transitions does grow too large, however, we could limit their number to use the n-best alignments as a heuristic (a beam search), for some empirically determined n, but we would no longer be guaranteed to find the most probable translation.

Another point to note concerns 0:1 alignments (those which are known as *insertions* in the sequence alignment literature), where during training a target word was aligned with nothing in the source. These alignments are implemented in the table with a single *null* source transition. This means there is just one entry in the table for all of the 0:1 alignments, with potentially a large set of target transition sequences and probability pairs associated with it. It is to capture these 0:1 alignments that we use the empty string in the substring closure, but its use means we will produce many target transitions which will have little relevance to the potential translation (i.e., many will come from target transition sequences which are unrelated to the sequences which end up being used in the translation, because they originally arose from completely unrelated training sentences). The presence of this (potentially) large number of alignments is typically not problematic, however, precisely because they are unrelated (i.e., literally not connected to other relevant transitions).[18] In the last translation stage (section 4.5.3) transitions which do not connect with other transitions are, in essence, ignored, except in a worst-case scenario of word-for-word translation.

Nevertheless, one could apply again a beam search solution to these 0:1 alignments if necessary. A more principled solution, however, would be to actually put these null source transitions into the source automaton. If they existed in source transition sequences, then only those relevant to the actual translation would be activated in the first place. This solution raises the same questions as those identified in section 4.3.5 for so-called empty transitions (those that arise from 1:0) alignments, namely, where should the transitions go? One solution might be to, when necessary, add to the end of each sequence of transitions a single epsilon transition which then leads to a final state, which in the table would be aligned with only the products of 0:1 alignments for that specific translation pair. The negatives of this solution are that there would be many more entries in the table, and many more transitions in the source automaton (see section 6.4 for more discussion on this topic).

4.5.2.1 A Digression on Alignment Probabilities

At this point, we should go into a little more detail about the interaction of alignment and (target) transition probabilities. Recall that alignment probabilities are multiplied with target transition probabilities when target transitions are activated (section 4.3.3). This is very straightforward for 1:1 alignments. But what about for other types of alignments, for example 2:2 alignments?

[18] A large number of irrelevant 0:1 transitions can, however, be problematic when processing fragments; see section 6.4.

We can illustrate what happens with a simple example. Suppose that in the source automaton we have two transitions, a and b, and that in the target automaton we have two transitions, c and d, which have transition probabilities of .3 and .4, respectively. Next, suppose that in the table a is aligned with c, with an alignment probability of .2, b is aligned with d, with an alignment probability of .2, and the sequence a, b is aligned with c, d, with an alignment probability of .2, as shown in (65) below:

(65)
$$\begin{array}{lcl} < a > & \rightarrow & \{<< c >, .2 >\} \\ < b > & \rightarrow & \{<< d >, .2 >\} \\ < a, b > & \rightarrow & \{<< c, d >, .2 >\} \end{array}$$

Now, suppose that a source sentence recognition uses a and b; then all of the alignments in (65) will be used. When possible, in the current implementation we combine transitions during activation when there is more than one (e.g., as in the $\{<< c, d >, .2 >\}$ case, and in step 6 of Figure 4.7 where *le* and *poisson* are combined, as specified in their word alignment). This has two benefits: It reduces the number of transitions which result for the next stage, and it solidifies the notion that the alignment in the alignment model was between a sequence of words, and not just the individual words (i.e., the information conveyed by two 1:1 alignments is different from the information conveyed by one 2:2 alignment of the same parts; see section 3.3).

If we follow the practice defined in section 4.3.3, this process will yield the following three transitions (where transition a has the label a, and so on, and only the label and combined transition and alignment probabilities are shown):

(66) $< c, .2 \times .3 = .06 >$

$< d, .2 \times .4 = .08 >$

$< cd, .2 \times (.3 \times .4) = .024 >$

We get the third transition, $< cd, .024 >$, with a probability of .024 because the probability of a sequence of two transitions is the product of their probabilities, which we then multiply with the alignment probability for the sequence (rather than first multiplying each transition by the alignment probability). It is probably clear now where this example is heading: If we go on to put the transitions together, as is done in the last translation stage, we arrive at two different probabilities for a transition labeled *cd*, since we can put the first two transitions in (66) together to get: $< cd, .0048 >$.

Now the point here is not how we put these two *cd* labeled transitions together, or if we should (the answer is that we should, since they represent different ways of arriving at the same result, thus the probabilities would be added; we discuss this in section 4.5.3), but rather, is it a good thing that one combination of *cd* has a different probability from the other? This is a theoretical question.

By using the alignment probabilities in this manner, we bias the translation system toward using larger alignments (i.e., the more target transitions in an alignment, the more it is favored) since the whole sequence is multiplied once by the alignment probability rather than each transition at a time, before they are combined. This is the current practice in the model. It is done because we want to favor alignments which have more translation information in them, and a 2:2 alignment tells us more translation information than do the two 1:1 alignments which comprise it (note that this is the inverse of the case for evaluating word alignments, in section 3.3, where 1:1 alignments are more specific, and therefore more informative; here, in the translation task, a 2:2 alignment not only gives the transitions but enforces how they are combined, thus it yields more translation information).

Of course, there are many simple ways to make it so that the resulting probabilities would be equal, should that result be desired. This can be handled by normalizing differently, at the table level. Recall that we normalize by the total count of all alignments; this could be changed to factor in the number of transitions in each alignment, or more elegantly, to use the number of words in the alignment, following a sort of weighting scheme similar to that discussed with regard to word alignment evaluation in section 3.3, where each word token contributes the same amount of weight. A different approach might be to make probability adjustments during the second translation stage, by using alignment probability to the nth power, for n target transitions used, i.e., rather than $< cd, .2 \times (.3 \times .4) = .024 >$ we would get $< cd, (.2^2 \times (.3 \times .4)) = .0048 >$.

4.5.3 Translation Stage III: Assembling Activated Target Transitions to Find the Best Translation

The final stage of translation involves taking the target automaton transition sequences that were activated in translation stage II and putting them together in all ways that the target language model allows. I will sometimes refer to this process as *target parsing*. As with the discussion of the earlier translation stages, I first describe the general process, leaving the discussion of special cases for later. Thus, in this section I assume that there is always a target parse that can be found. In cases where there is not, I describe a method called *partial target parsing* in section 6.3.4, which is used to extract partial results. Empty transitions (those which arise from 1:0 alignments) also require some special handling, and are dealt with in section 4.5.3.1.

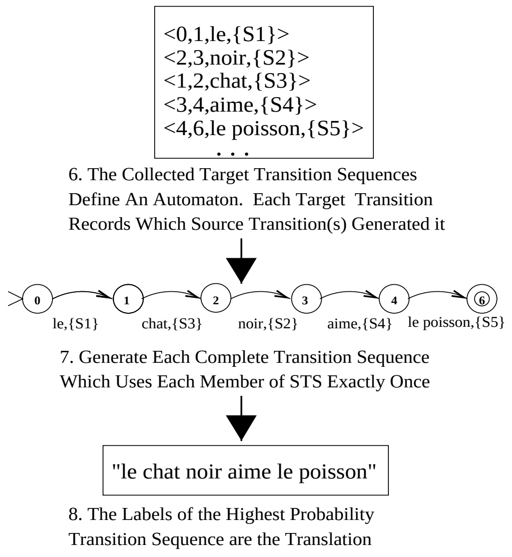
As mentioned previously in section 4.3.1, several different transition slots are used, depending on the needs of the system. Thus, for transitions to be used in target parsing, there is a new transition slot called the source-word-store (SWS). The use of the SWS, inspired by Johnston (1998), is analogous to the strategy used for word alignment in section 3.2.2: The SWS stores a numerical representation of the source language transitions from the STS which were responsible for generating the given target transition. For example, in step 6 of Figures 4.7 and 4.11, the target transitions contain as their last member a set, showing the source transition with which they were linked (e.g., $< 1, 2, chat, \{S3\} >$ indicates that this target transition was linked with the third transition, $S3$, of the STS used). The purpose of the SWS is to make sure that each source word is used exactly once in a translation.

The SWS can be implemented as a set of natural numbers; for example, a transition that had been aligned with the first two source transitions would have the SWS: $\{1, 2\}$ and one which had been aligned with just the fourth source transition of the STS would have the SWS: $\{4\}$. As will be seen shortly, part of the target parsing process can be viewed as putting target transitions together to form longer transitions. When this happens, the SWSs get combined. Thus the SWS of $\{1, 2\}$ and $\{4\}$ can be combined (i.e., unioned) to form $\{4, 1, 2\}$. We use unique numbers for the SWS instead of words so that in cases where a source word is repeated in a source sentence, there is no confusion. Because we want to ensure that no source word is used twice, we can only combine transitions when the intersection of their SWSs is empty.

Because we perform many set operations, such as union and intersection, on SWSs, it makes sense to implement them in a manner in which such operations are fast. One such technique is to use bit vectors, where the vector is as long as the number of transitions in the STS (i.e., as long as the number of source words), where a bit is marked 1 if the source word is present. For example, using natural numbers, an SWS containing words 2 and 5 from a five word source sentence would like $\{2, 5\}$, while the bit vector would be [10010]. Another implementation technique is to go a step further, and represent sets as integers, since a bit vector can be viewed as the binary representation of an integer (and some programming languages, such as Lisp, allow for fast logical operations on integers used to represent sets). Thus [10010] could alternatively be represented as 18.[19]

With this understanding of the SWS, we can now trace the third and final translation stage (shown again in Figure 4.11). We begin with a set of activated target transitions, like those shown in step 6 of Figure 4.11. We want to put these transitions together in all ways that the target model will allow, subject to the aforementioned constraint of using each source word only once. This target parsing process essentially means finding a connected path in the target automaton from the start-state to a final state, that both uses only activated transitions and has a *full* SWS (i.e., one that uses all the source words—all the bits in a bit vector representation are 1s).

[19]In the implementation, I used this integer representation technique, since it makes the programming tasks rather simple and elegant (e.g., one need not be concerned with what the size of the bit vector should be, and certain logical operations are predefined for integers which are not for bit vectors), but there are limitations, because one can only represent a source sentence of limited size, specifically, the number of words can be no greater than n, where 2^n must be <= the largest integer that the programming language can represent. Practically, this means that in the Lisp system one might have to break up sentences longer than 28 words into smaller parts; so the much better and more general solution is to stick with bit vectors, since they will only be limited by the memory available.



<0,1,le,{S1}>
<2,3,noir,{S2}>
<1,2,chat,{S3}>
<3,4,aime,{S4}>
<4,6,le poisson,{S5}>
. . .
6. The Collected Target Transition Sequences Define An Automaton. Each Target Transition Records Which Source Transition(s) Generated it
0
1
2
3
4
6
le,{S1}
chat,{S3}
noir,{S2}
aime,{S4}
le poisson,{S5}
7. Generate Each Complete Transition Sequence Which Uses Each Member of STS Exactly Once
"le chat noir aime le poisson"
8. The Labels of the Highest Probability Transition Sequence are the Translation

Figure 4.11: Translation Stage III: Put the target transitions together

In the first stage of the research, I used a modified chart parser to put the activated transitions together. Parsing was a matter of checking that begin and end states of transitions to be connected were the same, and making sure that SWSs had a null intersection. But this move to a chart parser lacks a crucial insight: The activated transitions, taken as a unit, are already connected. In fact, these transitions, along with the original start-state and final state information, define an automaton, one which is simply a much smaller part of the original target automaton. Imagine an automaton with thousands of transitions, where a small number of the transitions are chosen. These transitions, along with the original start-state and final state information, also form an automaton.

This insight means that the transitions need not be put back together at all. All we have to do is generate all the allowable (again, those whose SWSs can be combined) sequences of transitions from this newly defined very small automaton which begin at the start-state and end at a final state (this is step 7 of Figure 4.11). These sequences can be easily and quickly generated because of the automaton's small size. We begin at the start state, and store each transition sequence we come to (where a sequence of transitions is equal to the entire path seen so far) in a data structure (which ideally should be some sort of hash-table, indexed either by SWS or begin-state, or both, so that transitions can be quickly recovered as needed). We store the sequences so that they can be quickly accessed should we need them again for working with partial results (i.e., should target parsing fail). We stop going down a given path if the SWS of a transition we come to cannot be combined with the accumulated SWS so far (i.e., if they have a non-null intersection), or when we reach a state that is always final.

Once generation is finished, we choose the most probable complete target parse, i.e., one from the start-state to a final state and that has a full SWS (this is step 8 of Figure 4.11). As with the other translation stages, we now note a few of the finer points of this stage. First of all, the generation process is very fast. This is due to the automaton's very small size, and the fact that we can use fast bit operations when comparing SWSs. Nevertheless, in the event that these smaller automata grew to such a size that the generation began to be too slow, we could decrease the number of transitions we start with: As mentioned in section 4.5.2, we can limit the number of transitions that are activated in the second translation stage, to the n-best for each source transition subsequence, for some suitable n.

We also have held off discussion of the null alignments, those that derive from 1:0 and 0:1 word alignments. 1:0 alignments (empty transitions) require special care in the target automaton, because they do not exist, i.e., they have no state or label information, only an SWS. We will deal with them in section 4.5.3.1 next. 0:1 alignments (the insertion cases) are no problem at all. A 0:1 transition is just like any other, except that its SWS is null. This means that it can be combined with any other transition it is already connected to in the small automaton.

Another issue touched on briefly in section 4.5.2.1 is how to deal with identical transition sequences (i.e., transition sequences which result from "putting together" smaller activated sequences, where the begin state, end state, label, and SWS are identical). How can such identical transitions arise? Such transitions come about because of differences in the word alignments in training sentences, where a given word may be aligned in many ways, and sometimes as part of a larger sequence. We need look no further than the example given earlier in (65), repeated here as (67):

(67)
$$\begin{array}{lll} < a > & \rightarrow & \{<< c >, .2 >\} \\ < b > & \rightarrow & \{<< d >, .2 >\} \\ < a, b > & \rightarrow & \{<< c, d >, .2 >\} \end{array}$$

Given the alignments shown, when an STS contains source transitions a and b, identical transitions will exist after generation; only their probabilities will differ (I refer to these as transitions and not transition sequences because the generation process in essence "smushes" individual transitions together into single transitions, their intermediate state information no longer being relevant). For example, assuming that the labels of these transitions are the same as their names (i.e., that the label of a is a), and that the source sentence to be parsed was ab, the following three transitions would be activated (where i, j, and k are state numbers) after the second translation stage:

(68) $< i, j, c, .06, [01] >$
$< j, k, d, .08, [10] >$
$< i, k, cd, .024, [11] >$

The first two transitions in (68) can be combined to form $< i, k, cd, .0048, [11] >$. As is standard in working with probabilities, since this transition and the last from (68), $< i, k, cd, .024, [11] >$, represent different ways of arriving at the same result (i.e., they are disjoint events), their probabilities should be added (because we want to represent the probability that either would occur), to yield a single transition which we then store: $< i, k, cd, .0288, [11] >$.[20]

4.5.3.1 Handling Empty Transitions in Translation

Empty transitions (those which arise from 1:0 alignments; also known as *deletions* in the sequence alignment literature) require special treatment because they are unlike other transitions: They have no begin state, no end state, and no label. All they have are an SWS, and a probability (i.e. an alignment probability multiplied with a special empty transition probability, see section 4.3.5).

While the number of empty transitions which can exist during any target parsing session is limited to the number of words in the source sentence to be translated,[21] it

[20]Although I show all of the examples with a standard probability distribution (0–1), in the implementation I use logarithms. In the log domain adding is used for joint probability, so special steps must be taken to account for such disjoint probabilities. I follow the method outlined in (Manning & Schütze 1999:337).

[21]Each source word can only generate one empty transition, since the only thing that differentiates one empty transition from another is the source word which generates it (and the alignment probability, but in the table there can only be one pair between a source transition and a given target transition, in this case an empty one), and the translation algorithm uses only the most probable recognition (therefore there can be no more than one empty transition for each source word in the STS).

does not take many such transitions to cause a processing slowdown for a translation system, if they are treated the same as other transitions. This was especially true in the earliest implementation, when a chart parser was used. The reason why empty transitions are potentially computationally expensive is that they are very unconstrained. An empty transition can be "connected" to any other transition, so long as their SWSs have a null intersection. One might envision a 1:0 alignment as a source sentence, saying to a target sentence, "look, I don't care where you put it, but just take care of this word, go ahead and delete it if you want, but you're responsible for it."

The SWSs of empty transitions need to be accounted for, or translations with any 1:0 alignments would not be available. And since the last stage of translation means putting target transitions together in all ways possible, this includes the empty transitions, whose state information is unspecified. Ignoring SWSs, any empty transition can not only combine with any other empty transition, but also with most non-empty transitions. This means that with just a few empty transitions, the number of transitions to process quickly balloons.[22]

In this section, I delve a little more deeply into implementation details, to describe a solution which handles the empty transition problem quite efficiently, so that translation speed is not adversely affected by the number of these transitions.

[22]For example, suppose the target parsing process begins with 100 transitions, which can be "connected" in a number of ways (let's say by a factor of 10), yielding 1000 total transitions. If we add just 10 empty transitions, each which hypothetically could combine with, say, 90% of the original transitions, we go from 100 transitions to 1000 (i.e., $100 + 900$), before generation begins, and assuming this same hypothetical rate at which the transitions can be connected, we would get 9000 transitions instead of 1000.

The key to efficient processing with empty transitions is to realize that they only have limited information to offer (i.e., an SWS and a probability), and to not access that information until the final stage of the target parsing process. Here is how it is done: The only thing that constrains what an empty transition can combine with is its SWS. So what we want is some sort of record of all the SWS information available from empty transitions, so that we need only check them once. What we do is create an empty transition mask.

Suppose a source sentence has n words. Every SWS then for the given translation task is (given a bit vector implementation) a vector of n bits. A full SWS has all n bits with a value of 1. Let's assume we have a five word source sentence. So a full SWS is [11111], an empty one is [00000], one representing just the first source word is [00001], one representing the first and third words is [00101], and so on. An empty transition mask then is just a bit vector which represents which 1:0 alignments exist. If there is one for every word, we have a full SWS; if only some of the 1:0 alignments are present, the mask just has the relevant bits as one. Let's assume for this example that there are three 1:0 alignments, for the first, second, and fourth source words in the five word source sentence. Then the empty transition mask is [01011].

Now, every empty transition also has a probability. These probabilities are different, because while the transition probabilities of empty transitions may be uniform, the alignment probabilities need not be, and the resulting probabilities associated with an empty transition will be some combination of the two (see section 4.3.3). We use a second data structure, a small indexed table, called the *empty transition probability hash*, to store the probabilities. This hash is indexed by the SWS values. In our example, with three empty transitions, the hash might look like:

	[00001]	$\rightarrow$	*some_probability_value*
(69)	[00010]	$\rightarrow$	*some_probability_value*
	[01000]	$\rightarrow$	*some_probability_value*

With these two small data structures, the mask and the hash, target parsing is easy. We first complete parsing without the empty transitions. Recall that this leaves a store of longer transitions, which might typically be indexed by their SWSs values (see section 4.5.3).[23] Assuming the empty transition mask is non-empty (otherwise we need not worry about empty transitions at all, since there are none), we then simply make one pass through all of these longer transitions, checking to see for each transition that has a non-full SWS, if unioning its SWS with the mask (this is done with a bit-wise *inclusive or*) would create a full SWS. If so, we have found a target parse.[24] This process is made even faster (as done in the implementation) if the data structure storing the (non-empty) transitions is indexed by the SWS, because rather than iterating through all the transitions, one need only check once for each SWS that exists in the store (since the answer will be the same

[23]One can view this store as a chart, a hash, or whatever is convenient for the earlier stage. The overall strategy here is not dependent on the target parsing method.

[24]I describe how this process works for extracting partial results (i.e., when we have failed to find a parse, and are concerned with the best incomplete parses we can find), in section 6.3.4.

for all the transitions indexed by this SWS). For example, if the transition store contains a transition with an SWS of [10111], then unioning with the mask ([01011]) will yield a full SWS, thus a successful target parse.[25]

For each such transition (i.e., one which would have a full SWS when unioned with the mask) we create a new transition (i.e., a new parse). All that we have to do is adjust the probability, and for this we use the empty transition probability hash created earlier. We get the bit positions which were used from the mask,[26] convert them to an SWS for each bit position, and get the probabilities from the hash. The probabilities are then combined just as they would have been if the transitions had been chart-parsed together (i.e., they are multiplied; or added in the log domain). So, from the transition with the SWS [10111], we get a new transition (a complete parse, assuming the start-state and final state information are correct) with a full SWS, and a probability that is the original transition's probability multiplied by the probability that existed in the empty transition probability hash for [01000].

Thus, by using this mask and probability hash method, at the end of the target parsing process, we get the identical parses (complete transitions, those from a start-state to a final state with a full SWS) that we would have gotten by treating empty transitions as regular transitions, but without the computational overload. 1:0 alignments become unproblematic for the linked automata model to process.

[25]Indexing by state information will also be fast, because we can check just those transitions that begin at the start-state and end at a final state, although the SWS technique proves more useful, since we often want to be able to work with fragments anyway; see Chapter 6.

[26]This can be done in a number of ways, e.g., we use a bit-wise *exclusive or* between the original non-empty transition's SWS, and a full SWS: $xor([10111], [11111] = [01000])$; one could use a bit-wise *logical not* with the original non-empty transition's SWS, etc.

4.6 Comparing the Linked Automata Model to Previous Work

I began to locate the linked automata MT model with respect to other data-driven models in section 4.1. The model is a statistical MT system, which takes advantage of finite-state techniques by using automata as the language models and links between the automata transitions as the translation model. Thus, it is most closely related to the finite-state models described in section 2.2.2. In this section I compare important aspects of the model with these earlier systems, concentrating mainly on the data-driven finite-state models. I also discuss the linked automata model's relationship with EBMT.

4.6.1 Pure Statistical Machine Translation

Like all of the probabilistic finite-state systems, the linked automata model borrows the ideas of the combination of translation and language models from SMT. But unlike the SMT model (i.e., the IBM Model 3 in particular (Brown *et al.* 1993)), the linked automata model does not have uniform structure. Rather, the two automata (the source and target language models) capture the phenogrammatical structure of the sentences, a linguistically motivated step (as such, the linked automata model might be viewed as existing somewhere between the word-to-word and syntax-to-syntax mapping levels in the MT hierarchy shown earlier in Figure 2.1). Additionally, while for the IBM SMT approach proper word alignments can be viewed as the overall goal of the system, the linked automata model begins with word-aligned bitexts.[27] Word alignments are clearly represented via links between transitions, but they are only part of the story.

Another departure from pure SMT is that in the finite-state methods in general, there are no separate parameters for notions such as fertility and distortion, or for other potentially relevant factors not parameterized in IBM Model 3. These notions are implicit in the finite-state models, and thus for the linked automata model as well.

A major benefit of the proposed system over pure SMT is its simplicity. It is much more understandable and modular, making further development more straightforward. One area where the SMT model should generally be superior is in terms of scalability, since the linked automata model can tend to grow quite large as the number of training examples increases. This rapid growth can be constrained, however, through generalization, which has the positive side-effect of reducing the size of the system.

4.6.2 Probabilistic Finite-State Models

In section 2.2.2 I described several (mostly) finite-state translation models, including 1) stochastic inversion transduction grammars (which are not finite-state, but close enough in spirit to warrant discussion, section 2.2.2.1); 2) composed transducers (section 2.2.2.2); 3) subsequential transducers (SSTs, section 2.2.2.3);

[27]In fact, it would not be unreasonable to use the IBM algorithm first as a word aligner for the linked automata system, if its accuracy were high enough. This is experimented with in section 3.2.3.

4) weighted head transducers (HTs, section 2.2.2.4); and 5) a model for lexical selection and reordering (section 2.2.2.5); as well as a hybrid finite-state model (where some of the finite-state devices are hand-crafted, in section 2.2.2.6). Since the composed transducers of Knight & Al-Onaizan (1998) were intentionally a reimplementation of IBM's pure SMT model, it would be redundant to compare the proposed system to it. The clear motivation for Knight & Al-Onaizan's (1998) work was to demonstrate that the SMT model could be implemented in a very understandable and well researched finite-state framework. Nor will I compare the system to the hybrid system of Vogel & Ney (2000), presented in section 2.2.2.6, but as stated in that section, many of the techniques used in that research for incorporating human knowledge into an empirical system may be applicable for later research stages of the linked automata model.

4.6.2.1 The Linked Automata Model and SITGs

Direct comparison with the stochastic inversion transduction grammars (SITG) of Wu (1997) is difficult, but the proposed system tries to capture information much the same way as the SITG model. In the SITG system, a bilingual grammar is used to reflect the relationship between the two languages. This model combines the language and translation models into one, but the fact that different sets of symbols (i.e., different sets of non-terminals) for the two languages can be used in rule productions, as well as the fact that these symbols can occur in different orders relative to the two languages, coincides with the same information that the linked automata model tries to express: An MT model should have a means to represent the relationship between source language and target language words, yet it should not impose the ordering constraints of one language on the other. The

SITG formalism, although context-free, is somewhat less flexible than the proposed model, since some (generally unlikely) alignments are not permitted, and given its context-free nature, the SITG model is not as easily made efficient as the finite-state methods (at least potentially).

4.6.2.2 The Linked Automata Model and Subsequential Transducer Models

More direct comparisons can be made with subsequential transducers and head transducers. In fact, the linked automata model is most closely related to the SSTs of Vilar *et al.* (1999) and Amengual *et al.* (2000). Both the linked automata and SST models use a single, large finite-state system, and both are constructed from word-aligned bitexts. The main difference between the two models, as discussed in section 4.1, is that the SST model uses a transducer, whereas the linked automata model uses two automata, a design more naturally suited for translation. Because of this difference, the SST model must use special symbols (see section 2.2.2.3) in the transducer's output labels, to indicate the proper ordering for the target language words (as well as to attempt to handle discontinuous alignments (Sanchis *et al.* 2001), see Chapter 7). In addition to the added theoretical and computational overhead of this step, it is not clear that the SST model is easily reversed, to allow for translation from target language to source language. But again, the main point is that the linked automata model is simply a better fit for translation. The alignments between words fit naturally into its design, making the system more clear and the probabilistic modeling more appropriate to translation.

Another large difference between SSTs and the linked automata model is that the former are subsequential, while the latter is clearly not. Thus, the SST model has a determinism that allows for very efficient processing and generalization techniques, as well as straightforward probabilistic modeling. Unfortunately, it is not clear that deterministic finite-state devices are sufficient for natural language translation (i.e., natural language translation is not a subsequential function). For this reason, the use of SSTs has been restricted to artificial unambiguous limited domain languages. The linked automata model does not have these restrictions, but with its added complexity comes a price to be paid in terms of less efficient algorithms for training, translation, and generalization.

Both models are similar in that they accomplish generalization to unseen inputs mainly through merging of states or transitions, but because of its determinism, the SST model is able to merge to a greater extent, and thus is more scalable than the proposed system. The SST system described in Vilar *et al.* (1999) also makes use of an error model to restrict generalization, so that the source language model does not overgenerate strings which are highly unlikely. This problem is very relevant for the linked automata model, because generalization via merging vastly increases the size of the search space for parsing the source sentence. It is not clear if the SST error model technique is appropriate for the linked automata model, but the idea is an important one which needs to be further studied.

4.6.2.3 Comparison with Weighted Head Transducers

The weighted head transducer approach of Alshawi *et al.* (2000) and Alshawi & Douglas (2000) is much more complex than the linked automata model and the SST model. It uses many small transducers applied recursively, and attempts to model the hierarchical syntactic structure via dependency trees (see section 2.2.2.4). In fact, it was the Alshawi *et al.* (2000) model that originally inspired the research which resulted in the linked automata model, to see if a simpler translation model could be sufficient. Of all the finite-state models, the Alshawi *et al.* model (and the related Bangalore & Riccardi (2001) model) would probably be the most interesting, notwithstanding its complexity, if it could be guaranteed that the hierarchical alignments represented by the dependency trees were correct; but it is not clear if, as presently formulated, these hierarchical alignments capture much more information than do the word alignment and linear ordering information contained in the simpler finite-state models (see the example in section 2.2.2.5). One benefit in the use of many small transducers is better scalability than that of the linked automata system.[28]

The HT model must begin with bitexts and perform its own hierarchical alignment, which is a more complex process than word alignment. And given the recursive nature of the model, the algorithms for translation are more complicated than for the linked automata model (although both make use of dynamic programming to search through the space of translation possibilities). As demonstrated by good preliminary results, the HT model appears to be effective. The question posed by the proposed linked automata system is: Can a more simple system perform reasonably as well? The HT system stores all the complexity in the model itself. The linked automata system makes the translation and language models very simple, but allows for more sophisticated techniques to use them. Thus, in some senses the

[28]Thus, as presently described, both the SST and HT models will scale better than the linked automata model—a problem which needs to be addressed through increased generalization (i.e., less strict merging).

two approaches are the same, but we see the separation of model from translation algorithm in the proposed system to be a benefit, since it makes the system more understandable and easily modified.

In terms of increased coverage, the HT system and the linked automata system use some of the same techniques. Much generalization, as in the SST system, is achieved via merging. Merging in the HT system is much less conservative as compared to the linked automata model, allowing for greater coverage, but possibly at the expense of translation quality. Different (less conservative) merging strategies will continue to be tested for the linked automata system, until the system converges on the best translation results. Both the HT model and the linked automata model also use what we term *partial source parsing* (as described in section 6.3.3), for inputs which cannot be recognized. The techniques described are similar; matching the longest sequence possible and recursively handling the remaining parts. Alshawi *et al.* (2000) do not describe the use of heuristics for dealing with unknown words, but it is likely such techniques could easily be incorporated into their system.

4.6.2.4 Comparison with Lexical Selection and Reordering

Like the head transducer model, the lexical selection and lexical reordering (LSLR) model of Bangalore & Riccardi (2001) (presented in section 2.2.2.5) is a complex empirical finite-state approach which attempts to model hierarchical syntactic structure. In fact, it uses the same alignment strategy of mapping source and target dependency trees as does Alshawi *et al.* (2000). In this respect, comparison of the linked automata model to the LSLR model is much the same as to the HT model.

In both the LSLR and HT cases, it is not clear if the automatically induced dependency trees are accurate enough to provide better translations than can be derived from word alignments alone.

Like the HT model, the LSLR more easily generalizes than the linked automata model because, from the very outset, it aligns phrases rather than word sequences,[29] thus does not require a separate generalization strategy, as do models which only model phenogrammatical syntactic structure, such as the SST and linked automata models.

What the LSLR model and the linked automata model share is the separation of the language and alignment models. The lexical selection model of LSLR is like the alignment model (i.e., the table) of the linked automata approach, and the lexical reordering model is similar to the target language model (the target automaton) of the linked automata model. There are some crucial differences, however, beyond the aforementioned modeling of hierarchical structure. The LSLR models are in a sense more separate: The two models may be built from different bilingual corpora, whereas for the linked automata model, the same corpus must be used in construction (since it all takes place at once). So, in the linked automata approach, the language and alignment models are more tightly coupled than in the LSLR approach. Even though the LSLR models are in this sense more separate, the fact that both are transducers allows them to be composed, making decoding more straightforward (and likely more efficient) than for the linked automata model (see, for example, the linked automata translation process as described in section 4.5).

[29]In the case of the LSLR, this generalization strategy can be viewed as more of an approximation than in the HT case, however, because the phrases themselves are approximated with strings.

The fundamental question of comparison with the LSLR model, as with the HT model, remains whether the hierarchical alignments improve the translation process. Certainly they should, if the automatically induced alignments are reliable enough (which continues as an open question for these methodologies); and using such hierarchical (i.e., tectogrammatical) syntactic information appears to be the direction in which many statistical MT approaches are heading (see the discussion of Yamada & Knight (2001) in section 2.2.1). And it may be the case that more syntactic information can be incorporated into the linked automata model (see Chapter 7). Taking these steps will only be sensible, however, once it is clear how far the model in its most basic form can be pushed as an MT system.

4.6.3 Example-Based Machine Translation

The relationship between the linked automata model and EBMT is not as transparent as with the finite-state models. On an abstract level, they are quite similar, in that they learn from data and perform little linguistic analysis of inputs during translation. And indeed, the finite-state models could be viewed simply as storing examples just as the EBMT models do, but the focus of the general approaches is quite different. EBMT systems intentionally view examples blindly (training examples need not be word-aligned), as unanalyzable chunks to be retrieved via analogy with inputs. Statistical finite-state models seek to store data in a framework which captures more of the linguistic information inherent in the data, namely word alignments and linear ordering information. As such, the finite-state approaches may be more appropriate for pairs of languages where word order is relatively fixed, but may yield little improvement over EBMT for other types of language pairs, at least in terms of performance, but not necessarily understandability and the ability to be further developed.

Where the linked automata model bears the strongest relationship with EBMT is in terms of the increased coverage heuristics described in section 6.3. For example, the techniques of fragment processing and partial source parsing are exactly analogous to the EBMT techniques used in the matching stage. In EBMT the goal of the matching stage is to partition the input in some well-motivated fashion and use the associated examples. In the linked automata system, fragment processing matches inputs to subparts of training examples, and partial source parsing finds the best substring and recursively handles the remainder. Similarly, partial target parsing in the linked automata model is related to the recombination stage of EBMT: The translations of the associated examples must be put back together in a way consistent with the target language. The linked automata model uses the target language model to assist in this process. EBMT recombination algorithms use various other strategies for putting these translated parts together, but the goal is the same. Thus, the techniques of EBMT matching and recombination may offer insights for the linked automata model's increased-coverage heuristics.

4.6.4 Summary

In summary, the linked automata model, like other finite-state MT models, is a natural outgrowth of the pure SMT models. It makes use of translation and language models, but constrains the models to a finite-state shape which is more appropriate to natural language translation. The linked automata model is more expressive than the SST finite-state model, and is a better fit for translation than are transducers in general, because it naturally represents the word alignment between sentences without imposing the ordering constraints of one language upon the other, thus also allowing for a more direct representation of discontinuous alignments between word sequences. The model is much more simple than the HT model, since the large, paired automata represent the translation and language models, as opposed to the recursive application of smaller transducers. The linked automata model does not, however, model the hierarchical syntactic structure, as the HT and lexical selection and reordering models do. The hypothesis to be investigated is if this simpler, more direct approach can produce a reasonable natural language translation system. Lastly, while the system is not an EBMT system, it does share the ideas of matching and recombination with EBMT, when inputs cannot be completely processed.

CHAPTER 5

PRELIMINARY EVALUATION

There is no such thing as the correct translation (King 1997:261).

5.1 Introduction

There appear to be as many different evaluation methods for machine translation as there are machine translation methods. This state of affairs arises from the fact that there is little agreement on how to define what a correct translation is, much less how to measure it, be the measurement automatic or by human judgement:

> One of the moments that many MT research presentations have in common is the viewgraph that makes the assertion that there is no standard method for evaluating machine translation .... There are indeed many well-known MT evaluation methods .... But it is quite correct to assert that none are universally accepted as standard. It is also true that the useful ones take considerable effort, cost, and time to perform, and that none of the methods tell us all of the things different people might need to know about an MT system .... The best known example of why there isn't such a thing has to do with the fact that MT, unlike information extraction, topic or document detection, optical character recognition or speech recognition, doesn't have a possible "ground truth" .... This is not because no one has bothered to devise such a set of data, but rather because it cannot be done in a straightforward way (White 2000:100–101).

Further compounding the problem, it is common in the MT literature for researchers to discuss their systems in terms of "accuracy" or "coverage," with no definition of what these terms mean. These problems with evaluation are so long-standing and commonplace that MT evaluation (MTE) has become a small research niche in its own right, but, perhaps as should be expected, these efforts have yielded few accepted results. Researchers generally return to one of the fundamental truths of MT evaluation—that (as the quote at the beginning of this chapter indicates) just as there is more than way to express the same idea in a single language, there is more than one way to translate a sentence from one language to another; so there can be no unique, correct translation.

One aspect of MTE on which some consensus has been reached is that evaluation methods should vary depending on the reasons for the test. That is, the role of the MT system and its stage of development should play a part in determining the appropriate evaluation method. I begin with a brief discussion of the different types of MTE and their history. Next, I discuss the sorts of dimensions along which MT systems should be evaluated. In the final section, I present a simple, automatic evaluation method for the proposed linked automata model, and present some feasibility results.

5.2 Types of MT Evaluation

Probably the single most important event in the history of MTE was the publication of the ALPAC report (Pierce *et al.* 1966). The report had the effect of damning MT research in the United States for nearly 20 years (Arnold *et al.* 1994), and served to set a precedent for the methods of MTE. In that evaluation, human judges were asked to rate human and machine translations from Russian to English using a rating scale, in terms of two attributes, *intelligibility* and *faithfulness*, where one could informally take intelligibility to mean: *Is it a good sentence of English?* and faithfulness to mean: *Does the English translation have the same meaning as the Russian original?* Intelligibility was rated on a nine point scale and faithfulness on a ten point scale, in terms of informativeness. This second measure turned out to be somewhat counterintuitive, because judges were asked to measure how informative the original Russian version was compared with the translation, to gauge a degree of informativeness of the translation (i.e., had something been lost in the process). As King (1997) points out, although the evaluation was carefully designed, the methodology was flawed, since the human judgments were highly subjective, and dependent on the quality of the human translations, which could vary with the translator; and, more importantly, because the *fidelity* (we substitute this more common term for *faithfulness*; *accuracy* is also used in the literature) task was confusing and is inherently hard to measure.

Human evaluations, like the ALPAC judgments, have historically been the most common in the published literature, especially across different MT platforms. The other type of MT evaluation is *automatic*, i.e., carried out without human intervention. Within these two evaluation groups are many different types of tests, and in general the tests for human evaluation and automatic evaluation are quite different, since human and computer capabilities are so different. Human evaluation is time-consuming, expensive, error-prone, and can be very subjective. Nevertheless, if the resources are available, human evaluation is often the best as well, because of issues such as fidelity. For example, suppose we have a Russian utterance, whose expected (i.e., correct) English translation is:

(70) The small boy likes pop.

Next suppose that an MT system produces the following translation:

(71) The little lad is fond of soda.

A human judge might tell us that this is an excellent translation. But how would a computer fare? A computer which is given the reference translation (70), and checks only in terms of the number of keystrokes it would take to convert the resulting translation, (71), to the reference translation might judge the translation to be poor. Thus, humans will for the foreseeable future always be better judges of fidelity. Still, these fidelity judgments are quite difficult, since only in very rare cases (e.g., limited domains) can fidelity even be defined (King 1997). One key for getting human judgments which can be relied on is to get many judgments, and to apply statistical techniques so that it can be determined whether the judgments converge. The problem with this idea is that it makes getting human judgments even more expensive and more than a typical MT researcher wants to do to see if a slight change in the system has produced any benefit.

Since the ALPAC tests, human evaluation methods have generally improved, but are still difficult and error-prone, even when the resources are available. White & O'Connell (1993) report methods for the DARPA 1992 MT test and 1993 MT pretest. In the 1992 evaluation, accuracy was creatively measured by having mono-lingual participants read translated texts then take a multiple choice test to see how much information was accurately transmitted. Fluency of the translations was graded on a numerical scale. In the 1993 pretests fluency was judged similarly, but accuracy was judged as compared to reference (baseline) texts. Still, these tests have been viewed as highly subjective (Melamed 2001). The search for less subjective human evaluation methodologies continues today, with approaches ranging from methods inspired by tests used in language learning (e.g., those tests used to assess the abilities of government linguists, see Vanni & Reeder (2000)), to specialized evaluation tools that display a range of translations, which evaluators can compare against reference translations (Niessen *et al.* 2000).

In the world of automatic evaluation, tests are typically more modest.[1] One might begin with the same types of tests as used with human judges, but, as alluded to earlier, complications immediately develop. First, take the dimension of intelligibility, i.e., is the translated sentence a fluent instance of a target language sentence. Many MT systems develop a means to test target language fluency, namely a target language model. Unfortunately, target language models are always less than perfect, so how much can we rely on them as automatic evaluators? More importantly, if we use the same target language model as we did in the MT system itself, our approach would be circular: We would be using the same system to make the translation and evaluate it. Thus, using a target language model to evaluate intelligibility would require a separate language model, an enterprise many researchers might not want to undertake, just for the sake of evaluation.

[1]A notable exception is Akiba *et al.* (2001), where a decision tree is trained (with human evaluated translations) to evaluate translations based on the values of 16 different edit-distance measures.

One very interesting recent MTE approach which only attempts to measure intelligibility is presented in Corston-Oliver *et al.* (2001). They point out that humans can typically very easily distinguish between MT output and human-translated output, and suggest that the well-formedness (i.e., intelligibility) of the output can be viewed as a classification problem. They use two types of measures: perplexity, using lexicalized trigrams and part-of-speech trigrams; and (what they call) linguistic features, which included many tree-structure related features (number of nodes, amount and type of branching, etc.), number of modifiers, average lengths of certain constituents, ratios of different types of words (e.g., function words to content words), and more. Using these sorts of features, Corston-Oliver *et al.* (2001) build a decision tree to distinguish between human and machine translations, and achieve an accuracy of nearly 83%. Their fully automatic approach (which requires a bit of computational machinery for some of the feature measurements) may prove useful for more sophisticated MT evaluations (i.e., at the later stages of a research project), where it may be important to pinpoint what sorts of constructions are problematic for a system in terms of its intelligibility only.

This brings us to the harder question: accuracy. It is difficult to correlate distance metrics between translations with human judgements of translation quality (Brew & Thompson 1994). While we can easily automatically calculate how much a resulting translation differs from a reference translation, in terms of keystrokes (keystroke distance, see section 2.3), or words (edit-distance or word-error-rate), how do we automatically tell if one sentence has the same meaning as another? This was the problem we identified with examples (70) and (71). To do so would require the ability to parse utterances, get semantic representations, and compare them, judging a kind of semantic distance. Again, this would mean that we have this kind of computational machinery at our disposal, which is not always the case, and, in addition, the same circularity argument would arise: If we used the same machinery as in our MT system, the evaluation would be flawed, so we would need separate syntactic and semantic modules for the evaluation. Even in the face of these difficulties, we should not understate the desirability of automatic MT evaluation. The average MT researcher simply does not have the time or resources to do frequent human evaluation. Automatic MT evaluation is (or should be when done properly) efficient, inexpensive, reproducible, and most importantly, objective. Quality automatic evaluation is the holy grail of MTE (Vanni & Reeder 2000).

5.3 Other Aspects of Evaluation to Consider

Given the difficulty with human evaluation in terms of cost and objectivity, and the shortcomings of automatic evaluation in terms of the breadth of areas that are testable, an intermediate goal becomes finding ways to constrain and decompose automatic evaluation so that the results are meaningful. As it turns out, the meaningfulness of a given test is dependent upon the stage of the development of the MT system, and its intended uses. White (2000) identifies five types of MT evaluation: 1) feasibility evaluation, 2) internal evaluation, 3) declarative evaluation, 4) operational evaluation, and 5) usability evaluation.

A feasibility evaluation's purpose is to decide if a new MT approach has any chance of success; therefore, it is not unreasonable for such tests to be confined to a smaller set of possible outcomes. Of all the evaluation types, feasibility evaluations are the most easily automatically evaluated, since the smaller, bounded set of outcomes means that resulting translations can be compared directly to reference translations, when the correct answers are known. An appropriate test here may be edit-distance.

Internal evaluation has as its goal to be able to discern if a system is improving (e.g., is its coverage increasing) and if the parts of the system are working as intended (i.e., do different modules function as designed). This sort of evaluation will occur continually during the development of a model. Automatic internal evaluation is more difficult. If an MT system has different modular components (i.e., several different units, where each unit has an output that it feeds into the next), such an evaluation may mean that at each stage an expected output needs to be compared with an actual output. Human interpretation will likely be necessary to figure out what the cause of errors are, but again tests such as edit-distance can be used. If the MT system under consideration is simply a black box, i.e., a source text goes in and a target text is returned, with no intermediate outputs, then internal evaluation may behave exactly as feasibility evaluation, where reference translations are simply compared to resulting translations. This type of testing may be sufficient to gauge accuracy improvement and increased coverage, and White (2000) suggests it may be possible to have automatic tests which also identify reasons for failure.

A declarative evaluation purports to identify the actual performance of the system, to quantify the system's behavior in terms of fidelity and intelligibility. After a declarative evaluation, a researcher would expect to be able to claim that their system was $x\%$ accurate, whereas for feasibility and internal evaluations, only some degree of success or improvement needs to be demonstrated. As discussed in section 5.2, these sorts of measures are notoriously difficult to get automatically, but again there are some possibilities (e.g., the use of target language models to test intelligibility).

An operational evaluation is intended to determine if an MT system is suitable for the purpose for which it was created. This sort of evaluation deals with more engineering-like tasks, such as compatibility with related systems, and managerial tasks, such as, would such a system be cost-effective. Some of these tasks could be handled automatically, via regression testing with related software, and costs may be able to be automatically estimated. But clearly these sorts of concerns are beyond the typical domain of a new research initiative, such as the linked automata system presented in Chapter 4.

The final type of evaluation is a usability evaluation. These sorts of evaluations aim to measure such aspects as *utility* and *satisfiability*, and would aim to answer questions regarding how easy it is to use and learn such an MT system. These sorts of questions are certainly beyond automatic evaluation, and beyond the needs of the proposed model.

Given these five evaluation types, I will attempt to identify which is relevant for the proposed system in the next section, but there are first some remaining aspects of MT evaluation to consider. In the MTE literature, discussions usually center on issues of fidelity and intelligibility. Focusing on these sorts of performance measures makes sense, since if a system does not perform reasonably well, other issues are irrelevant. Nevertheless, it is surprising that, given the wealth of literature on MTE, including the discussions that accompany nearly every MT proposal, there is very little discussion of such issues as efficiency (both in terms of time and space), applicability (to different language pairs), ease of implementation, and scalability (related to efficiency). A notable exception is Hovy (1999). Creating an MT system that is only relevant to a single language pair, that does not scale reasonably, or that takes hours to return the translation of a sentence means the system will likely not be useful, no matter how good its performance in terms of fidelity and intelligibility. Evaluating these sorts of non-performative aspects is not a question of human or automatic testing, but rather a factor that should be considered in all aspects of development. This means that performance cannot be considered in isolation, and that during the development of the proposed system, such aspects need to be considered both internally and in comparison with other systems.

The final aspect of MT evaluation to consider is comparative evaluation. Unfortunately, evaluating one MT model against another is often difficult, except in abstract terms. For example, one may on occasion be able to demonstrate by means of a proof that one system is more adequate than another, but in general, the most reliable and believable means of comparing systems are in head-to-head competitions with the same training and test data. Such competitions are difficult to create, with the exception of some of the sponsored evaluation efforts like those by ALPAC and DARPA, since often both the systems and the data used are proprietary (e.g. SYSTRAN and VERBMOBIL), or simply hard to replicate, and many researchers are not eager to share code for their complete systems.

Fortunately, there has been an effort by some researchers to make comparative MT evaluation more practical. In the summer of 1999, several of the most prominent MT researchers came together in a workshop at Johns Hopkins University (see Al-Onaizan *et al.* (1999)), where one of the goals was to create an MT toolkit which would not only assist in the development of MT systems, but also provide an implementation of IBM's Model 3 (Brown *et al.* 1993), so that researchers could compare their systems' results with another well-known system, without having to rebuild it. Unfortunately, a decoder for the toolkit was never completed, leaving the system able to provide word alignments, but not translation. Another means to perform comparative evaluation may be by sharing data, and comparing the results with earlier published results. Melamed (2001) has made such training data available, in the form of a *gold standard* of hand-aligned French and English bitexts (although this data would likely be most useful for comparisons in terms of word alignment, rather than translation). The use of such resources may one day make comparative evaluation an attainable goal.

5.4 An Evaluation Method for the Proposed System

Having described some of the different types of MT evaluation and important issues to consider, we now turn to the evaluation of the proposed linked automata model. As discussed in section 5.3, there are several different stages in the development of an MT system, and evaluation techniques should vary accordingly. Thus, we first need to identify which stages are relevant for the dissertation research. Certainly, during the development of the prototype system, the MT development was in a feasibility stage. Automatic evaluation methods were used at this time and were quite suitable, since, naturally, early results were poor, especially prior to any generalization, and small changes in the model could yield large changes in test scores. For the early stages of the dissertation research, development continued to be at the feasibility stage, since a number of new techniques were tried, and some of the existing techniques were reimplemented to see if they could yield improved efficiency. Automatic evaluation is called for in these cases; in fact, the goal should be to replicate earlier tests exactly, so that any improvements can be judged without bias.

In later research, the development moves to the internal evaluation stage. We will want to assess not only if the approach is feasible, but if it is providing the improvements we expect. For example, in the final stages of the research, we further hybridize (here *hybridize* means adding more linguistic information, not human-coded information) the model, in experiments using additional linguistic features, such as part-of-speech tags (see Chapter 7). While we will want to know if such an approach is feasible, we will also need to be able to discern if it results in significant enough improvement to warrant its use. The hypothesis at this point is that translations will improve, but that translation error rates will be high enough so that automatic evaluation methods are appropriate, i.e., we hope to see changes with some of the methods, since there will be ample room for improvement (given that without the use of generalization and other increased-coverage techniques, the system is intentionally designed to provide only rough coverage). Automatic evaluation appears to be most suitable at stages where there are large leaps in performance, and human judgments are most warranted for finer grained distinctions. Additionally, many of the non-performative aspects we will be evaluating, such as translation time and the overall size of the system (i.e., efficiency and scalability), are best evaluated with automatic methods. Lastly, the current reality for the research is that human evaluation at this stage would be too costly, and should be considered premature until the system approaches the performance of the better systems available today.

If automatic evaluation is appropriate, the next issue is what sort of automatic evaluation. First, the design of tests should follow typical practice in machine learning, as was done during the development of the prototype system: Given word-aligned bitexts, a large portion should be used for training the system, and a smaller remainder should be set aside for testing. Of course the system should never be trained with test data. We may also set aside a third set of data for the fine tuning of parameters. This methodology will provide us with test bitexts where there is always a single, correct reference translation that is expected.

We will be testing the system often, thus we want our tests to be efficient. As is often done in MT research, especially in cases where results are expected to be less than perfect, string-based evaluation metrics appear to be the most appropriate (see, for example, Alshawi *et al.* (2000)).[2] It is important to remember that string-based metrics can give only a rough idea of translation quality, as mentioned in section 5.2, since in terms of accuracy, they can, for example, give poor scores even when the meaning is right, if unexpected words are used; and similarly, in terms of intelligibility, give very high scores to sentences that are ungrammatical.[3] Even given these imperfections, string-based metrics seem to be the best choice for this stage in the model's development, because they will still allow us to gauge the impact of modifications to the model on its performance, and to test it frequently and automatically.

[2]Matching techniques from EBMT may provide insights for more appropriate metrics.

[3]Instances of evaluation metric coarseness can be found in the appendix of translation examples (see Appendix A, on page 280). For example, in (A.9), *lamp* counts as a full error even though the correct word is *lamps*.

When using string-based metrics, it is also important to consider the types of data being tested, which for MT means which language pairs. For the proposed linked automata model, we hypothesize that it will be most effective for fixed word order languages, and therefore we will test it mainly as a translator from English to Spanish (and in some cases the reverse, in Chapter 7). These language pairs are chosen partly because data is available, and partly because they should be relatively easy. Since the system will be in feasibility and internal development stages, it makes sense to use language pairs that offer it the best opportunity to demonstrate its capabilities. Given these types of language pairs, string-matching metrics are appropriate, but they would not be adequate evaluation measures, if, for example, one of the languages was highly agglutinative (languages in which the morphology is such that an entire sentence may consist of just a few multi-morphemic words, such as Turkish), even if the system were able to handle such language types.

Perhaps the most common type of string-matching in natural language research is *edit-distance*, the minimum number of insertions, deletions, or substitutions to convert one string to another (see Kruskal (1999) for a discussion on how to compute edit-distance and sequence comparison in general). In this case (and given these types of language pairs), the edit-distance should be measured in terms of words (although for a language like Turkish, characters would be a more appropriate measure). I will use edit-distance as the basic measure, with some slight adjustments to make results more easily comparable (i.e., to yield a number between 0 and 1, where good scores have higher numbers).[4]

[4]Given the MTE problems identified in sections 5.1 and 5.2, I was reluctant to introduce any new evaluation measures, and thus add to the confusion. I therefore selected what I feel are good automatic measures which are easy to understand and replicate, and have been used

Alshawi *et al.* (2000) suggest using a measure related to edit-distance, but amended so that it is more appropriate for translation evaluation. They suggest that *transpositions* should be counted along with insertions, deletions, and substitutions, so that a deviation where two words are misordered is counted as only one error, rather than two (i.e., an insertion and a deletion, or two substitutions), to avoid double-counting of errors. I will refer to an edit-distance which counts the minimum number of insertion, deletions, substitutions, or transpositions as the *transposition-edit-distance*. Akiba *et al.* (2001) also employ edit-distances which count transpositions (they use the terms *interchange* and *swap*), using it as one of four different variables (along with such techniques as restricting the edit-distance comparison to content words and keywords) for a set of 16 different edit-distance measures. One way to make such transpositions easy to calculate is to count them by looking at the alignments after a more traditional edit-distance (one limited to insertions, deletions, and substitutions) has been performed (Bangalore & Riccardi 2001).[5] I follow Bangalore & Riccardi (2001) for transposition-edit-distance calculations in the dissertation (and therefore also for translation-accuracy, see (73), below), in treating an insertion of a token at one location of a string and a deletion of the same token at another (not necessarily adjacent) location in the string as a transposition.

significantly in other research (Alshawi *et al.* 2000; Alshawi & Douglas 2000; Bangalore & Riccardi 2001), namely *simply-accuracy* and *translation-accuracy*.

[5]Another aspect of edit-distance which one might want to vary is the weighting. Not only can insertions, deletions, and substitutions be weighted differently (I weight each at 1, for all results in the dissertation), but one could also weight individual words differently (e.g., one might want to weight content words more than function words; in the dissertation, I weight all words equally). These notions are not unlike those possibilities presented for different types of word alignment evaluation in section 3.3.

Edit-distance returns a natural number (e.g., 0 or 5), but the meaning of these numbers depends on the length of the sentences being tested. So, in addition to edit-distance, we would like to have a measure that we can compare across test examples, that returns a value between 0 and 1. Following Alshawi *et al.* (2000) and Bangalore & Riccardi (2001), I use *simple-accuracy* (SA), and *translation-accuracy* (TA), as defined below, where I, D, and S are the number of insertions, deletions, and substitutions, respectively, between a resulting translation and a reference translation; I' and D' are insertions and deletions if transpositions, T, are taken into account, and R is the length of the reference translation.

(72) $$\textit{simple-accuracy} = 1 - \frac{\textit{edit-dist}}{R} = 1 - \frac{I + D + S}{R}$$

(73) $$\textit{translation-accuracy} = 1 - \frac{\textit{transposition-edit-dist}}{R} = 1 - \frac{I' + D' + S + T}{R}$$

Intuitively, SA can be thought of as measuring what percentage of words are correct and in the proper position in the translation. Note, however, that these measures do not strictly yield a number greater than zero, since it is conceivable to have a translation which is wrong by more words than R. In such instances we treat the distance as R, thus SA and TA are 0.[6]

[6] To ensure that simple-accuracy and translation-accuracy always yield a non-negative number, one might be tempted to use some worst possible edit-distance or transposition-edit-distance number, W, in the denominator of (72) and (73), in place of R. The problem with this strategy, however, is that W is impossible to calculate given unconstrained word alignments, because a resulting translation could be made longer, for example, if one more 0:1 alignment had been used in the translation. My solution, as mentioned above, is to treat any negative numbers, if they arise, as 0, the lowest possible score. An equivalent approach which would guarantee a result between 0 and 1 would be to use R as the denominator, unless the edit-distance (or transposition-edit-distance) was greater than R, in which case the distance number would be used as the denominator, thus yielding $1 - \textit{distance}/\textit{distance} = 0$.

In summary, we have our automatic evaluation metrics, simple-accuracy and translation-accuracy, for the development stages which the proposed system will be in during the course of the dissertation research. In addition, evaluation will also consist of assessment of those non-performative (i.e., non-accuracy related) criteria identified in section 5.3, such as efficiency, ease of implementation, and scalability. Efficiency (in terms of time) can simply be measured automatically during the testing phase by collecting run times. Space efficiency and the related scalability will need to assessed as the system develops, i.e., as the number of training examples increases, how fast does the size of the system increase and how much do average run-times increase. These measures will also need to be assessed each time the model design is varied. Evaluative aspects not related to run-time issues, such as ease of implementation and applicability, are more abstract concepts, and can be assessed more directly through analyzing the system design than through automatic testing. A thorough evaluation of the proposed system and its viability as an MT approach should also include these types of assessments.

Test Suite	Word-Align Data	Mean Source Length	Simple Accuracy	Translation Accuracy	Mean Run Time (secs)
1	CYK	20.3	1.00	1.00	0.90
1	Giza^{++}	20.3	1.00	1.00	1.00

Table 5.1: Summary of feasibility test results (English to Spanish)

As a feasibility test for the basic linked automata architecture, I created a translation system using 1,529 bitexts from English and Spanish versions of the Bible, word-aligned with the CYK word-aligner and Giza^{++} (see Chapter 3). I then collected 10 of these bitexts at random (I call this test-suite 1), to see if the system could correctly process them. This was, of course, not intended to measure the system's translation performance, as I used sentences on which it was trained. Rather, it was a reality check, to make sure that the system could handle the bitexts from which it was constructed. The results of true tests, on unseen data, will be presented in Chapters 6 and 7. As shown in Table 5.1, the system translated all the sentences correctly, and relatively quickly, with mean run time for the CYK-aligned training set well under one second, and for the Giza^{++} set at one second.[7]

Lastly, Appendix A, which begins on page 280, provides some examples of translations produced by various linked automata MT models that were constructed for the tests reported in the dissertation. Translations are shown which yield different translation-accuracies, ranging from very poor (0.13) to perfect (1.00). Although detailed error-analysis is premature for a system at this stage of development, Appendix A should help give an idea of what the accuracy numbers mean, here and in Chapters 6 and 7, as well as give a sense of the type of sentences in the translation domain.

[7]The slight run time disparity between the two is due to more target alignments being produced in the Giza^{++} case; this becomes more pronounced with more difficult translation tasks (individual run times were rounded to nearest second).

CHAPTER 6

EXTENDING THE MODEL, PART I: GENERALIZATION

[S]ince word-for-word translations are surprisingly good, it seems reasonable to accept a word-for-word translation as a first approximation and then see what can be done to improve it. (Yngve 1955:208).

6.1 Introduction

As presented in Chapter 4, the linked automata model can translate only the sentences on which it was trained. In order to generalize to unseen examples, a number of techniques will be considered in this chapter, the most important of which is *merging*. Merging refers to combining transitions in the individual automata, which can also be thought of as combining states (i.e., the states on either end of one transition with the states on either end of another). Merging also necessitates changes to the alignment table, so that the mapping between transition sequences remains accurate. Merging has the twofold benefit of increasing coverage for the model and reducing its size, which is important so that the model can scale reasonably. In this chapter, I first present merging at the level of the automata in section 6.2.1, and then at the level of the entire translation system in section 6.2.2. I also present some of the complications that merging can bring for the model, in section 6.2.4. Next, in section 6.3, I present some other techniques which can also be used to increase the coverage of the model, which deal more with how the model is accessed than changing its structure. These heuristics include processing of fragments (section 6.3.1), dealing with unknown words (section 6.3.2), extracting partial recognition results from the source automaton (i.e., dealing with cases where recognition fails, in section 6.3.3), and extracting partial results when the set of activated target transitions do not yield a target parse (section 6.3.4). I also delve into more of the details as to how these steps are made efficient. In the final section of the chapter (section 6.4), I reevaluate the translation model using different combinations of merging and these other increased-coverage techniques, to see how it generalizes to unseen examples, as well as note any changes in size and efficiency.

6.2 Merging

6.2.1 Merging at the Automaton Level

Merging is the combining of two transitions. For all discussion of merging in this section, I refer to transitions that share the same label. Thus, merging a transition *t1*, labeled x, from state i to state j, with count c_1 ($t1 =< i, j, x, c_1 >$),[1] and a transition *t2*, labeled x, from state k to state l, with count c_2 ($t2 =< k, l, x, c_2 >$), means effectively removing *t2* from the automaton and changing every other transition

[1]Recall, as described in section 4.3.1, transitions have a count, in addition to a probability; in this section I typically show just the counts (probabilities obviously change as well as the result of a merge, but recalculation is usually done once all merging or construction is finished).

that started/ended at k to start/end at i, and every state that started/ended at l to start/end at j. More formally, a merge comprises the following steps (where we also make sure that no start-state or final state information is lost):

(74) To merge $t1 =< i, j, x, c_1 >$ and $t2 =< k, l, x, c_2 >$

(a) if k is the start-state, make i the start-state
(b) if k is a final state, make i a final state
(c) if l is the start-state, make j the start-state
(d) if l is a final state, make j a final state
(e) for all transitions with k as begin state, make i begin state
(f) for all transitions with k as end state, make i end state
(g) for all transitions with l as begin state, make j begin state
(h) for all transitions with l as end state, make j end state
(i) change $t1$ count to $c_1 + c_2$
(j) delete $t2$

Again, one can view merging as the combining of states rather transitions; thus, in (74), state k is merged with state i, and state l is merged with j (so k and l are effectively removed from the automaton). I focus on the transition aspect of merging because it highlights the reason for merging in the first place, as labels of transitions represent the words whose translation behavior we can generalize.

The primary reason for merging, as mentioned earlier, is to increase coverage. A second benefit is that merging decreases the size of the automaton, in that after a merge there are fewer states and transitions. As mentioned earlier, merging (of states) is also the primary means to achieve generalization and size reduction in some other finite-state MT systems, such as subsequential transducer models (see section 2.2.2.3). Merging for the linked automata model is similar, but more complicated because transitions do not exist in isolation, but rather are linked to one another via the table.

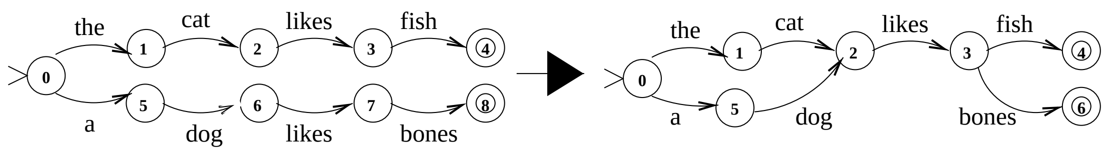


Figure 6.1: Merging on the automaton level

As an example of merging, in Figure 6.1, the automaton on the left recognizes the two sentences *the cat likes fish* and *a dog likes bones*. After merging the two transitions labeled *likes* to get the automaton on the right side of the figure, the automaton can recognize the two original sentences, plus *the cat likes bones* and *a dog likes fish*, even though these two new sentences were not in the training data. This increased coverage means that we now have the possibility of recognizing (and thus at least a chance of translating) two sentences which we could not prior to the merge. Thus, merging as described is not a 'sound' operation, like minimization or determinization, because it intentionally changes (increases) the number of strings that the automaton can recognize. Also note in Figure 6.1 that the merge reduces the number of states from nine to seven and the number of transitions from eight to seven. The decrease in system size resulting from merging does not necessarily mean that there will be in increase in translation speed, however. This is because while there may be fewer transitions, the overall search space can in many instances be much larger, because of the large increase in the number of possible paths.

When we consider merging in the model as a whole, there are two major constraints to keep in mind. First, as mentioned earlier, we desire for our automata to be acyclic (see section 4.2.1). Acyclicity in the automata makes the ordering of activated transitions more clear. Second, we also want to make sure that we preserve what we call the *translation integrity* for an entire translation system. This just means that we want to maintain the translations on which the system was trained. We formalize this notion as follows:

(75) Let $P = \sum_{source}^{*}$ the set of strings over the set of source words,
$Q = \sum_{target}^{*}$ the set of strings over the set of target words,
$T \subseteq P \times Q$ be the set of training bitexts, i.e., the set of ordered pairs containing one source string and one target string, and
T' be the set of ordered pairs which are translations in the merged translation system, then:

$$(\forall p)(\forall q) \ \ (< p, q > \in T \rightarrow < p, q > \in T')$$

I describe how we make sure that merging preserves the translation integrity in section 6.2.2. The basic idea is to not only check that the labels of transitions to be merged match, but also that the labels of the transitions they are linked with also match, i.e., that the words have the same translations.

To ensure that a merge operation will still leave the given automaton acyclic, we need to do a check before the merge: To see if a cycle would be created by the merge, we check if any of the states involved in the proposed merge are already in an ordering relation in the automaton. If so, we do not want to allow the merge. We formalize this notion as follows:

(76) Given an acyclic automaton, A, a $merge(< i, j, x, c_1 >, < k, l, x, c_2 >)$ is acyclic in A if and only if all of the following six conditions hold:[2]

(1) $i \neq l$
(2) $j \neq k$
(3) there is not a path from j to l
(4) there is not a path from l to j
(5) there is not a path from i to k
(6) there is not a path from k to i

The proof that these conditions will prevent the formation of cycles is somewhat tedious, so we do not burden the reader with it here.[3]

[2]It would be much more efficient if we collapsed conditions (3) and (5) to a single condition: there is not a path from i to l. However, while this single condition covers all the cases of (3) and (5), it is stronger, and rules out some non-cyclic cases. For example, it would rule out merging when there was a transition from i to l not through j or k. The analogous argument is true for conditions (4) and (6). Nevertheless, if a speed-up is needed for merging, this might be a good trade-off, since the potential number of merges missed would probably be small, and the time for merging could be almost halved (this is in fact done in the implementation of the system).

[3]One can get a feel for the structure of the proof by imagining two pieces of rope laid out horizontally from left to right. If the ropes are already knotted in one place, and we attempt to add any knots where a piece of rope to the right of the knot is newly tied with a piece of rope to the left of the original knot, then we have created a loop in the ropes (the same is true if a piece of rope originally to the left of the knot is to be tied with a piece of rope to the right of the knot).

We show an algorithm which demonstrates the two checks we require for a merge to be able to take place on the automaton level in Figure 6.2 (this algorithm is employed by the algorithm which does the actual translation system merging, shown in the next section, in Figure 6.5).

```
fsa-merge-okay ($fsa$, $tran_1$, $tran_2$)
{ if [ equal(transition-label($tran_1$), transition-label($tran_2$)) AND
       merge-would-be-acyclic($fsa$, $tran_1$, $tran_2$) ]
  then  TRUE
  else  FALSE   }
```

Figure 6.2: The algorithm to okay automaton transition merges

6.2.2 Merging at the Translation System Level

Having defined merging at the automaton level, I now describe merging at the level of the entire translation system. Again, the goal is to increase the coverage of the system—the number of sentences it can successfully translate. While at the automaton level, we only check for the same labels and for acyclicity to permit a merge, it turns out that merging the automata independently (i.e., without regard to the translation system as a whole) has potentially negative effects on the quality of translation (i.e., it does not in general preserve translation integrity). This result should not be surprising. The translation system's two automata are intimately connected via the table, and changes in one automaton are bound to affect the other.

We illustrate the situation with two very simple examples in Figures 6.3 and 6.4. In both figures, we show the source automaton on the top and the target automaton on the bottom, with the pre-merge version to the left of the large arrow and the post-merge version to the right. In the first example (Figure 6.3) the source sentence AB is translated into the target language sentence $\alpha\beta$ and source DB is also translated as $\alpha\beta$. Now, one possibility is to merge the two source B transitions (we also merge the β transitions in the target, but the results would be the same if we did not, or if the β transitions had already been merged). As can be seen on the right side of the figure, the translation integrity after the merge is preserved. We still would translate both AB and DB as $\alpha\beta$.

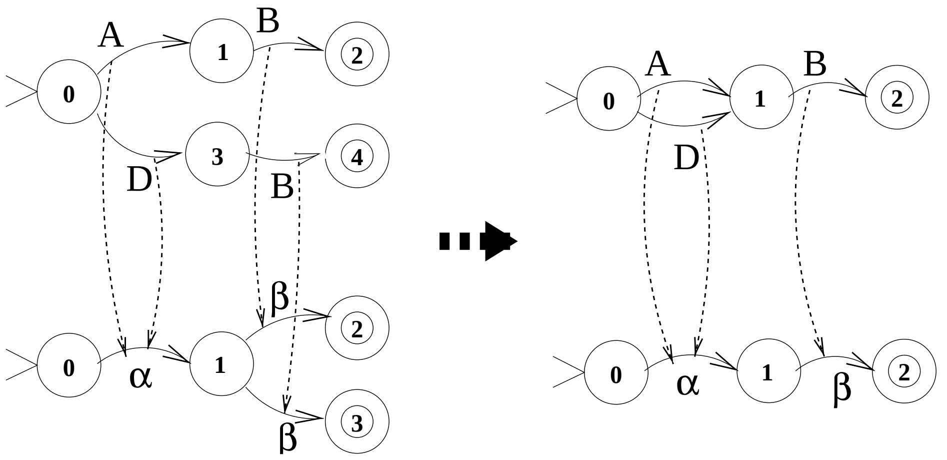


Figure 6.3: Safe merging on the translation system level

Now consider the situation, created from a different hypothetical pair of training sentences, depicted in Figure 6.4. AB translates to $\alpha\beta$, but this time DB translates to $\alpha\delta$. We naively go ahead and merge the two source B transitions. This time, however, (again looking at the right side of the figure) the translation integrity has been lost. An attempted translation of AB now allows both $\alpha\beta$ and $\alpha\delta$ as possible translations, the second of which is wrong. In fact, given the probabilistic nature of the alignments, the most likely scenario is that one translation will always win out: that translation which was spawned from the most frequent training transition sequence alignments, whether a possible translation or not. This is certainly not the behavior we are after.

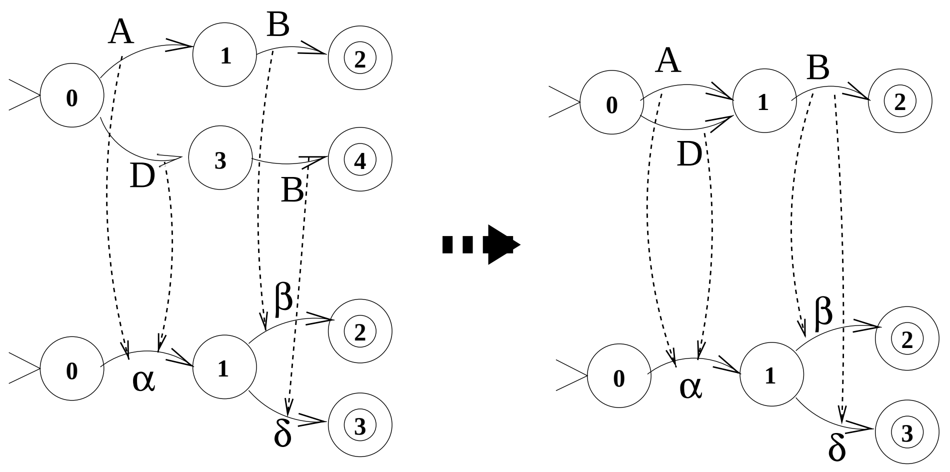


Figure 6.4: Unsafe merging on the translation system level

To prevent such unwanted results, yet promote the increased coverage we seek, we add a constraint on merging. We allow transitions to merge in an automaton only if all the transitions they are associated with in the other automaton can (and will) also be merged. We call this constraint *merge congruity*. This is our most conservative version of the constraint, and the one we stick with for the results reported in this chapter, but more lenient constraints (such as requiring that only some of the associated transitions in the other automaton can be merged) are also worthy of exploration in future research. It is important to remember that these scenarios simplify the picture a bit as well, since the actual alignments are between sequences of transitions, and there can be more than one alignment (i.e., recall that the value in the table for each source transition sequence is a set of pairs of target transition sequences and probabilities).[4] Thus, we now have three constraints on merging (i.e., constraints which will preserve properties which we desire the translation system to have, namely acyclicity and translation integrity): labels must match, merges must not create cycles, and merges must be congruent.

We have not yet discussed how merging affects the table. Since in essence what a merge does is remove a transition from the system, we must go through the table and change all references to the removed transition(s) to the transition(s) it was merged with (we will call this the *remaining transition*). Further, any reference to the states of the removed transition must be changed to the appropriate states of the remaining transition.[5] Additionally, we need to remove the table entry for the alignment between the removed source transition sequence and removed target transition sequence, since the transitions of these sequences no longer exist (thus we

[4]In the implementation I simplify this one step further. I only allow merges where a single source transition was aligned with a single target transition (i.e., I did not merge sequences of transitions with a length greater than 1), so there may be many more possible merges remaining in the system than I later report.

[5]This proceeds automatically in the implementation, since I do not use copies of the transitions in the table, rather I just have pointers to the transitions. So, once I have completed the automaton-specific changes, the states are set correctly in the table.

reduce the size of the table as well when we merge). Finally, as in the automata case, we increment the count (the unnormalized probability) for the remaining alignment by the count for the alignment that was removed.

6.2.3 The Effects of Merging and When to Merge

Having discussed the questions of what merging is, why it is desirable, and how to do it, there remains the question of when to do it. The answer to this question is that it does not matter, but it can be very time consuming if put off to the end. In this initial foray into merging, I began with doing all of the merging after system construction (i.e., after over 1500 pairs of sentences were read in, making over 34,000 source transitions and 32,000 target transitions). Using a rather inefficient algorithm,[6] merging took well over 12 hours to complete. This glaring inefficiency is due to the fact that as the translation system (i.e., number of transitions) grows, so do the number of checks for a merge, and, more importantly, the cycle-check becomes extremely expensive. Thus, merging during construction (so that mergeable transitions are never even made, or made only momentarily) will be much more efficient, since the automata and table are smaller, and we can (possibly) omit the removal steps altogether. As mentioned in section 4.2.1, I also plan to experiment with allowing some cyclicity in the model. This will make the merging process much faster.

[6]I do not give the algorithm here, but basically it amounts to comparing each transition of a given label to every other with the same label, checking to see if the labels of the aligned transitions also match, then checking for acyclicity if both merges were allowed, and finally, if these conditions hold, completing the merge and necessary system adjustments.

I present a very simplified post sentence construction merging algorithm in Figure 6.5. This algorithm is used after each new sentence is processed during system training, thus always keeping the system size to a minimum. The algorithm shown assumes that any new transitions necessary to the system have been constructed, and that only these new transitions need be checked for merging (this is a simplification of what actually takes place). This is not necessarily the fastest algorithm, since the newly constructed transitions may be immediately removed (i.e., merged away), but has the benefit of making the cycle check straightforward.

```
do-construction-merge ($trans\_system$, $src\_tran\_seq$)
{ let $source\_fsa$ = get-source-fsa($trans\_system$)
  let $target\_fsa$ = get-target-fsa($trans\_system$)
  let $ts\_table$    = get-table($trans\_system$)
   ;;; for each transition of newly constructed source transition
   ;;; sequence, get the transition it is aligned with
   foreach ($s_1$ of $src\_tran\_seq$)
   { if new-transition($source\_fsa$, $s_1$) ;; merge only needed if new
         let $t_1$ = table-align($ts\_table$, $s_1$)
           if new-transition($target\_fsa$, $t_1$) ;; merge only needed if new
             let $s_1\_label\_trans$ =
               get-trans-w-same-label($source\_fsa$, transition-label($s_1$))
             ;;; for each transition with the same label as source
             ;;; transition, see if transitions are mergeable, and if
             ;;; aligned transitions are mergeable, if so, do merge
             for each transition $s_2$ of $s_1\_label\_trans$
             { let $t_2$ = table-align($ts\_table$, $s_2$)
                  if [fsa-merge-okay($source\_fsa$, $s_1$, $s_2$)  AND
                      fsa-merge-okay($target\_fsa$, $t_1$, $t_2$)]
                        do-trans-system-merge($trans\_system$, $s_1$, $s_2$, $t_1$, $t_2$)
                        return }}}
```

Figure 6.5: Overview of (simplified) construction merging algorithm

The basic algorithm is to check each transition used for the addition of the source sentence to the source automaton with other source transitions of the same label. If two source transitions can be merged, and the transitions they are aligned with in the table can also be merged (i.e., share the same labels and would not create a cycle), then a merge is completed using the algorithm shown in Figure 6.6. Note that there are several simplifications in the construction merging algorithm (Figure 6.5), shown to make the overall steps more clear, such as the *table-align()* function returning a single transition, rather than a set of transition sequences and probability pairs, and that as shown, there is a bias toward the merging of source transitions (i.e., there may be mergeable target transitions which are not tried), etc. The figure should, however, give an idea of the overall process.

```
do-trans-system-merge ($trans\_system$, $src\_tran_1$, $src\_tran_2$,
                              $trg\_tran_1$, $trg\_tran_2$)
{ do-fsa-merge(get-source-fsa($trans\_system$), $src\_tran_1$, $src\_tran_2$)
  do-fsa-merge(get-target-fsa($trans\_system$), $trg\_tran_1$, $trg\_tran_2$)
  table-merge-adjust(get-table($trans\_system$), $src\_tran_1$, $src\_tran_2$,
                      $trg\_tran_1$,$trg\_tran_2$)    }
```

Figure 6.6: The algorithm to merge in a translation system

Using the basic construction merging algorithm on the same set of training data (just over 1500 bitexts), merging while constructing took approximately one hour. While this is much slower than construction without merging (which takes less than one minute, see section 4.4), it is significantly faster than doing construction, then merging, which, as mentioned, took over 12 hours. Using the absolute most conservative merging strategy (i.e., letting source transitions drive the process, and merging only source transitions which were aligned with singleton target transitions, and also not trying any null alignments), the size of the system was reduced by approximately 31% (measured in terms of source automaton states, which went from 34,537 to 23,833). Given the algorithm's conservativity, there should be significantly more merging possible which still preserves the translation system integrity.

6.2.4 A Complication of Merging

In our discussion so far we considered only the simplest cases of merging, where not a lot of swapping took place in the alignments. But suppose it did. What this means is that we may be left with a series of activated transitions in the target automaton which do not connect. In Figures 6.7 and 6.8, we demonstrate this problem. Figure 6.7 shows the translation system before merging, where $ABCDE$ translates to $\alpha\beta\gamma\delta\varepsilon$, and $FGCHI$ translates to $\phi\chi\gamma\eta\iota$. Now, from the figure we see that the source C transitions can be merged and that the target γ transitions can be merged.

We complete the merge to get the translation system shown in Figure 6.8 (only alignments relevant to problem to be demonstrated are shown). We will still get the correct translations for the two original source strings, $ABCDE$ and $FGCHI$. But what happens, given the alignments in Figure 6.7, if we try to translate the string, newly made recognizable because of the merge, $ABCHI$ (see the thicker transition arrows in the source automaton in Figure 6.8)? The activated transitions in the target (thick arrows) do not connect!

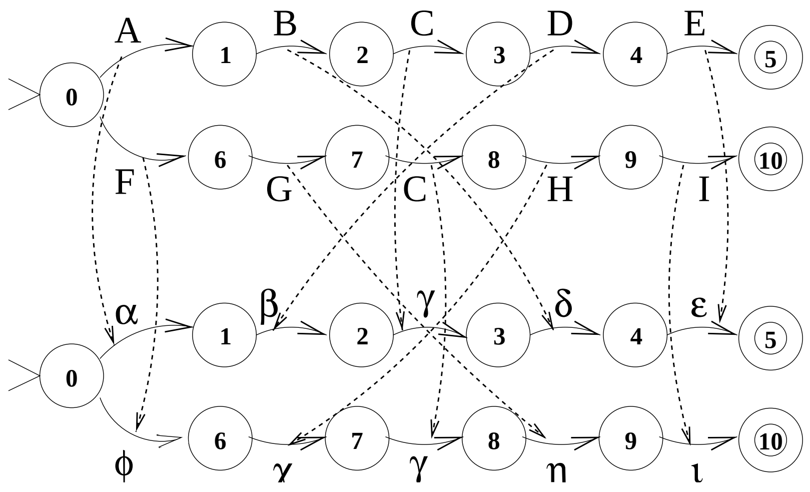


Figure 6.7: An emerging merging problem

Thus, if we allow merging, to have useful results we also require an algorithm for extracting what we might call 'partial' results form the target automaton. Any such algorithm will necessarily be a best-guess algorithm, based on heuristics , since it is not guaranteed that there is one best order. The algorithm I choose for now uses two heuristics, and is called *partial target parsing*. I present it shortly, in section 6.3.4, along with the other increased-coverage heuristics.

6.3 Additional Increased-Coverage Techniques

There are four other methods used in addition to merging to improve generalization in the translation model. They are discussed somewhat briefly here, mainly because I do not see them as important theoretical steps, but rather as straightforward, common-sense heuristics which may in certain cases improve performance but

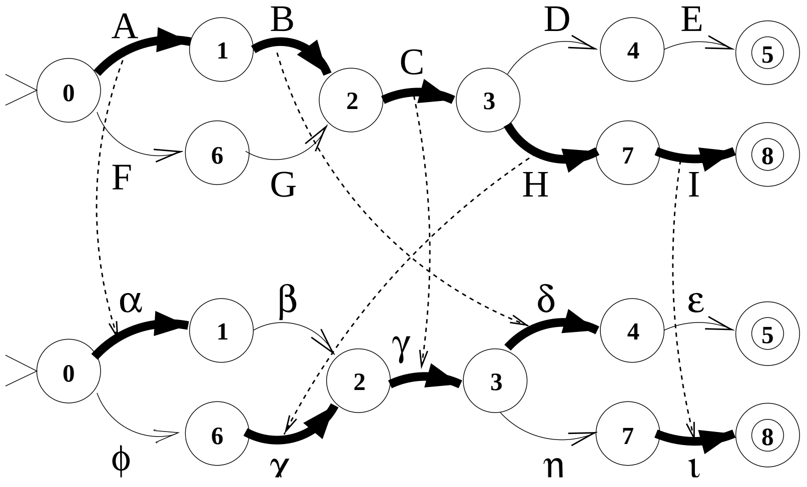


Figure 6.8: Results of merging problem

cannot be claimed to be sound in the general case. The four methods are 1) *fragment processing* (section 6.3.1), 2) *unknown word fall-through* (section 6.3.2), 3) *partial source parsing* (section 6.3.3), and 4) *partial target parsing* (section 6.3.4).

It should be noted that these four techniques are typically used together, and should not be thought of as stand-alone techniques. For example, both partial source parsing and partial target parsing will usually need to invoke fragment processing (the ability to recognize a given substring anywhere in an automaton) to be effective. A typical scenario for translation might be as follows: Given a source sentence, we try to recognize it from the start state to a final state. If successful, we attempt to translate, as described in Chapter 4. The increased-coverage heuristics come into play if source recognition fails. By relaxing the start-state and final state requirements (i.e., fragment processing), we might be able to still recognize the source sentence. If this fails, we then break the sentence into parts which can hopefully be recognized (partial source parsing), which will also involve allowing fragments. These recognized parts must be translated, and any unknown words can be left for special treatment (unknown word fall-through). Translation of such broken-up parts will often produce sequences where the ordering is unclear (as in Figure 6.8) or worse, where the transitions are completely unconnected. Identifying the best transitions to use and putting them in a reasonable target language model order is the job of partial target parsing, which will also involve dealing with fragments.

As mentioned earlier, all four of the increased-coverage methods involve changing the way the model is accessed, rather than changing the structure of the model, as does merging. Thus, unlike merging, none of these four heuristics changes the model's size (since they make no changes to the model). The heuristics begin to give an idea of how the linked automata model can be used as a translation data structure, to enable processing of translations which might otherwise have been deemed beyond the model's reach.

One overarching theme in this section is that the behavior I desire for the model is that it always produce a translation, no matter how poor. I find this useful for two reasons. First, it makes the evaluation process much more clear and useful, since I can gauge improvements more easily by using different quality translations than by only outputting a translation when it is perfect (which may never happen in some cases). Secondly, a translation system which always produces a response is arguably more useful to a user (even in some of the poorest translation cases, a translation still might prove useful if it can give the user an idea of the subject matter, for example, in the translation of a web page). To this end, I have one additional heuristic which I will barely discuss at all, because it is only necessary under certain implementations, and is strictly a practical move. Under some implementations of the system, there may be limits on the length of sentences which can be translated (e.g., the implementation of the SWS as an integer, representing a set, may limit the source sentence size, as discussed in section 4.5.3). In those cases, a quick solution is to break up the sentence into parts, perhaps at an arbitrary breakpoint, and to translate these parts individually. For the results reported in this dissertation, this was not done, because test sentence lengths never exceeded the implementation's cutoff. If such techniques are used, however, they will likely be most successful if they use sensible breakpoints. This notion will be explored further in the discussion of partial source parsing (section 6.3.3), where it is also relevant.

6.3.1 Fragment Processing

Of the four improvements, fragment processing should be the most straightforward. Suppose that the system has been trained on several bitexts, which include a source sentence, S. Next, suppose that we wish to translate a substring of S, and further that this substring was not among the original training sentences (and that it was not covered by merging). The system as described will fail because this shorter string cannot be recognized by the source automaton, since it will not always be true that the path of transitions which would represent it in the source automaton will begin at the start-state and end at a final state. So, although we may potentially have a perfectly reasonable path in the source automaton, we cannot use it. The clear solution here is to relax the recognition requirement, so that any path which covers all the words is a potential candidate source recognition solution, regardless of whether it begins at a start-state or ends at a final state; hence the term *fragment processing*, since these are fragments of the original trained-on sentences. We employ this option only if we cannot recognize the string first from a valid start-state to a valid end-state. We also use the same heuristic in the target automaton. If we have transitions which cover all the source words but none which begin with the target start-state and end in a target final state, we relax this requirement.

Fragment processing is a quite reasonable thing to do, especially when we consider the types of training bitexts we began with: Biblical verses. These verses are often more than one sentence in length. So, the only way we could process the individual sentences of which they are comprised is with fragment processing. Moreover, fragment processing will prove to be the most necessary of all our heuristics, because the other heuristics for extracting partial results rely on the identification of fragments. Fragment processing gives us a method to extract reasonable paths of transitions from the automata, enabling at least the possibility of translating the substrings of the strings on which the system was trained. One instance where fragment processing is likely to be used often, and where its efficiency will be important, is in the recognition of source sentences. Automata, however, are typically constructed with only recognition from the start-state to a final state in mind. Thus recognition of fragments may be impossible, and even with modifications to recognition algorithms, painfully slow. I next describe an implementation of automata which processes fragments efficiently.

6.3.1.1 A Few Implementation Details Concerning Fragments

There are many ways to implement automata. One aspect which all implementations of any size must share is a means to quickly determine which states, if any, can be reached, from a given state, using a given label. That is to say, all automata implementations must model a (partial) transition function (a relation, for nondeterministic automata), t, from states and labels to states. Thus, if Q is the set of states, and W is the set of labels, then we can describe the transition function t as:[7]

(77) $t :< Q, W > \;\; \rightarrow \; Q$

Now the transition function (which is really the heart of an automata implementation) can be implemented as a two dimensional array (sometimes called an *adjacency matrix*), of states and labels, with the value at each index being either a state or empty.[8] This array implementation yields an extremely fast transition function. The problem with this approach is that for most applications, the vast majority of the array cells will be empty. For large automata, the memory requirements make this approach impossible.

[7]The presentation of a transition function here differs from the presentation of automata given in section 4.2.1, where transitions are presented as a set. The two presentations are equivalent, and hopefully not confusing to the reader. In the earlier section, the set presentation makes for easier description of the linking via the table, and of the notion of transition sequences. The functional presentation is used here to more naturally evoke the operations which must take place in an actual traversal of the automata.

[8]In this approach, one would typically make sure the states and labels are ordered in such a way that cell addresses can be calculated in constant time. For states this may mean just ordering things by number, while for labels a unique integer ID for each word may be useful, maintained in a separate table.

Another implementation strategy uses *adjacency lists*. In this approach, one might have an array of states, where associated with each state is a linked list of labels, which each in turn point to a state (or for nondeterministic automata, a set of states). The problem with the adjacency list approach is speed. While the initial array access is fast (again, assuming the top-level array is ordered by state number, with no gaps), searching through the associated linked list is relatively slow, since one may have to (in a worst-case) search the entire list of labels. There are, of course, techniques to speed up this approach. For example, the list of labels need not be a list at all, but something much faster, such as some sort of balanced search tree (e.g., a 2-3 tree or a red-black tree), so that search times are never worse then the height of the tree (i.e., for a tree of N labels, worst-case search time would be $log(N)$).

Unfortunately, for automata used in translation, this is still far too slow. One has to imagine that during translation, once one begins to look for partial results, and therefore no longer limits the search to paths that begin at the start-state, but rather to all possible paths (i.e., a path starting at any state), search must be in constant time. We need a function that when given a begin state and a label, immediately gives an end state (like the two-dimensional array approach) but that does not waste memory. The perhaps obvious answer at this point is to use a hash-table, from states and labels to states. A hash-table has the benefit of very high speed, and only need store entries for the state and label pairs that have values (i.e., for the transitions that exist). In the actual implementation, I do a variant of this, using a two-dimensional hash. That is, there is a top-level hash-table of begin states (i.e., each state used as the beginning of a transition). The begin states are the hash keys. The values associated with each key are also hash-tables, whose keys are labels, whose values are states (i.e., the end states of transitions).[9] This setup is pictured in Figure 6.9 below. This two-dimensional hash is equivalent to having a single hash with states and labels together as the keys (but of course does use more resources), but has the benefit of allowing an application to access all the transitions that begin with a given state instantaneously.

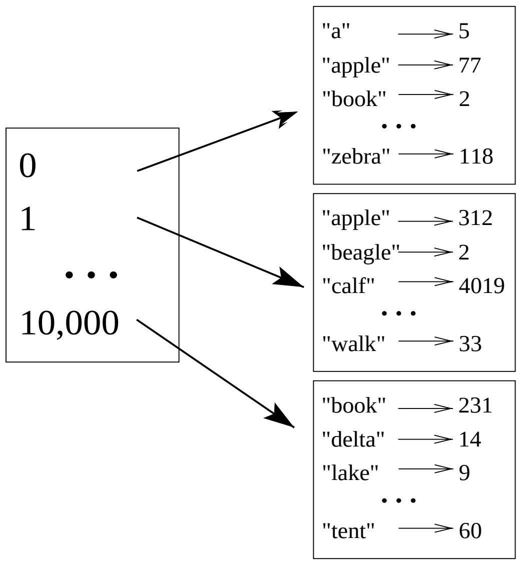


Figure 6.9: A two-dimensional hash as a transition function

Given this overall setup, recognition, the very first task necessary in translation (see section 4.5.1), is very fast. For example, using a very simplified depth-first

[9]This is a bit of a simplification, since transitions in the linked automata model have more than just states and transitions associated with them, but it gives the overall picture of the architecture.

recognition algorithm, such as in Figure 6.10 below, search proceeds quite quickly, since even in very large automata, the combination of states and labels at each step severely limits the numbers of paths tried (i.e., we only continue down a path if $transition\text{-}function(cur\_state, next\_word)$ yields a new state, as shown in Figure 6.10). For example, in a deterministic automaton, there will be at most one such path, so recognition of a string of n words takes just n applications of the transition function; i.e., search time is linear with respect to the length of the input sentence.

```
recognize1 (fsa, wordlist)
{ recognize-next1(fsa, get-start-state(fsa), wordlist, <>) }

recognize-next1 (fsa, cur_state, wordlist_left, transeq_so_far)
{ if ( empty(wordlist_left) AND final-state(fsa, cur_state) )
  { push transeq_so_far onto *successful_recognitions*  }
  else
     let next_word  = remove-first-word(wordlist_left)
     let next_state  = transition-function(cur_state, next_word)
     if  next_state
     { recognize-next1(fsa, next_state, wordlist_left,
               append(transeq_so_far,< cur_state, next_word, next_state >)) }}
```

Figure 6.10: Overview of depth-first recognition algorithm (pruning not shown)

But what happens when we return to the topic at hand, the processing of fragments? A quick fix might be to adapt the algorithm shown in Figure 6.10, so that instead of beginning at the start-state and ending at a final-state, search could begin and end at any of the automaton states, as shown in Figure 6.11. This approach works, but again, for automata of a very large size, it is very slow, because, even with techniques for making sure that the same paths are not repeatedly traversed, the entire search space of the automaton must be searched (assuming one wants to find all the given transition sequences for any fragment, so that the most probable sequence can be found).

```
recognize2 ($fsa$, $wordlist$)
{ foreach $state$ of get-all-states($fsa$)
  { recognize-next2($fsa$, $state$, $wordlist$, <>) }}

recognize-next2 ($fsa$, $cur\_state$, $wordlist\_left$, $transeq\_so\_far$)
{ if empty($wordlist\_left$)
  { push $transeq\_so\_far$ onto $*successful\_recognitions*$  }
  else
     let $next\_word$  = remove-first-word($wordlist\_left$)
     let $next\_state$  = transition-function($cur\_state$, $next\_word$)
     if  $next\_state$
     { recognize-next2($fsa$, $next\_state$, $wordlist\_left$,
                append($transeq\_so\_far$, $< cur\_state, next\_word, next\_state >$)) }}
```

Figure 6.11: An inefficient fragment recognition algorithm (pruning not shown)

A much better solution is to use some extremely valuable information which we already have, to help guide the search. What if instead of beginning with every state, we begin our search with only the states that we know are the begin states of transitions labeled with the first word of the fragment? For example, suppose we are searching for the fragment *black cat.* Instead of looking to each state, to see if it happens to begin a transition labeled with *black*, we could narrow the search space dramatically if we could check only the begin states of the transitions that are actually labeled with *black*. However, given the two-dimensional, begin state to label hash-table architecture, this is cannot be done very efficiently (since one must go through each top level key (each begin state), and check to see if keys (i.e., the labels) in the associated value match the first word of the fragment.

Fortunately, there are several solutions. Perhaps the most elegant solution would be to simply switch the two dimensional array from having begin states at the top-level to having transition labels at the top level. This would be very fast, and use no more resources. The reason why I do not opt for this strategy is that in doing so I would remove the ability previously mentioned, to instantaneously get all the transitions that begin with a given state. This can be a valuable feature for other applications, as well as for other automata operations (such as state merging).

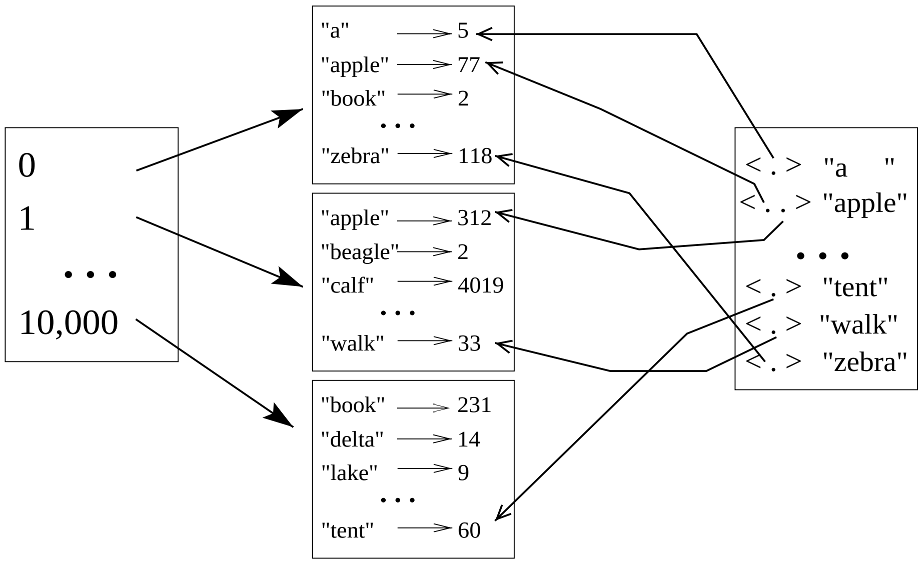


Figure 6.12: Adding a label index for fast fragment recognition

I instead take the pragmatic approach often used with databases. When it is found that certain information is very often used to access the database, then it is reasonable to add an additional index into the data (i.e., the speedup in access usually more than justifies the additional overhead). Thus, I build a second (one-dimensional) hash table, with labels as the keys.[10] The architecture is shown in Figure 6.12, where the index is added to the two-dimensional hash-table pictured in Figure 6.9.

The modified recognition algorithm which makes use of the new indexing scheme is shown in Figure 6.13. Here, the function *recognize3* immediately limits the search space by only using states that begin transitions labeled with the first word of the fragment. Using such an algorithm with the modified architecture, fragment recognition becomes very fast indeed.

```
recognize3 (fsa, wordlist)
{ foreach state of get-states-beginning-with-label(first-word(wordlist))
  { recognize-next2(fsa, state, wordlist, <>) }}
```

Figure 6.13: An efficient fragment recognition algorithm using the label index (pruning not shown; *recognize-next2* is unchanged from Figure 6.11)

To demonstrate this, I built an automaton from 1529 verses of the Bible (in English) and searched for the fragment *the man*. Using an algorithm like that sketched in Figure 6.11, which does not use special indexing for labels, (but does use

[10]These can point to pairs of begin and end states, lists of transitions, or, more likely, lists of pointers to actual transition objects, etc.)

a pruning strategy, to make sure the same search path is never crossed twice), to find every instance of the fragment took 1.7 seconds. Now this may seem relatively quick, especially when one considers that *the* occurs in over 2,000 different transitions in the automaton. But if we employ the label-indexing architecture, and use an algorithm like that shown in Figure 6.13, recognition takes less than .03 seconds! And for less common fragments (i.e., those beginning with less common words, such as *eat*), while recognition in the non-label-indexed case remains relatively constant around 1.7 seconds, in the indexed case (with the fast algorithm) recognition takes place in .001 seconds, or less—several orders of magnitude faster.

This difference can have a tremendous effect on overall translation time, especially when source parsing fails miserably (i.e., in the worst-case, a word-for-word translation, see section 6.3.3), since, for example, this sort of fragment searching must be done for each word. This means the source parsing alone for a 10 word sentence goes from around 17 seconds, to perhaps less than .010 seconds. This divergence only gets worse with the growth of the automata, since the slower approach must always begin with every state. So, the changes are important for scalability of the system too. Thus, hopefully I have demonstrated how efficient fragment processing can be done, and why it is important. All the other increased-coverage heuristics to be discussed next rely on efficient fragment processing.

6.3.2 Unknown Word Fall-Through

Having dispatched with fragment processing and its implementation, we move to another increased-coverage technique which is arguably the most practically-minded of the four. *Unknown word fall-through* deals with the case of a source string containing source language words never before seen in the training sentences. Rather than simply rejecting the string, we use the unknown word(s) as a pivot of sorts. We let the unknown word(s) *fall through*, untranslated, and attempt to translate the words on either side of this pivot, as if these strings were translated on their own. Thus, unknown word fall-through works best when used in conjunction with fragment processing, since the pieces on either side of the pivot may be fragments of sentences which could be translated. Note that this method is not guaranteed to work on the pivot-divided substrings even if they can be recognized, since (as described in the previous merging complications discussion, section 6.2.4) activated target transition sequences may not be contiguous when broken up into parts; and unknown word fall-through necessarily breaks the source string into parts, which can lead to broken target sequences. Once again, partial target parsing may help us extract partial results, if needed (see section 6.3.4). There are several instances where the effects of unknown word fall-through can be seen in Appendix A. For example, in (A.8), the Spanish words *quedarán* and *galaad* fall through to the results, as does *libación* in (A.4).

The prediction with unknown word fall-through is that our translation accuracy will in general be much better than if we had simply rejected the source string, knowing we could not recognize it. Unknown word fall-through enables us to process what we know, even if this means dividing up the input into very small parts. The idea is that when all else fails, working with these parts should prove more accurate than the typical worst-case scenario of not translating at all (recall that there exists a worse possibility, one where the resulting translation actually has a higher edit-distance from the desired translation than would the untranslated source string). Again, given the desire to always produce a translation no matter how low the accuracy, dealing with unknown words is a must, since they are bound to occur in any real-world application. This sort of technique allows us an unbiased method for dealing with these words, which may in addition to newly seen words include misspellings, proper names, and combinations of words which are punctuated in ways which make them unrecognizable to the system.[11]

6.3.3 Partial Source Parsing

The third method is partial source parsing. Suppose that all the individual words in a source string have been seen before, but that we cannot recognize the string itself, i.e., we cannot find any path in the source automaton for the given string, even if we allow fragments. Once again, the perhaps most sensible thing to do is look for 'parts' of this source string which we can recognize, and to translate these parts individually in the linear order in which they occur. The most straightforward greedy algorithm here is to get the longest substring which we can recognize, translate it, and then do the same with the remaining parts. We do a variant of

[11]There are likely more principled and effective ways for dealing with unknown words, which will not be dealt with here. For example, one might be able to relatively accurately predict the lexical category of unknown words, based on the categories of surrounding words and on other clues such as punctuation. These could lead to more accurate translations of surrounding words, when generalizations about categories are incorporated into the system, and might also mean that breakpoints might not be needed, or could be less arbitrary, to keep phrases intact (see the next section on partial source parsing, section 6.3.3, for more discussion of sensible sentence breakpoints).

this,[12] by first finding the largest substring recognition we can that includes the first source string word, translating it, and then doing the same with the remaining words. This approach, at its worst, devolves to a simple word-for-word translation, where the source string imposes its word ordering on the translated target words. As mentioned in section 6.3.1.1, efficient fragment processing is imperative in such instances.

Note that another option is available here. The individual recognitions can be collected, then translated all at once, letting the target language model determine the order, as it is intended to do—but remember, determining the proper ordering may be quite difficult, since the target transitions may be completely unconnected. We experiment with this technique when handling discontinuous alignments, in Chapter 7. A side-effect of such a technique can be significantly faster run times (see section 7.3.4).

Partial source parsing is valuable again because it allows us to work with the parts we know, translating substrings of the strings supplied. Partial source parsing, when used in conjunction with the appropriate methods to extract partial results from the target automaton, will yield translations where they would otherwise be impossible, but may do so at the expense of accuracy (as opposed to more principled generalization techniques, such as merging; i.e., merging is more likely to produce correct translations, but ordering may be an issue; while partial source parsing may

[12]Of course, a better approach would be to check all possible substring combinations for the highest scoring recognition and/or translation, but this can be more expensive computationally. I briefly make use of such a technique to maximize performance when discussing extensions to the model in section 7.3.4.

produce translated parts which are locally correct, but globally incorrect). Nevertheless, we hypothesize that partial source parsing for otherwise unrecognizable strings should allow for marked improvement in translation accuracy (see the results in section 6.4). Partial source parsing may serve the same function in translation as does partial parsing in the world of parsing in general: It allows processing of those phrases or substrings which are easy to handle, with the idea that partial results are more desirable than none.

There are several ways in which one may improve partial parsing performance. One method alluded to previously was to use less arbitrary breakpoints. Instead of matching the longest word sequence possible, partial source parsing is likely to be more effective if segmentation occurs at natural boundaries, such as between phrases, rather than in the middle of them. For example, given a sentence with an embedded clause, a sensible breakpoint might be at the beginning of the clause, rather than, say, in between an adjective and the noun that it modifies. To do so, one might do a full syntactic parse of the sentence, but such means are not always possible and (at least at this point) are beyond the capabilities of the basic system described (i.e., if one could obtain an effective parser for any language, one might envision a different translation model).

An alternative to carrying out a full parse is to attempt to spot certain words which are likely to indicate phrasal boundaries. As mentioned in the survey of Example-Based Machine Translation, in section 2.3, Veale & Way (1997) propose a technique which involves using a closed-class of function words, called *markers* (which might be hand coded, or automatically induced via word-frequencies), to delineate phrasal boundaries. They justify their approach based on psycholinguistic studies of human sentence processing.

Another approach to this segmentation problem comes from the world of statistical machine translation, where the idea of a *rift* is introduced (Berger *et al.* 1996). A rift is a position, j, in the target sentence of an aligned bitext such that all target words to the left of j are aligned with (i.e. generated by)[13] source words to the left of the source word, e_{a_j} (i.e., the source word that target word f_j is aligned with via the alignment a_j), and all target words to the right of j are aligned with source words to the right of source word e_{a_j}. An example of a word-aligned bitext indicating safe segmentation points (i.e., rifts) and unsafe ones is shown in Figure 6.14. As Berger *et al.* (1996:61) point out, using such segmentations will not always "result in semantically coherent segments." For example, in Figure 6.14, *le* and *chat* are part of one noun phrase, yet could be separated by the rifts indicated.[14]

Berger *et al.* (1996) create a segmentation model, based on this idea, which can be used during translation, to (ideally) make reasonable breaks of the input sentence which will not cause problems later in the translation process. They train the model based on a set of word-aligned bitexts (i.e., they use their main translation model to align the bitexts), and use several features to determine whether a position in the target should be marked as *rift* or *no-rift* (the features include part-of-speech

[13]This follows the same terminology typically used in statistical machine translation, where in a translation task from f to e, e is called the *source*, which is responsible for generation the *target*, f; see section 2.2.1).

[14]This issue will arise again in the discussion of the handling of discontinuous word alignments, in Chapter 7.

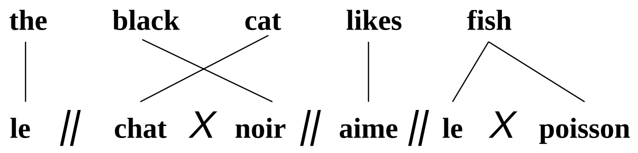


Figure 6.14: A word-aligned English and French bitext, showing safe segmentation points (known as *rifts*, marked with ||), and unsafe segmentation points (marked with X).

tags, word class information, and a number of others obtained within a six-word window of context around the position in question). They then use a dynamic programming algorithm to determine an optimal split point for each training source sentence. Although Berger *et al.* (1996) do not provide a quantitative evaluation of their segmentation model, the idea appears promising, and could be relevant for future partial source parsing research with the linked automata model.

One final interesting partial source parsing idea, previously mentioned in section 2.2.2.6, departs from the goal of finding suitable segmentation points, and attempts to instead match (otherwise unrecognizable) source sentences by allowing mismatches of certain words (Vogel & Ney 2000). This is accomplished by using a weighted edit-distance during recognition, where the recognition process can continue so long as the edit-distance so far accumulated in the process is less then some threshold. For example, suppose we can recognize the sentence *the dog went in the door* but not *the dog went to the door*. If function words like the prepositions *in* and *to* are weighted sufficiently low, such a recognition might be permitted (i.e., using the first sentence as an approximate recognition of the second), allowing for perhaps a better translation then if segmentation of the source sentence was tried. Such a technique puts more of the burden on the target language model, which can be viewed as having an opportunity to fix the source model's approximations (e.g., two function words in the source language might have the same target translation anyway).

Partial (source) parsing, is, of course, a potentially vast research area in its own right. In this dissertation, we have just scratched the surface in terms of its possibilities. Clearly, the performance of the linked automata model could be enhanced by improving the partial source parsing, and many of the ideas identified in this section could be of some use. Perhaps the most interesting might be full-blown regular expression matching for the automata (i.e., allowing for matches such as *word X followed by some number of words followed by phrase Y*), which might prove especially useful when discontinuous alignments are used (see Chapter 7). We next turn to the last of the increased-coverage techniques, one which also concerns extracting partial results, but this time from the target automaton.

6.3.4 Partial Target Parsing

Partial target parsing is used at the last stage of the translation process, to extract information from the target automaton, when neither a complete parse (i.e., a transition which begins at the start-state, ends at a final state, and has a full SWS; see section 4.5.3) nor a complete fragment (a transition with a full SWS) can be found. Partial target parsing is just like normal target parsing, in that it begins with activated target transitions, generates longer transitions from the small automaton created, and identifies which of these transitions to use in the resulting translation. However, partial target parsing differs in one crucial respect:

The resulting transitions will likely need to be ordered, since they probably do not connect (if they did connect, they would have been found via normal target parsing that allows for fragments, except in cases where the SWS was not full).[15]

The situation of having unconnected transitions can arise as the result of merging and the other generalization techniques, as was demonstrated in section 6.2.4 (see Figure 6.8). The algorithm used to order unconnected transitions employs two heuristics. First, given two activated transitions A and B, if A precedes B in the automaton, then A precedes B in the resulting translation. We use the following second heuristic only if the first heuristic does not hold in either direction: If the distance from the start-state to A is less than the distance to B, then A precedes B in the translation. The distance of the various transitions from the start-state are pre-compiled (i.e., done at training time), as a time-saving step, but the precedence heuristic must be computed at run time. Note that both heuristics are computed relative to the entire target automaton, and not the smaller one that is defined by the activated target transitions (since precedence relations might not show in the smaller automaton, and distance calculations are both quite different, and potentially impossible to calculate in the smaller automaton). Using these techniques, we can guarantee that the system always produces something; again, this a property that simplifies the task of evaluation.

There may, of course, be better techniques to order the transitions using just the basic architecture of the model (i.e., without moving to techniques which make more use of linguistic information, such as POS tags, or different statistical information,

[15]Recall that, as described in section 4.5.3, the ordering in normal target parsing comes for free, because the sequences are all connected as a single path in the target automaton.

such as separate language models), and although the precedence techniques make a great deal of intuitive sense, I do not see the distance heuristic as being very helpful or well-motivated, especially given how much transitions may move around after merging (but remember, merging will not affect the order of transitions already in a precedence relation). One such technique which makes a great deal of sense, especially when language pairs exhibit some degree of ordering similarity, is to let the source sentence suggest an order for the activated target transitions. This third heuristic can be very valuable after using partial parsing techniques, especially those which collect recognized source transition sequences for translation as a unit (see section 6.3.3). This suggested-ordering heuristic is used for some of the results reported in section 7.3.4. I leave the search for other such techniques to future research.

Although the main processes of partial target parsing are similar (with the exception of ordering) in terms of their implementation to those of normal target parsing, we should spend a moment highlighting some of the differences in terms of both generation of transitions and selection of which of these transitions to use.

Of the two, the generation process is the most straightforward. Recall that in the third translation stage, we generate and store all the transitions that can be built by putting them together (i.e., *smushing* them), so long as their SWSs do not intersect. This generation process, which uses the smaller automaton (i.e., only the activated target transitions), begins at the start-state. Thus after normal target parsing has failed, every transition in the chart which stores these smushed transitions has a begin state which is the start-state. To obtain all other possible smushed transition sequences, we simply redo the generation process, but this time using all of the smaller automaton begin states (less the start-state) as potential starting points. There are clearly more efficient ways to do this step, since in doing so we are likely crossing some paths already crossed in the first generation stage, but given the automaton's small size, the time used is unproblematic.[16]

After generation, the selection of which transitions to use is somewhat trickier. The process uses the same data structures that were discussed regarding the processing of empty transitions, in section 4.5.3.1, namely the empty transition mask, and the empty transition probability hash, as well as the store of newly built transitions made from generation.

The technique used is based on one important theoretical assumption. We are likely to get better translations by using connected target transitions which cover a lot of source words, than by using unconnected target transitions which cover the same amount of source words (or more) but are less connected. This is consistent with our notion of assuming word-for-word translations will typically not be as good as translations made from units longer than a word, and makes better use of the target language model. This assumption will come into play should a situation arise like that depicted in (78) below.

(78) a. $\{< i, k, .7, [01110] > \quad < a, d, .8, [10001] >\}$ combined probability $= .56$

b. $\{< a, b, .9, [10000] > \quad < w, x, .9, [00010] > \quad < l, q, .9, [00001] >$
$< t, q, .9, [00100] > \quad < a, c, .9, [01000] >\}$ combined probability $= .59$

[16]As mentioned before, should the small automaton's size become large enough to make this technique problematic, one could limit the number of activated transitions produced in the second translation stage (see section 4.5.2). Should this situation arise, however, more efficient generation strategies will first be developed, since they retain the possibility of guaranteeing that the most probable transition sequence could be found, while the technique of restricting the number of activated transitions does not.

Suppose we began translation with a five-word source sentence, and are trying to choose between the two partial target parses (i.e., sets of transitions, where only states, SWSs, and probabilities are shown), (78a) and (78b). The parse with five transitions, each representing one source word, in (78b), has a higher combined probability at .59, than does the alternative at .56 in (78a), which uses only two transitions to cover the same source words. Nevertheless, we will select (78a), because it will likely make a more fluent target language sentence than will the transitions of (78b).

With this assumption explained, the algorithm to select which transitions to use is relatively simple, and is depicted in Figure 6.15 and Figure 6.16. The basic idea begins with the store of the generated target transitions. Here we assume that they are indexed by the number of source words they cover. The top-level function, *get-best-transition-set* (Figure 6.15), extracts the best available transition which covers the most source words not yet covered, adds this transition to its collection, and continues, until it cannot find another transition or the combined SWS of the collected set is full.

The function which *get-best-transition-set* calls to get the best transition, *extract-best-tran* (shown in Figure 6.16), uses the indexed store to quickly get the best transition that covers the most source words, and also checks to see if this transition's combined probability could be beaten by a collection of empty transitions which cover the same number of source words (where all the source words are needed, but are not necessarily the same source words that the best transition covers). In this manner, the empty transitions are pulled into the process. Note that this does

```
get-best-transition-set (transeqs_by_numsources, full_sws)
{ sws_size      = 1 - numwords(full_sws)
  cur_sws       = make-empty-sws(full_sws)
  best_tran     = extract-best-tran(sws_size, cur_sws, transeqs_by_numsources)
  best_tran_set = push best_tran onto make-set()

  while(best_tran AND not(full-sws(cur_sws)))
  {  cur_sws  = union_sws(cur_sws, tran-sws(best_tran))
     best_tran = extract-best-tran(num-sws-needed(cur_sws), cur_sws,
                                    transeqs_by_numsources)
     push best_tran onto best_tran_set }
  return best_tran_set  }
```

Figure 6.15: Overview of algorithm for selecting best set of target transitions in partial target parsing

not deviate from the earlier mentioned desire to bias the collection to connected transitions, since the empty transitions can be viewed as being able to connect to anything.

Once this the selection of the best transition set is complete, the target parsing process finishes with ordering the set of transitions, using the two heuristics described earlier (i.e., precedence and, if that fails, distance from the start-state). As mentioned, a third heuristic, using a suggested order from the source recognition process, can also be used if the precedence heuristic fails. In these cases, the distance heuristic is used only as a last resort. For all of the results in the dissertation, except for where indicated in section 7.3.4, this suggested-ordering heuristic was not employed, as it was developed after the tests were completed. Note that we

```
extract-best-tran (size_to_check, cur_sws, tranhash_by_num_sources)
{ best_tran = null
  while((size_to_check > 0) AND null(best_tran))
  { best_tran = get-best-of-sws-size(size_to_check, cur_sws,
                                     tranhash_by_num_sources)
     if not(best_tran) { size_to_check = 1 − size_to_check } }
  if exist_better_empty_tran_collection(best_tran, cur_sws)
  { best_tran = get_empty_tran_collection(best_tran, cur_sws)}
  return best_tran }
```

Figure 6.16: Overview of algorithm for selecting best transition in partial target parsing

must use some heuristic to order the set, because they cannot already be connected, otherwise we would have selected the combined longer transition in the first place, i.e., partial target parsing would not have been necessary.

So ends the presentation of the heuristics for increasing coverage in the linked automata model. In the next section, I evaluate the impact of all of the generalization techniques, including these heuristics and a very conservative implementation of merging.

6.4 Evaluation After Generalization

As with the earliest evaluation (the feasibility evaluation, from Chapter 5), I began the evaluation of generalization by first making sure that I did not break anything that was working before. To that end, I repeated the same feasibility tests performed in section 5.4, shown in Table 5.1, but this time used translation systems that had been merged during construction (and which were again trained on 1529 bitexts). The size of the translation systems were reduced by over 30%, as was described in section 6.2.3.[17]

I present the test results in Table 6.1. As in the earlier test (Table 5.1), I used test-suite 1, which consists of 10 training sentences. The goal here is to make sure that merging did indeed preserve the translation integrity, i.e., to make sure that all translations which could be accomplished prior to merging were still available. Again, this being a test on training data, I view it as more of a reality check; tests on unseen data are presented next.

Test Suite	Word-Align Data	Mean Source Length	Simple Accuracy	Translation Accuracy	Mean Run Time (secs)
1	CYK	20.3	.99	.99	0.30
1	Giza^{++}	20.3	.99	.99	1.80

Table 6.1: Summary of feasibility test results (English to Spanish), merged systems

The very slight decrease in accuracy, from 1.00 to .99 for both the CYK and Giza^{++} word-aligned training sets, is unproblematic. Manual inspection of the

[17]Size reduction is an important goal for the model, so that it scales well, and while I will not further evaluate the model in terms of its size, it will remain an important area for future research in tandem with the prototypical generalization goal of increasing coverage.

translations produced showed that the one-word deviation from the reference translation which resulted in the accuracy reduction was a perfectly acceptable alternative translation that was made available to the system via merging.[18] As was also the case in the earlier results, the times for the CYK trained data are significantly faster than for the Giza^{++} data. As stated earlier, this is because the number of transitions produced (i.e., activated) in the Giza^{++} is typically much larger (usually by a factor of 4 to 5). We will discuss this more as additional results are presented. Readers may also note that the run time for the CYK data using the merged system is faster than the CYK times for the unmerged system (see Table 5.1). This should not be weighed too heavily, because the tests were run in somewhat different environments, and times were also subject to rounding errors.[19] Nevertheless, important deviations in mean run times are quite apparent (i.e., show larger differences) in some of the additional results to be presented shortly, where they will be further discussed.

In Table 6.2 we present the results for test-suite 2, in a slightly different format from the two earlier result tables. First, we add the information for each generalization and increased-coverage technique, to indicate what was used on each test. Thus, we indicate when a merged system was used, and when partial source parsing,

[18]More specifically, there are many English sentences in the Bible beginning with the word *And*, sometimes appearing in Spanish with the word *Entonces*, other times with *Cuando*, and other times with no word at all, etc. The one word deviations were of this type; ones which we would actually hope the model would make (i.e., without other information, to choose the most probable alternative).

[19]I tried as best I could to measure actual CPU seconds, rather than real time, to minimize the effects of other processes on the same machine, but result times still varied in rather similar contexts, suggesting that the measuring mechanisms were not as independent of other processes as one might hope.

fragment processing, partial target parsing, and unknown word fall-through were invoked. The tests are presented in the fashion that those with fewer techniques used are shown first, and those employing additional techniques shown later. Thus, the expectation is that results further down in the table should be better. Also, to save space, we present the simple accuracy and translation accuracy results for each training set in a single column, divided by a '/'. Lastly, because there are more tests being presented than in the earlier tables, and to make for easy cross training-data comparison, we present the results for the CYK and Giza^{++} sets on the same line, for each different test configuration.

Test-suite 2 consists of ten randomly chosen (English and Spanish) bitexts from the Bible, which were not in the training data. Additionally, the bitexts were selected to ensure that they only used source words which had appeared at least once in the training data. Thus, this might be viewed as the first real test of the system. As will be shown in this test, and the succeeding one, accuracy results are relatively low (never higher than .40). This should not be surprising, given that the translation domain is a particularly difficult one, the amount (and accuracy) of training data was limited, and that the model is still in a very early stage of development, and may turn out to be unsuitable for such unrestricted translation tasks. Accuracy and future prospects for its improvement will be discussed further in the final chapter of the dissertation (Chapter 7), and some examples of a range of translations with different translation-accuracies are shown in Appendix A. The area to focus on here is whether the changes in the use of different techniques result in improvements in accuracy, so that we can gauge the extent to which they have improved coverage in the model, and assess their viability as future foci for improvement.

TS	SL	Mrg	PSP	FRG	PTP	UFT	CYK-wa SA/TA	Giza-wa SA/TA	CYK Time	Giza Time
2	19.5	no	no	no	no	no	0.00/0.00	0.00/0.00	0.0	0.0
2	19.5	yes	no	no	no	no	0.00/0.00	0.00/0.00	0.0	0.0
2	19.5	no	yes	yes	no	no	0.16/0.16	0.39/0.39	4.1	15.2
2	19.5	yes	yes	yes	no	no	0.13/0.13	0.38/0.39	4.0	14.4
2	19.5	no	yes	yes	yes	no	0.30/0.30	0.40/0.40	4.2	15.2
2	19.5	yes	yes	yes	yes	no	0.28/0.28	0.39/0.39	4.1	14.4

Table 6.2: Summary of test suite 2 results *key: TS=test-suite, SL=mean source sentence length, Mrg=Merged, PSP=partial source parsing, FRG= fragment processing, PTP=partial target parsing, UFT=unknown word fall-through, CYK-wa=CYK word alignment, Giza-wa=Giza^{++} word alignment, SA/TA=simple-accuracy/translation-accuracy, Time=mean run time (seconds)*

In terms of the increased-coverage techniques, Table 6.2 shows a marked improvement (for each training set) as the various techniques are added, showing the necessity of being able to extract partial results. Thus, translation of unseen sentences is impossible without, for example, invoking partial source parsing, since the source sentences are unlikely to be recognized (thus we see accuracies at 0.0 in the first two result lines of the table). One important exception to the improvements is merging. A merged configuration of each test is shown on the line following the same unmerged configuration. As can be seen, in many cases merging actually slightly decreases the accuracy. This, together with 0.0 accuracy results for the merging without additional increased-coverage heuristics case (the second result line of the table), suggest that the merging as currently being done is not sufficient to provide increased coverage.[20] This means that more merging (i.e., less conservative merging) will have to be done, before it will bear fruit as a generalization technique. Merging remains the most important technique (i.e., the one with the most potential), since it achieves generalization in a principled way (allowing sentences to be processed in full, rather than broken into parts), and, as stated earlier in this chapter (see section 6.2), less conservative merging is one of the most important and most promising areas for future research in the model.

Another aspect of Table 6.2 to be discussed is the mean run times. Times in general increase, as expected, over the run times reported earlier, since much more work is being done in terms of processing partial results. Worst-case run times for the CYK trained data are around 4 seconds for translation of source sentences averaging 19.5 words in length. Worst-case run times for the Giza^{++} data are much worse, at just over 15 seconds. This disparity is again due the greater number of transitions being activated in the Giza^{++} case. In fact, the factor by which the time increases is roughly equal to the factor by which the number of activated transitions increases. Part of the reason for the greater number of transitions in the Giza^{++} case is because all of the 0:1 alignments are activated with each translation task, and the Giza^{++} data has more 0:1 alignments. When we motivated this step in section 4.5.2, we hypothesized that this number would not be problematic. In

[20]Manual inspection of the results supports this conclusion; recognition of the full source sentence failed in all cases.

retrospect, however, it seems that a principled approach to limiting the number of 0:1 transitions used, such as those described in section 4.5.2, could be helpful, and would make the system scale much better, since as currently defined, the number of activated transitions will increase with the size of the system, in terms of the 0:1 alignments.

Although in the dissertation I will not experiment with any of these improvements, I do identify several possible steps that might be taken. Such solutions should be relatively easy to find, and range from what might be viewed as hacks, to restructuring of the system. On the hack side of things, one might simply restrict the number of 0:1 alignments used in a translation (i.e., the beam search idea from section 4.5.2), or could not use them at all in partial target parsing, since they are unlikely to be used anyway given the present search algorithms (given that they offer no new SWS information). More principled solutions include that given in section 4.5.2, of adding extra transitions to the automata so that 0:1 alignments could be registered for each transition sequence (thus we would only activate the relevant ones, a very small number). Another solution could be having a second, small alignment table, which only dealt with the null alignments (i.e., the 0:1 and 1:0 cases), which could for each non-null transition sequence of a source sentence, also align it with target transitions involved in any 0:1 alignments for the given bitext. In this manner, when a fragment of the sequence was used for recognition, the appropriate 0:1 alignments would be activated. This solution is somewhat like the solution to fragment recognition of section 6.3.1.1 (i.e., adding an additional index), in that we would trade some additional memory used for much more efficient processing.

Almost all the time increase comes in the third and final translation stage, which includes identifying and selecting and ordering (if necessary) the activated target transitions. I take this is a good sign, first, that the time increase is linear with the number of activated transitions and not worse. Secondly, the time increase (and overall long times—I would like to see run times faster than 4 seconds for the CYK data) is due to the one translation stage which has not been optimized at all. For example, the generation step of the process uses no pruning, and techniques to greatly improve the efficiency should be able to be developed. In fact, it may turn out to be the case that generation is not the most efficient technique at all. This is one area of the project where future research should have an immediate and profound impact (although of course only on efficiency, and therefore scalability, but not on the accuracy of the model).

One other good sign is that the times for using the merged systems are roughly the same as in the unmerged systems, even though the search spaces are much greater in the merged case. This indicates that the first stage of the translation process, recognition using the source automata, is operating at a very efficient level, since the increase in search space has seemingly no effect (thus demonstrating the importance of the extra indexing mechanism for labels, discussed in section 6.3.1.1).

A final time efficiency point to note is that, as touched on in section 6.3.3, even before improvements in the third translation stage's efficiency are made, the use of different partial source parsing techniques (such as those which collect recognitions and translate them as a unit) will likely vastly improve run times. This is because for each translation, the slowest translation stage will be invoked once, rather than one time for each recognized source substring.

The results for the third test set, test-suite 3, are shown in Table 6.3. Test-suite 3 consists of 10 randomly chosen bitexts from the Bible which were not included in the training data. In addition, the bitexts were selected so that the source sentences contained some unknown words (i.e., words not which did not appear in the training data). Thus, test-suite 3 is the most difficult of the test sets, where perfect scores would likely be impossible for any MT system. I included this test to get an idea of how well the unknown word fall-through technique was working, and would expect the accuracy numbers to be lower than they were for the previous test (as they are), where all words were known.

TS	SL	Mrg	PSP	FRG	PTP	UFT	CYK-wa SA/TA	Giza-wa SA/TA	CYK Time	Giza Time
3	24.3	no	no	yes	no	yes	0.02/0.02	0.02/0.02	0.2	0.9
3	24.3	yes	no	yes	no	yes	0.02/0.02	0.02/0.02	0.1	0.8
3	24.3	no	yes	yes	no	yes	0.16/0.17	0.33/0.33	4.8	20.4
3	24.3	yes	yes	yes	no	yes	0.21/0.21	0.36/0.36	4.7	18.4
3	24.3	no	yes	yes	yes	yes	0.22/0.23	0.34/0.34	5.1	20.4
3	24.3	yes	yes	yes	yes	yes	0.31/0.23	0.38/0.38	4.8	18.4

Table 6.3: Summary of test suite 3 results *key: TS=test-suite, SL=mean source sentence length, Mrg=Merged, PSP=partial source parsing, FRG= fragment processing, PTP=partial target parsing, UFT=unknown word fall-through, CYK-wa=CYK word alignment, Giza-wa=Giza^{++} word alignment, SA/TA=simple-accuracy/translation-accuracy, Time=mean run time (seconds)*

As can be seen in Table 6.3, the test-suite 3 results roughly pattern after those for test-suite 2 (Table 6.2). The increased-coverage techniques significantly improve the overall accuracies. Similarly, the times required for translation show the same patterns (i.e., Giza^{++} data taking more time than CYK data), with the overall run times being higher, given that the sentences are more difficult, as well as longer (averaging 24.3 words per source sentence). What is perhaps most interesting in Table 6.3 is that using the merged translation systems consistently improved accuracy. Thus merging showed the greatest effect on the most difficult of the tests, i.e., where additional generalization would be expected to be the most helpful.

The final test, using test-suite 4, consisted of sentences not found in the corpus, but rather fragments of sentences from the training bitexts, with one or two words replaced (all the words had been encountered in the training data, and the categories of words replaced varied, including nouns, verbs, determiners, etc.). As such, this test suite should be the easiest of the latter 3 tests (i.e., test suites 2–4). The translations for these slightly modified source sentences were also adjusted to reflect the changes (using what were taken to be the most likely translations after having inspected the training sentences). An example of such a sentence might be instead of using the sentence:[21]

(79) In the beginning God created the heaven and the earth

this test suite might contain something like:

(80) In the beginning God created the day and the light

The goal for this fourth test was to see how the linked automata system would fare when presented with an easier task: shorter sentences that were closer to the original training data. I present the results for test-suite 4 in Table 6.4.

[21] As stated elsewhere in the dissertation, I apologize if my use and manipulation of such Biblical sentences offends anyone, but it was a simple way to create another test set for the system.

TS	SL	Mrg	PSP	FRG	PTP	UFT	CYK-wa SA/TA	Giza-wa SA/TA	CYK Time	Giza Time
4	17.6	no	no	no	no	no	0.00/0.00	0.00/0.00	0.0	0.0
4	17.6	yes	no	no	no	no	0.00/0.00	0.00/0.00	0.0	0.0
4	17.6	no	yes	yes	no	no	0.66/0.66	0.51/0.53	1.2	3.9
4	17.6	yes	yes	yes	no	no	0.62/0.62	0.51/0.51	1.0	3.5
4	17.6	no	yes	yes	yes	no	0.88/0.88	0.80/0.82	1.2	3.9
4	17.6	yes	yes	yes	yes	no	0.84/0.85	0.80/0.81	1.0	3.5

Table 6.4: Summary of test suite 4 results *key: TS=test-suite, SL=mean source sentence length, Mrg=Merged, PSP=partial source parsing, FRG= fragment processing, PTP=partial target parsing, UFT=unknown word fall-through, CYK-wa=CYK word alignment, Giza-wa=Giza^{++} word alignment, SA/TA=simple-accuracy/translation-accuracy, Time=mean run time (seconds)*

The results in Table 6.4 appear somewhat promising. Although merging did not improve accuracy, the overall high relative accuracy scores suggest that the increased-coverage heuristics are on the right track. Replacing trained-on words at certain locations with different trained-on words does not cause the system to completely fail, when the heuristics are used, and in fact, accuracy scores range as high as .88 with an unmerged system. Interestingly, here the poor word-aligned data leads to better results than the better word-aligned data. I have no explanation for this result; using more test sentences might give a better indication if this was just coincidence or not. Also, the overall run times are much faster than they are for the more difficult tests (see Tables 6.2 and 6.3). This should be expected, since easier sentences should mean less work needs to be done. It also bodes well for the future, in the sense that higher accuracies and faster translations will typically go hand in hand (i.e., as we improve accuracy, we can typically also expect to improve the run times).

In summary, the results of the tests post generalization suggest that increased-coverage techniques are both necessary and helpful for the system to handle sentences on which it was not trained. Clearly more generalization will need to be achieved in future research of the model for more reasonable accuracies to be achieved. It may also be the case that unrestricted translation, especially that as difficult as the Bible—where translations are sometimes more distant from one another than one might expect, and where the language can be at times rather figurative—may not be the appropriate domain for the linked automata model. I hope that, through these tests, I have been able to begin to evoke some of the possibilities of the model as not only a translation tool, but as a data structure from which translations can be accessed. In the next chapter, I attempt to further expand the model, and suggest how far it may be taken as a translation system.

CHAPTER 7

EXTENDING THE MODEL, PART II: USING MORE LINGUISTIC INFORMATION AND MORE COMPLEX ALIGNMENTS

A number of classical mathematical problems have been rigorously proved to be unsolvable, as a result which no mathematician, worthy of such a title, would ever again try, for example, to trisect an angle by means of a straightedge and compasses. There are, however, scores of ways in which an angle can be approximately trisected to a high degree of accuracy. The art of translation is not as fortunate. Not only can one adduce rigorous proof that a perfect version of the import, expressed in a source language, can not be achieved in any target language, but one cannot even guarantee an acceptable approximation to the solution of the formidable problem of translation. (Rhodes 1967:431).

In Chapters 4 and 6, I introduced the linked automata MT model in its most basic form, and gave the primary techniques for generalization. As mentioned at several points in the dissertation, I view the linked automata model as a foundation upon which more ambitious MT systems can be built. In this chapter, I present experiments with two different types of possible extensions to the model: using more linguistic information by means of part-of-speech (POS) tags (section 7.2), and increasing the coverage of the model by allowing for discontinuous alignments (section 7.3). Before presenting the results of these two experiments, however, I first attempt to answer another question for the model, by presenting a brief test to examine how training set size affects translation-accuracy, in section 7.1. In the final section of the chapter (section 7.4), I close the dissertation with some comments on where the model stands and where it may be taken in the future.

7.1 On the Effect of Training Set Size

The question of the effect of training set size on translation performance is an important one; in data-driven approaches one needs to have an idea of what amount of training data is necessary to yield a given performance. More specifically, one hopes to learn (for a given translation domain) how many bitexts are necessary for the MT system to begin to be able to translate at all (i.e., until at least a minimal amount of data is processed, no translations will be possible, since most words will be unknown); and how fast performance improves relative to increases in training set size.

In an attempt to shed some preliminary light on these questions, I tested linked automata MT systems of various sizes, using the best performing word alignments identified earlier (namely, Giza^{++}, see Chapters 3 and 6). While in all of the earlier tests I used approximately 1,500 bitexts (i.e., the Biblical book Genesis), in this test I used training sets ranging from 0 to 3,000 bitexts, at intervals of 250 bitexts. I used all of the increased-coverage techniques in the tests, including the very conservative merging described in Chapter 6.

Although the topmost size of this range is still small by data-driven MT standards, the span of training set sizes is enough to give an initial view of the effect of training size on the model. Given that the model is still in an early stage of development, and therefore not fully optimized for handling large training sets, and further that, as mentioned in Chapter 6, the preliminary generalization techniques are currently insufficient to yield top-level performance, testing the size with larger training sets would be inappropriate at this time. Of course, further testing of training set size should be done as the base performance of the model is improved.

I tested the various systems with a test set of 20 randomly selected sentences from English and Spanish Bible bitexts which were not part of the training data (i.e., they were not a part of the training data for any of the tested models). The results are shown in a graph in Figure 7.1, where training set size is on the x-axis and translation-accuracy is on the y-axis.

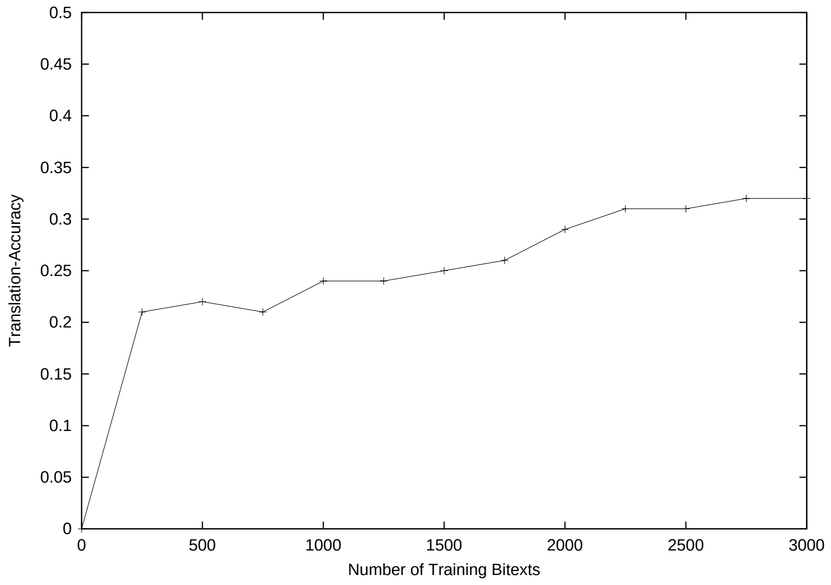


Figure 7.1: The effect of training set size on translation-accuracy

As can be seen in the figure, translation-accuracy begins at .00 with 0 bitexts, rising rapidly to .21 at 250 bitexts, then continues to generally rise—there are some falls as well, as should be expected when a few examples may incorrectly skew the model, which in turn may get outweighed by the addition of other examples—but much more slowly, to approximately .32. Note that we see an increase in translation-accuracy from .25 to .32 when we double the training set size used for earlier tests from 1,500 to 3,000.

While the overall translation-accuracy numbers remain low, this is an encouraging result, since by adding 1,500 bitexts (i.e, doubling the size) we were able to increase the translation-accuracy by 28%. Of course, as is suggested by the graph, the rate of improvement will slow (this too should be expected, since the addition of 250 bitexts should have relatively less effect at each increasing stage), and will likely taper off. This is because there is a limit on how good the system's performance can be, until significant generalization is achieved, since translation without sufficient generalization typically amounts to concatenated translations of small fragments or even words.

What should be clear from the test is that as expected, increasing the amount of training data does improve the model's performance. One might question whether the test is fair, since the smaller models have no chance of correctly translating words which they have never seen, and there are likely to be more instances of unknown words for the smaller models. However, selecting sentences which contained only words known to the smaller models (and therefore known as well to the larger models) would bias the tests towards the smaller models. That is, even if we randomly selected such sentences, it would be as if we got exactly the sentences where the smaller models would be likely to perform the best. Thus, I stuck with standard practice in such test selections, using truly randomly chosen test sets. I leave the relative performance of larger sizes and of different training sets and increased-coverage configurations, as well as questions as to how increases in training set size and translation-accuracy relate to data-sparseness and the number of unknown words, to future research. In the next two sections, I turn to expanding the model.

7.2 Extending the Model with More Linguistic Information

In earlier chapters of the dissertation I hypothesized that the linked automata model could benefit from using more linguistic information. More linguistic information can take many forms, such as using hierarchical syntactic information (as do some other MT methods, see sections 2.2.2.4 and 2.2.2.5); grouping words into classes (like the IBM model in described in section 2.2.1); enabling special handling of function words, idioms, or collocations; having separate grammars for important subdomains (such as times and dates, see section 2.2.2.6), etc. The list of possibilities here is quite large, and ranges from the mundane: special treatment of capitalized words; to the more complex: detailed semantic analysis. Using a given method of course requires that one has the means to obtain the necessary information.

The goal for this part of the dissertation research was to choose one such type of information which could be easily folded into the model, and which would be consistent with the overall automatic, data-driven approach (e.g., I did not want to hand-code a grammar), and therefore hopefully applicable to a wide range of language pairs and translation situations. I selected part-of-speech (POS) tagging.

7.2.1 A POS Tagging Experiment

POS tags can benefit a translation model in a number of ways. First, POS tags can be used to differentiate between words which are spelled the same but have multiple meanings (i.e., that are polysemous) and therefore often different translations.[1] Second, POS tags can be used to make generalizations about classes of words; for example, a specific verb followed by a word tagged as a preposition might be translated differently than the same word when followed by a word tagged as a noun. In the POS experiment to be described, I focused on the first benefit, dealing with multiple meanings.

POS taggers can be constructed automatically from tagged corpora. In this sense, the use of POS taggers is consistent with the overall data-driven approach taken in this research. The fact that the corpora must be POS tagged, however, suggests that at some point people must have done at least some of the tagging—thus the method is not fully automatic, and therefore not necessarily applicable to all languages, since the tagged data may not exist. There are however, POS taggers which do not require previously tagged corpora from which to train (although their performance will typically not be as good as those which use hand-tagged data).

[1]I will employ *polysemy* here to cover both different word senses and true lexical ambiguities.

These unsupervised tagging approaches are thus quite similar to methods which attempt to induce word classes (i.e., such as in the IBM model experiments, in section 2.2.1).

For this experiment, I wanted to use the best tagger I could find, and chose the TnT tagger for English, a Hidden Markov Model approach which is trained from previously tagged corpora (Brants 2000). There are a number of different directions I could have gone with the experiment. As with other parts of the research, I opted for as straightforward a technique as possible, and chose to POS tag only the English half of the bitexts, rather than both the English and Spanish, or just the Spanish. I hypothesized that tagging just the English would give us the most benefit in terms of research effort. This is because given an English-to-Spanish translation direction, tagging of the source language is likely to have a significant effect in terms of reducing the overall search space, since it would directly affect the first translation stage, source recognition, by allowing for better differentiation among polysemous source language words. Words and their tags could be stored together in transition labels (for example, instead of the label *word*, one would use *word.tag*). Thus two different taggings of the same word would be treated as different words by the source automaton. This means that instead of always choosing the most likely translation (i.e., activated target transition sequence) from what might have been two choices, the system only makes the single, correct choice, because tagging differentiates between the words.

POS tagging just the target language (i.e., Spanish), would likely produce fewer benefits, since the search space would be being reduced only at the final translation stage, and the number of transitions in the target would have increased. This would in turn likely result in more unconnected transitions being activated, which, as mentioned in section 6.3.4, can be problematic. POS tagging of both the source and target would be an interesting experiment, because it would allow for generalizations between categories (e.g., a source word of category X is often translated with a target word of category Y). Such generalizations could lead to better choices between translation alternatives. This is the second sort of benefit which one may derive from POS tagging (i.e., in addition to handling polysemy), but it requires more complex handling than the simple POS experiment done here, and is left as a potentially fruitful area for future research.

A final decision to be made for the experiment was when to do the tagging: before or after word alignment. This decision was easy. I assume all along that the model is built from word-aligned bitexts. Thus, to be consistent, the data should not be accessed prior to word alignment, save for choosing which word-aligner or word alignments to use. A second, and perhaps more important criteria, is that word-aligners (such as Giza^{++}) often are designed to make their own generalizations about word classes. POS tagging the data might interfere with this process and result in poorer word alignments, or might simply be duplicative. Thus, I chose to POS tag the training bitexts after word alignment.

The procedure for the experiment then was rather simple. For the same test sets as used previously (in Chapters 5 and 6), I POS tagged each English sentence in the test sets, and used these as input to a translation system where the English sentences of the training bitexts had also been POS tagged. In essence, this resulted in a source automaton with more word types than one without tagging, and hypothetically therefore finer-grained distinctions. As in the other experiments reported in this chapter, we used the Giza^{++} word alignments, since they were of the best quality.

7.2.1.1 Results of the POS Tagging Experiment

The results of the POS tagging experiment are shown in Table 7.1. The presentation follows that of some of the earlier tables. I show the test-suites and various configurations used for each training set, side by side (POS tagging on the right). The four test-suites are the same four described in Chapters 5 and 6: Test-suite 1 is actually a training set, used just to make sure no translations were lost in the process—i.e., a reality check; test-suite 2 uses all previously seen words; test-suite 3 contains some unknown words; and test-suite 4 contains training sentences which were hand-manipulated to yield new, unseen sentences. Times for the test runs are not shown, because they are typically nearly identical to those shown earlier, and in this case were run on a different computer. Note also that in the table I show only the results for configurations which used all of the increased-coverage heuristics, since these are the most important comparisons, but do again show a merged configuration below each unmerged one.

As can be seen in Table 7.1, the results were mixed, and thus inconclusive. For the feasibility test (test-suite 1), results were correct, as expected. For test-suite 2, POS tagging of the English sentences slightly degraded simple-accuracy and translation-accuracy, and for test-suite 3, POS tagging slightly improved performance. In the last test, using test-suite 4, results were nearly identical.

TS	SL	Mrg	PSP	FRG	PTP	UFT	Giza^{++} wa SA/TA	Giza^{++} waPOS SA/TA
1	20.3	no	no	yes	no	no	1.00/1.00	1.00/1.00
1	20.3	yes	no	yes	no	no	0.99/0.99	0.99/0.99
2	19.5	no	yes	yes	yes	no	0.40/0.40	0.39/0.39
2	19.5	yes	yes	yes	yes	no	0.39/0.39	0.37/0.38
3	24.3	no	yes	yes	yes	yes	0.34/0.34	0.35/0.35
3	24.3	yes	yes	yes	yes	yes	0.38/0.38	0.39/0.39
4	17.6	no	yes	yes	yes	no	0.80/0.82	0.80/0.82
4	17.6	yes	yes	yes	yes	no	0.80/0.81	0.79/0.81

Table 7.1: Summary of POS tagging experiment results *key: TS=test-suite, SL=mean source sentence length, Mrg=Merged, PSP=partial source parsing, FRG= fragment processing, PTP=partial target parsing, UFT=unknown word fall-through, CYK-wa=CYK word alignment, Giza^{++} wa=Giza^{++} word alignment, Giza^{++} waPOS=Giza^{++} word alignment with English sentences then POS tagged, SA/TA=simple-accuracy/translation-accuracy*

If we read these results on the whole as indicating that POS tagging did not significantly change performance, we need to ask why. There are several possible explanations. First, as elsewhere, improvement may be limited by the poorly generalized or poorly word-aligned base from which we start. In either case (i.e., insufficient generalization or less-than-ideal word alignment), we may be applying POS tags in many instances to data that is already somewhat incorrect or not informative enough to produce a significant change in translation performance.[2] Thus, as elsewhere, it would be instructive, for example, to re-run this test after improving

[2]For example, if merging is insufficient and/or very conservative, not many cases will arise where identical source words with different translations would exist as labels for the same transition, so it would be unrealistic to expect a big change.

the overall generalization in the model. Second, as mentioned, we only took advantage of one of the ways by which POS tags may improve translation performance (i.e., in terms of polysemy), and we might see more improvement from also making generalizations about categories, as previously suggested. A final possible explanation might be that, as being used in the experiment, POS tags may add little more information than is already gained from the word alignment process itself (since Giza^{++} does indeed use word classes), and may be subject to the same sorts of errors as automatic word-aligners. In the next section, I present a more promising result, one which focuses on increasing the coverage of the linked automata model, rather than the amount of linguistic information used.

7.3 Extending Model Coverage: Discontinuous Alignments

As mentioned earlier in this chapter, there are two different dimensions along which the linked automata MT model may be extended. The dimension discussed in the previous section centers on improving translation performance by making use of additional linguistic information. The second and perhaps more crucial dimension along which the model can be extended is in terms of its coverage. As described earlier in Chapter 4 (see especially section 4.2.2), the model architecture does not lend itself to efficient processing of discontinuous alignments. In this section, I remedy the situation, presenting a modified table architecture as well as slightly modified translation algorithms which expand the model so that discontinuities are no longer a problem. Before doing so, I present translation examples which motivate the extension and give a brief review of the alignment terminology first presented in Chapter 3.

7.3.1 Motivation

When the words of sentences in two languages are aligned, the alignment may sometimes be discontinuous. For example, in Figure 7.2, the non-adjacent German words *habe* and *gesehen* are aligned with the English word *saw*.

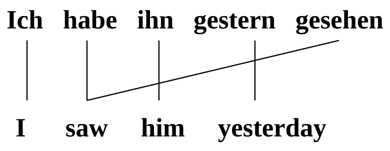


Figure 7.2: A discontinuous German to English word alignment, from Schiehlen (1998)

While, as mentioned in the chapter on alignment (see in particular sections 3.2.1 and 3.2.2), I first hypothesized that lack of coverage of discontinuous alignments would not be problematic, I soon realized that discontinuous alignments are quite prevalent in the data (see section 3.2.4). For example, in Figure 7.3 (repeated from the earlier Figure 3.6) the discontinuous English words *set* and *up* are aligned with the Spanish word *erigió*, in a sentence fragment from Genesis. In fact, during the hand-aligning process, at least 15% of the bitexts contained an alignment where discontinuous alignment was preferable. Discontinuous alignments may also be prevalent in word alignments which are automatically produced. For example, in the Giza^{++} word alignment set for Genesis, where singleton source words can be aligned with many target words (see section 3.2.3), over 50% of the bitexts exhibit some *target discontinuity* (see section 7.3.2).

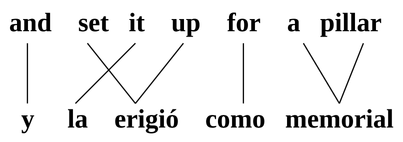


Figure 7.3: A discontinuous English to Spanish alignment from Genesis

As can be seen from these sorts of examples, MT systems which model word alignments need to be able to account for discontinuous alignments. In purely finite-state translation systems, such as those which use transducers, incorporating these discontinuities directly into the model is made difficult by the lack of ordering flexibility (i.e., discontinuities cause even more difficult ordering issues for transducer models than those identified in the literature review, in section 2.2.2.3). In section 7.3.3, I will show how the basic separation between language and alignment models of the linked automata can be exploited to handle such alignments.

7.3.2 A Review of (Discontinuous) Alignment Terminology

Discontinuous alignments are a property of translations (i.e., of *bitexts*). They are therefore to be distinguished from other discontinuous phenomena, such as topicalization or scrambling, which can occur in the individual source and target sentences of bitexts. For example, in Figure 7.4, although in the German sentence the pronouns are some distance from the verbs which take them as arguments, the alignments produced with the English translation are all continuous.

[4]Note that in all of the word-aligned examples, I and not the cited authors made the alignments, and bear responsibility for any errors.

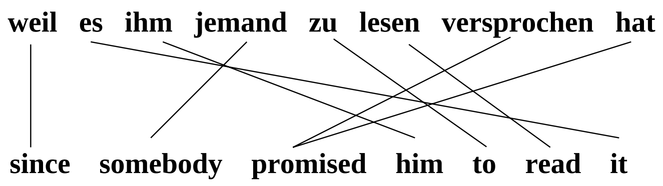


Figure 7.4: Discontinuous constituents without discontinuous alignments (Müller to appear)[4]

Thus, single sentence discontinuity does not necessarily pose problems for alignment-based translation models (at least not on the alignment model level), because the alignments themselves may still be continuous.

To help demonstrate exactly what sort of alignments are considered discontinuous, I now review the alignment terminology from section 3.2.1. An *alignment* is a correspondence between (possibly non-contiguous) sequences of source and target words of a bitext. The set of all such correspondences in a bitext (i.e., the mapping between source and target words) is called a *word alignment*. I refer to the source words of an alignment as *source anchors* and the target words as *target anchors*. An alignment is *discontinuous* if either the source anchors (*source discontinuous*) or target anchors (*target discontinuous*) do not form a contiguous substring (i.e., they are not adjacent). So, discontinuous alignments can only occur where either the number of source or target anchors is greater than one.

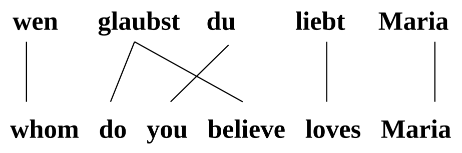


Figure 7.5: Target discontinuity in a German to English word alignment (Kallmeyer 2000)

Consider the word alignments shown in Figure 7.5 and earlier in Figure 7.3. In Figure 7.5, a plausible word alignment might be to align *glaubst* with *do* and *believe*.[5] This alignment is target discontinuous. In Figure 7.3, as in Figure 7.2, we have an example of source discontinuity.[6]

The notion of aligning conceptually similar entities follows Brown *et al.* (1993), where the set of source anchors viewed as generating target anchors in an alignment are call *cepts* (evocative of part of a concept). Figures 7.2, 7.5, and 7.3 all involve complex verb predicates (CVPs) (Schiehlen 1998). The types of discontinuous alignments, if any, will depend on the pairs of languages (e.g., discontinuous CVPs need not result in discontinuous alignments if both languages divide the semantic entities similarly).

[5] *do* encodes the tense and person of *glaubst* and *believe* the meaning, so it is reasonable to view them as a unit (see Schiehlen (1998)).

[6] Note that any example of source discontinuity can be viewed as an example of target discontinuity (and vice versa), if we invert the senses of source and target (i.e., if we reverse the translation direction).

7.3.3 Handling Discontinuous Alignments in the Model

In this section I describe how the linked automata model can be used to process both source and target discontinuities. As mentioned in the previous section, discontinuous alignments can exist on either side of the translation relation. Other finite-state MT models either cannot handle these discontinuities, or, as in techniques to handle word-order differences, insert special symbols to account for them (see, for example, Sanchis *et al.* (2001)). Here I present what I view as a more direct encoding of such alignments.

7.3.3.1 Target Discontinuities

For the linked automata model, the treatment of target discontinuities is easy. Recall that in the last translation stage (see section 4.5) activated target transition sequences are collected to form a smaller target automaton. This process is shown in Figure 7.6 steps 6 and 7 (repeated from Figures 4.7 and 4.11 which were shown with the description of the translation process).

Contiguous transitions can be joined into single transitions which span the same range of states. This step has the benefit of reducing the size of the automaton produced. Joining the transitions in this way also allows for a clear accounting of the source words which generated the transitions, in cases where source transition sequences are aligned with more than one target transition. This occurs in the alignment of bitexts shown in Figure 7.7 (repeated from Figure 4.2 shown earlier). There, *fish* is aligned with *le poisson*, and thus the two transitions produced during

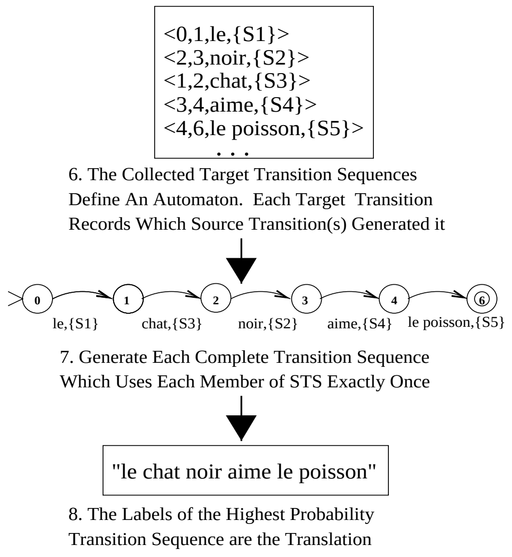

<0,1,le,{S1}>
<2,3,noir,{S2}>
<1,2,chat,{S3}>
<3,4,aime,{S4}>
<4,6,le poisson,{S5}>
. . .
6. The Collected Target Transition Sequences Define An Automaton. Each Target Transition Records Which Source Transition(s) Generated it
le,{S1}
chat,{S3}
noir,{S2}
aime,{S4}
le poisson,{S5}
7. Generate Each Complete Transition Sequence Which Uses Each Member of STS Exactly Once
"le chat noir aime le poisson"
8. The Labels of the Highest Probability Transition Sequence are the Translation

Figure 7.6: The final translation stage

translation: $\langle 4,5,le\rangle$ and $\langle 5,6,poisson\rangle$ are combined to form $\langle 4,6,le\ poisson,p,S5\rangle$, where the SWS (this is the source-word-store, see section 4.5.3), $S5$, represents *fish*, and p represents the probability of this combined transition.

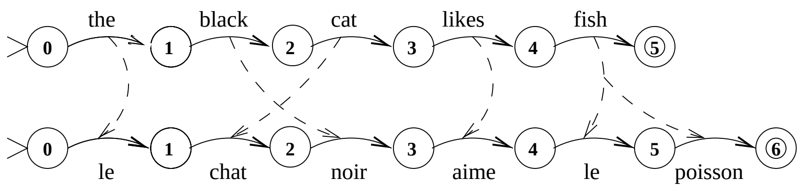


Figure 7.7: A bitext in the linked automata MT model

When target transition sequences are generated from the same source transition sequences and are not contiguous, they cannot be joined (i.e., such sequences are not permitted by the target language model). So, the probability and SWS need to be allocated among the individual target transitions. In the case of the probability, a number, spreading the value over the target transitions is just a matter of dividing it appropriately among the parts.[7] The SWS, on the other hand, represents a set of words. It is implemented as a bit vector representing the set of source words covered. The correct step then is to change the SWS from a vector of bits to a vector of rational numbers. Given n generated target transitions, each transition gets $1/n$ of the given source word representation, and a transition has a full SWS only when all of the vector positions contain a 1.[8]

7.3.3.2 Source Discontinuities

Recall that in the second translation stage (see section 4.5.2), the transitions of the most probable source transition sequence (STS) are used to activate target transition sequences. This is done by first computing the substring closure (the set of all substrings—i.e., those transition sequences that are contiguous) of the STS. As mentioned previously, this use of the substring closure naturally precludes the use of discontinuous source alignments. For a source sentence of n words, the

[7]This depends on the interaction of the various probabilities. For an alignment probability, a, and n non-contiguous transitions, $\sqrt[n]{a}$ should be allocated to each transition (this becomes $\frac{a}{n}$ in the log domain).

[8]This change may cause a slowdown, since fast bit operations are disallowed. For the results described, I implemented a hack of giving an SWS value of 1 to one target transition of a sequence, and 0 to the others. This permits the transitions to be later joined, but does not guarantee that the most probable result will be found.

substring closure contains $n(n+1)/2$ substrings, plus the empty string. So, the time needed to retrieve target transition sequences is quadratic with respect to the length of the source sentence. Given sentences of a typical length (<100 words in our translation domain), which can be broken up into parts if necessary, and the fast operation of the target transition retrieving table function, the time needed for retrieval comprises a negligible portion of the overall translation time.

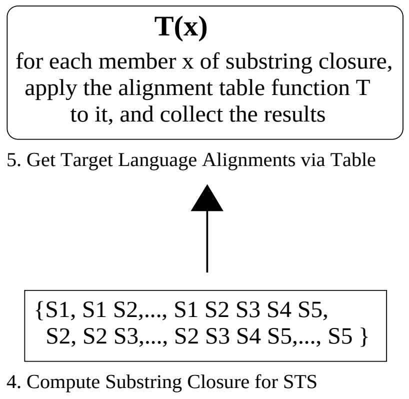

T(x)
for each member x of substring closure, apply the alignment table function T to it, and collect the results
5. Get Target Language Alignments via Table
{S1, S1 S2,..., S1 S2 S3 S4 S5, S2, S2 S3,..., S2 S3 S4 S5,..., S5 }
4. Compute Substring Closure for STS

Figure 7.8: The second translation stage as presented with the simple table architecture of section 4.5.2

When we move to discontinuous sequences of source transitions, however, this number can quickly explode, and is, in fact, exponential with respect to the source length. If we think of the source transition sequence as being a set, the set of all (possibly non-contiguous) sequences is the power set of the set of transitions (the *sequence closure*), with cardinality 2^n. Using the notion of trying to apply the table function, T, as shown in Figure 7.8 (repeated from Figure 4.10), to each possible sequence is untenable. For example, consider a source sentence of 30 words. Using the substring closure, this means 466 applications of the function. For the sequence closure method however, this means 1,073,741,824 applications! The time needed to check each sequence would quickly overwhelm the system.

Given the simple table architecture, as described earlier and shown on the left-side of Figure 7.9, one could attempt a brute-force solution of using the table itself to decrease the search space. This would mean going through every source transition sequence in the table and checking if the sequence contains only transitions retrieved after source parsing. While this is an improvement over the sequence closure method, it is far from ideal, since the number of different STSs used in alignments can be very large. For example, in a completely ungeneralized system,[9] trained with just over 1,500 bitexts, the table contains over 30,000 STSs. The problem with this solution, like the sequence closure method, is that many STSs are being tried which have no chance of succeeding.

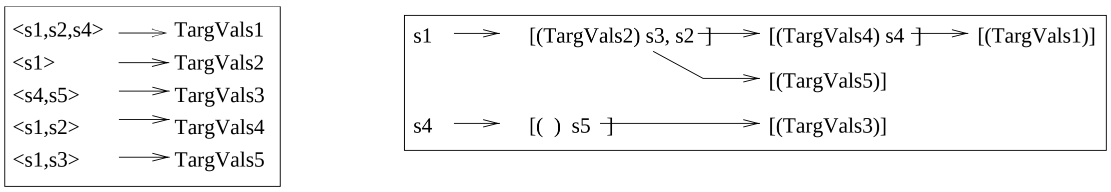


Figure 7.9: Two table architectures: simple (left) and discontinuous capable (DC) (right)

A much better solution is to make a modification to the table, so that we can use the words of the source sentence to decrease the search space. Recall that tables

[9]Generalization greatly reduces this number.

are functions from source transition sequences to sets of pairs of target transition sequences and probabilities. I will abbreviate the elements of this range by simply calling them *target values*, since their implementation will not change. The simple table can be envisioned as a tall, narrow table, implemented via a hash from STSs to target values (see Figure 7.9). This implementation means that to check the table's hash keys for a given transition, we must check each key.

The modification to the table which I will make (called a DC table, for *discontinuous capable*), is to store each STS as a chain of individual transitions. At the top level, each source transition used in an alignment points to a data structure called a *link box*. A link box contains two elements: 1) A (possibly empty) target value for the transition sequence which points to the box, and 2) a (possibly empty) list of transitions, each which points to other link boxes (see Figure 7.9). This makes for a shorter, wider table than a simple table, and means the table can be searched based on individual transitions (i.e., those which begin a given transition sequence) rather than an entire transition sequence. This has the effect of greatly reducing the search space, since one need only check the transitions that actually arise as a result of recognizing the source sentence (rather than all of them, as in the brute-force approach, or a huge number that do not exist, as in the sequence closure approach).

Consider a hypothetical source sentence of five words, represented by the transition sequence: $s1, s2, s3, s4, s5$. Rather than trying 32 possible subsequences, 16 possible substrings, or all of the potentially many STSs in the (simple) table, we simply, for each transition s_i, get the associated link box, $T(s_i)$, if there is one, and store its target value. We then proceed recursively with any transitions in the link box's transition list which are also transitions of the sentence we started with.[10] In this manner, we let the words of the sentence decrease the search space. The speed of this step improves from roughly quadratic in the substring closure method, to nearly linear using the DC table, in a typical case, since even DC tables will be much taller than they are wide (chains of transitions tend to be quite short, and can never be longer than the sentence(s) from which they came).

7.3.4 Evaluation

Following the feasibility test methodology used previously for the basic system, in section 5.4, I first tested modified and unmodified systems with training sentences, to see if discontinuous alignments could be processed. Again, because I used training sentences, these tests do not assess the model's translation performance, but rather the capability of the different architectures—that is, I wanted to make sure that I had indeed extended the coverage of the model.[11]

I used training sentences for this first test because doing so was the only way in which I could guarantee that I was testing the handling of discontinuous alignments. The goal for a test of the discontinuous alignment architecture is to see if bitexts which make use of discontinuous alignments can be correctly translated. To find such bitexts which use discontinuous alignments (i.e., which would need to make use of these alignments), we need to be able to inspect the alignments. Since only

[10]This replaces steps 4 and 5 in Figure 7.8.

[11]Note that the table modifications should have no effect on the translation performance for translations where discontinuous alignments do not arise; but they crucially provide a means to process translations when they do.

training sentences have alignments which we can inspect, they are the only option.[12] Thus this use of training sentences should again be viewed as a feasibility test of the discontinuous alignment modifications, where the focus should be on the difference in performance between the unmodified and modified architectures. That said, at the end of this section, I will, for completeness, present results which used true test sentences, and describe some of the steps which I made to construct appropriate test sets, as well as additional partial parsing techniques I employed to better draw out distinctions in performance.

I used the same Giza^{++} word-aligned training set as in the POS-tagging experiment in this chapter (i.e., the paired English and Spanish verses of Genesis, see section 7.2.1). The Giza^{++} data set was useful because it contains some alignments which are target discontinuous. I selected 10 bitexts at random from those which exhibited at least one alignment which was target discontinuous, and used these as the test set.

To test source discontinuity, I made use of a feature of the linked automata system briefly mentioned in Chapter 4: I inverted (or *reversed*) the translation system. Inverting a linked automata model is a simple process, and in effect amounts to renaming the source language model to the target language model (and vice versa), and making one pass through the alignment table to invert the alignment function. I also inverted the test set. As in other tests, I report both simple-accuracy and translation-accuracy.

[12]Test sentences may or may not have word alignments, i.e., one might use the same word alignment techniques on test data as used on training data (although, of course, test sentences need not be aligned—all that is needed are the source and target sentences); but there is no reason to assume that these alignments are relevant to those in the trained-on model, since the model has never seen them.

The results for the test are shown in Table 7.2. In terms of testing target discontinuity, without the modifications described for handling the activated discontinuous target transition sequences (section 7.3.3.1), one would expect the system to have a lot of trouble, even with training sentences. That is indeed the case, as shown on the first line of the table results. Simple-accuracy and translation-accuracy were .39 on these trained-on sentences. Note that these tests did not use any of the methods for extracting partial results (i.e., the increased-coverage heuristics). Thus, without the modifications for discontinuous alignments, some activated target sequences could not be used at all. When the modifications for target discontinuity were invoked, however, accuracy returned to its expected level (see the second results line of the table).

Source	Target	SrcLength	DC-modified	SA/TA	Time
English	Spanish	25.7	no	0.39/0.39	3.5
English	Spanish	25.7	yes	0.99/0.99	1.8
Spanish	English	23.0	no	0.00/0.00	3.0
Spanish	English	23.0	yes	1.00/1.00	3.0

Table 7.2: Summary of discontinuous alignment test results *key: SrcLength=mean source sentence length, SA/TA=simple-accuracy/translation-accuracy, Time=mean run time (seconds)*

Turning to discontinuous source alignment tests, we look at the last two lines of the table. Using the inverted translation system and the inverted test set, the unmodified system (i.e., one which used a simple table) had no means to activate the necessary transitions. In other words, discontinuous alignments could not be used at all, since they would never even be tried using the substring closure method first given in section 4.5.2. This means in turn that no complete target parses could be found, since the SWSs could never be full (at least two source words would never be represented). Again, since none of the increased-coverage techniques were invoked for these tests, the unmodified architecture was expected to fail completely, as it did, with accuracies of 0.00. As in the target discontinuity case, the architecture modification again proved to be successful. Use of the DC table allowed the source discontinuous alignments to be handled perfectly.

7.3.4.1 On Finding A Test Set for Discontinuous Alignments

As mentioned at the beginning of section 7.3.4, tests on training sentences are likely the best means to check whether the discontinuous alignment architecture is functioning as planned. This is because training sentences have alignments which we can inspect, in order to discern if discontinuous alignments would be necessary for accurate translations, absent other means to produce the same translations (e.g., increased-coverage techniques, similar bitexts with non-discontinuous alignments, etc.). Nevertheless, I also wanted to see if I could create tests which were at least somewhat appropriate using real test bitexts (i.e., those which the model had not used for training). As mentioned, this turns out to be rather difficult. To know that it is only the supposed handling of discontinuity which makes the difference in translation performance, one needs to make sure that discontinuous alignments are the only means by which a correct result would be achieved and that these alignments are selected (i.e., that they have the highest probability, and, in the case of source discontinuity, that all the source anchors are recognized as a unit).

Both of these criteria are difficult to enforce. Since we typically also need to invoke increased-coverage heuristics to achieve accuracies where we can even begin to make comparisons, and since different alignments can be used to produce the same results, it is not transparent which alignments are ultimately responsible for producing a translation (recall that translation involves the interplay of three models, as well as, on occasion, the heuristics for extracting partial results). As far as the second criterion, whether we would select the relevant alignments at all, we would only expect to see differences in results where the alignments in question were selected—meaning they came from a highest probability source recognition, and then were the highest probability alignments this source transition sequence had (and which also ended up being used in the final translation stage). Given the generally low accuracies of the model so far, this means that many times these alignments would not be selected in the first place, making comparison between architectures impossible (i.e., modified and unmodified systems would produce incorrect results).

In order to ameliorate these difficulties, I sought to create test sets which would give us the best chance to bring out the differences afforded by the expanded coverage. Certainly, if discontinuous alignments were to be used, they could only be invoked if the words used in the alignments had been seen before. So, as in some of the tests from Chapter 6 (namely, test-suite 2), I made sure that all of the source words used in the test bitexts had been seen at least once in the training data. For the same reasons, I also made sure that all the target words had also been seen—knowing that in addition, a translation system could normally never produce target words which it had never seen (i.e., unless words happened to be spelled the same in the two languages and unknown word fall-through was employed). The last step I took, even though I knew it bore no direct relevance on the alignments in the constructed model, was to word-align some of the potential test bitexts using Giza^{++}. I then selected only bitexts which showed at least one target discontinuity (recall that this is the only option available with this training set), with the hopes that these same discontinuities existed in the trained model, and could thus be exploited. I found 18 such bitexts in the potential test bitexts which I had word-aligned, and I refer to them as *test-suite 5*.

As mentioned, these testing efforts also had some interesting side-effects in terms of the implementation of new (parts of) partial parsing algorithms. Specifically, I implemented the source partial parsing technique of first finding the longest, most probable recognition, rather than the longest, most probable recognition beginning with the first word (this idea was first discussed in section 6.3.3), hoping that such a step would improve the overall accuracy. Along these same lines, I also implemented the partial source parsing technique briefly touched on (again, in section 6.3.3) which collects recognized source sentence parts so that they may be translated as a unit, rather than one at a time (I refer to this technique as *collecting partial source parsing*). This technique makes it more likely that discontinuous source transitions will be used during the activation stage of translation (i.e., that the alignment function will be applied to them), since when using a non-collecting strategy, source anchors recognized separately might not be present in the same activation stage (since the various stages will occur more than once). These two partial source parsing ideas can be used together. Lastly, so that the activated transitions produced by a collecting partial source parsing strategy can be ordered in the absence of better information, I implemented the third partial target parsing heuristic which I identified in section 6.3.4, which I call the *suggested-ordering* heuristic, in which the source sentence word order can be used to influence the activated target transition order. I will touch on some of the more interesting properties of these techniques, and the motivation for them, as I present the test results.

TS	Source	Target	SL	DC Mod	Mrg	PSP Method	PSP Match	SA/TA	Time
5	English	Spanish	23.4	no	no	F	S	0.31/0.32	21.9
5	English	Spanish	23.4	yes	no	F	S	0.34/0.34	18.7
5	English	Spanish	23.4	no	yes	F	S	0.37/0.38	14.6
5	English	Spanish	23.4	yes	yes	F	S	0.41/0.42	14.3
5	English	Spanish	23.4	no	no	L	S	0.31/0.32	17.8
5	English	Spanish	23.4	yes	no	L	S	0.34/0.34	17.9
5	English	Spanish	23.4	no	yes	L	S	0.36/0.37	15.5
5	English	Spanish	23.4	yes	yes	L	S	0.39/0.40	15.3
5	English	Spanish	23.4	no	no	L	C	0.29/0.30	3.2
5	English	Spanish	23.4	yes	no	L	C	0.32/0.33	3.2
5	English	Spanish	23.4	no	yes	L	C	0.32/0.33	2.9
5	English	Spanish	23.4	yes	yes	L	C	0.35/0.36	2.8

Table 7.3: Summary of test suite 5 results for target discontinuous alignments *key: TS=test-suite, SL=mean source sentence length, DC-Mod=adjust for discontinuous alignments, Mrg=Merged, PSP=partial source parsing (Methods: S=standard, C=collecting; Matches: L=most-probable longest, F=most-probable longest—with first word), SA/TA=simple-accuracy/translation-accuracy, Time=mean run time (seconds)*

I first experimented with target discontinuities, and present the results in Table 7.3. I show 12 runs of six different basic configurations; there could of course be other combinations than are shown, but I present those that I found the most interesting. For each of the six basic configurations, I first show the test results for the system unmodified for discontinuous alignments, then show the same configuration with the discontinuous handling enabled. For example, the first line shows the results of an unmerged system using the standard partial parsing techniques (i.e., non-collecting) and the same longest-most-probable match containing the first word strategy as reported in earlier tests. The next line in the table shows the same configuration with the discontinuous handling enabled. As can be seen, the accuracy measures improve when the discontinuity architecture is used. Following these two lines of results, I show the results for the same configurations, but this time using merged systems. Encouragingly, not only do we also see that the discontinuous handling improves the accuracy results, but that merging yields a real benefit (although the merging result did not hold up for the source discontinuous tests). The next two groups of four show the use of the longest-most-probable match strategy, and the longest-most-probable match strategy together with collecting partial source parsing, respectively (note that for all of the results in this and the next table, I also used the new partial target parsing heuristic, suggested ordering). In both of the sets of four, the same basic results hold: Discontinuous handling improves accuracy, as does merging.

Surprisingly, however, the longest-most-probable match strategy actually slightly degrades performance, as shown in the merging cases (the seventh and eighth result lines in the table). I have yet to determine why this should be the case and hypothesize that it may be simply a coincidence, but leave the answer for future testing. The accuracy again lowers when the collecting method is invoked. This should not be too surprising, since the target language model (at least in some senses) has to do more work, without much more information. I also suspect there may be some (negative) impact from the fact that all of the 1:0 alignments (the empty transitions) are available at this final translation stage, leading to more spurious deletions.

Given that both of these techniques were implemented rather quickly, I suspect I should be able to improve their performance, and see their combination as probably the best way to go for the model (especially since they may show dramatic improvements over the non-collecting techniques for language pairs whose word orders are quite different). Additionally, a real positive for the collecting strategy is a run time improvement of nearly an order of magnitude better than the non-collecting strategy. As discussed in section 6.4, this speed-up is due to using the last translation stage only once (note that this result also runs counter to the tendency of lower accuracy scores to take longer, suggesting that even faster run times will occur as these techniques are improved). In any case, the major point to take from these results is that where the only difference between two systems was the ability to process discontinuous alignments, for bitexts where discontinuous alignments were at least somewhat likely to be needed, performance improved.

When I turned to source discontinuities, results were not so easy to come by. Reversing the translation systems in the same manner described for the tests reported in Table 7.2, so that Spanish was the source language and English the target (thus giving us hopefully some source discontinuous alignments) I ran the same basic test configurations. Invariably, discontinuous alignment enabled scores were always identical to the unenabled scores (thus I do not show these results). Why should this be? The answer is that the relevant discontinuous alignments were never invoked. The reason that this is the case is that such alignments are more difficult to invoke than target discontinuous alignments. To activate a target discontinuous alignment, only one source word need be recognized (recall that the Giza^{++} alignments are one-to-many, where there may be discontinuities in the target).

When a translation system is reversed, however, the discontinuities move to the source language. Given the basic recognition strategy, this means that to activate the same discontinuous alignments that existed before, the source parsing stage would have to recognize not one word, but at least two, and these two would never be adjacent. This is a much more difficult task, and the partial source parsing algorithms have not been adapted to fully handle such a situation (indeed, an appropriate treatment of this problem—which generalizes to full-blown regular expression matching for handling discontinuity in partial parsing, as mentioned in section 6.3.3—is a likely dissertation topic in its own right). This means that activation of the appropriate transitions is even more dependent on very good partial source parsing—and that the recognitions not only need to span more words, but more than ever need to match the alignments. For example, if the English *the men* are source discontinuously aligned with the Spanish *hombres* in the bitext shown in (81), but the partial parsing algorithm (or any of the other algorithms which segment, such as unknown word fall-through or the technique for breaking up long sentences) segments in the middle of such an alignment (e.g., between *good* and *men*), then the proper alignments would not be available to the system.

(81) the good men
hombres buenos
((the men:hombres)(good:buenos))

Given that proper segmentation models (such as the rift technique identified in section 6.3.3) are beyond the scope of this research, and will still not guarantee proper segmentation for discontinuities (i.e., in some cases, segmentation may not be possible, given very long discontinuities), things might appear dire. But the situation is not as bad as it seems. The recognition pattern need not match the alignment pattern, so long as all the relevant source anchors are available at the same time. For example, if the partial source parser recognizes first: *the good* and then: *men*, the discontinuous alignments will still be available, if the source transition sequences produced are used all at once to generate the appropriate alignments. This is the reason why the collecting partial source parsing technique was invented. In cases where source parsing is less than optimal, it still gives us a chance to activate the appropriate alignments.[13] Unfortunately, even with this modification to the partial source parsing strategy, I was unable to demonstrate that the modified architecture was more suitable for discontinuous alignments.

Thinking about the situation a bit more, I realized there still might be a way to demonstrate the capability of the architecture on real test sentences. I was convinced that the lack of difference between test runs on the modified and unmodified architectures was due to the combination of the relatively poor recognitions and the

[13]Note that these difficulties are independent of the discontinuous alignment architecture, and are more indicative of familiar problems that occur in the processing of single language sentences (i.e., as opposed to being a property of bitexts), where long-distance dependencies have often caused problems for tools such as parsers, partial-parsers, and taggers.

overall low-accuracy, but that the architecture was still correctly built to handle source discontinuities which otherwise could never be processed (i.e., with a simple table and not using the brute-force techniques as described in section 7.3.3.2). We had already seen, however, in section 7.1, that increasing translation size could increase translation-accuracy. Also, increasing the size of the test set might yield some bitexts where the situation would be right so that discontinuous alignments would be activated, if permitted by the architecture. So, I doubled the size of the translation system (to just under 3,000 Spanish and English verses), and using this new system, randomly selected 50 test bitexts as before (where source and target words were known to the system, and Giza^{++} would align each bitext with at least one target discontinuity). This test set is called *test-suite 6*. I then reversed the translation system, and ran the tests, translating from Spanish to English.

TS	Source	Target	SL	DC Mod	Mrg	PSP Method	PSP Match	SA/TA	Time
6	Spanish	English	23.5	no	no	L	C	0.34/0.35	13.5
6	Spanish	English	23.5	yes	no	L	C	0.34/0.36	12.7

Table 7.4: Summary of test suite 6 results for source discontinuous alignments *key: TS=test-suite, SL=mean source sentence length, DC-Mod=adjust for discontinuous alignments, Mrg=Merged, PSP=partial source parsing (Methods: C=collecting; Matches: L=most-probable longest), SA/TA=simple-accuracy/translation-accuracy, Time=mean run time (seconds)*

The test-suite 6 results are shown in Table 7.4, where for brevity, we show just the results for an unmerged configuration, using collecting partial parsing and a longest-most-probable match strategy (results for the other configurations patterned similarly). On four of the 50 sentences, the translation-accuracy improved when the discontinuous alignment modifications were enabled. On none of the sentences was the performance degraded. These results cause only a slight change in the overall accuracy numbers shown in the table (although of course small changes are more significant given a larger number of test sentences), but serve to make the point. The modified architecture correctly processes source discontinuous alignments, and the unmodified architecture does not. The degree to which such differences will be exploited in a translation situation is dependent, as mentioned, on the overall quality of other translation operations, such a recognition, as well as on the overall prevalence of discontinuous alignments in the training data. Nevertheless, the results sufficiently demonstrate that the architecture has extended the coverage of the model as desired.

Thus, the results shown in Tables 7.2–7.4 demonstrate that the extensions to the model are successful, for both target discontinuous and source discontinuous alignments. The linked automata approach, so modified, handles any possible type of word alignment, leaving future development of the model free to focus on improving translation accuracy. These results epitomize one of the basic strengths of the model: Its straightforward architecture of decoupled language and alignment models allows for relatively easy modifications which can expand its coverage. This then leads us to the final section of the dissertation, where I briefly assess the linked automata model's present and its future.

7.4 Conclusions: Where the Model Stands and a Look to the Future

In this final section of the dissertation—having completed a preliminary exploration into the linked automata MT model—I will spend a moment considering how we got here, and what directions this research will next take.

7.4.1 The Present

The linked automata model was created with the intention of filling a void in data-driven MT. Probabilistic finite-state machine translation systems, while limited in power, have shown some potential to serve as the foundations of MT systems. Finite-state techniques are attractive because they can often yield very efficient processing within a well-understood and well-researched domain. Most finite-state MT applications have employed transducers in one form or another. It is the nature of these devices to tightly couple source and target language words by means of single transitions. In essence such devices used as MT systems are intended to combine language and translation models into one. As was demonstrated, this coupling makes the direct representation of word-order differences between languages quite difficult, and poses additional challenges for the representation of alignments between sequences of words, in particular those that are discontinuous.

The linked automata MT model was created in order to expand such finite-state MT approaches, so that ordering differences and discontinuities can be directly stored in a model. By breaking apart the source and target language models from their alignment, this augmented finite-state device achieves the flexibility necessary for storing all of the possible word alignment relationships that may occur in bitexts.

In addition to providing this model which is more expressive than pure transducer-based approaches, I have also provided detailed descriptions of the means by which it may be constructed, used as a translation device, preliminarily generalized, and expanded in terms of both information used and coverage, as well as presented detailed algorithms and implementation details for its efficient use at crucial points in the MT process. Ideally I have also identified those aspects of the model which remain problematic and suggested possible steps for improvements.

In addition, I have attempted to demonstrate how the separation of language from alignment models allows the linked automata system to be treated as a database from which translation information can be mined. It is in this sense that the linked automata form the stone which serves as the catalyst for the good soup to come—as in the folktale. Hopefully too I have been able to demonstrate in this chapter and the preceding one how the model can be extended beyond its modest means, both in terms of the use of techniques for making use of partial results and in terms of the use of additional linguistic information. In this manner the model may be expanded beyond its finite-state framework.

The linked automata model is a data-driven statistical approach which makes little use of syntactic or semantic information in a traditional sense. As such, its place in the world of machine translation is clear, in that in its most basic form—like many other finite-state models—it does not have sufficient means to fully process unrestricted natural language translation. My goal in this research, however, was to begin to explore how far the system could be pushed and to demonstrate that the model can be further augmented to serve as a foundation for more powerful systems.

7.4.2 The Future

One of the keys to improving the model's performance will be to improve its ability to generalize. Better generalization will have the twofold benefit of improving scalability (i.e., it will keep the model to a reasonable size, although not necessarily reduce search space) and coverage. In the model's future there will also need to be more efficient techniques for translation, which will likely involve improved methods for handling null alignments. Along with these developments, additional testing on the use of more linguistic information will be called for, so that the model operates less blindly.

As I write the last sentences of this chapter, a number of ideas for improving the system come to mind—from better implementation techniques—to cleaner definitions of the model—to ways to incorporate more information—to ways to hybridize the model with other MT techniques and human input. Some of these ideas will hopefully prove promising in future research. The ideas fall into two major directions in which the linked automata research may turn.

The first research direction is the evolution of the linked automata model as an MT system. This dissertation has presented a number of ideas for expansion and improvement, and clearly portions of the overall architecture and algorithms can be made better. Of equal importance is the second research direction: Linked automata as formal devices need to be investigated, apart from their use as models for machine translation. Certainly different classes of linked automata can be defined, each with particular properties in terms of their expressive power, efficiency, and ability to be combined and augmented. Research in either of these directions will ideally inform the other. I hope that the dissertation has provided enough of a kernel from which such research can be continued, both by me and by others.

Appendix A

TRANSLATION EXAMPLES OF VARIOUS TRANSLATION-ACCURACIES

To help give an idea of what the translation-accuracy numbers reported in the dissertation mean (as well as to acquaint readers with the type of data used), in this Appendix I present a small sample of actual translations of test sentences. The test sentences all come from Bible verses on which the system was not trained, and were drawn from various tests run for the dissertation, as produced by linked automata models trained on approximately 3,000 Biblical verses.

Two examples are shown for each of five different translation-accuracy ranges, where 0.00 is the worst possible score, and 1.00 is a perfect score.[1] For systems trained on 3,000 verses and test sentences where all of the source words had been seen at least once during training, typical translation-accuracies were in the low .40s, at the close of the research reported in the dissertation.

In each range one of the examples is from English to Spanish, and the other from Spanish to English. Translation-accuracy scores and ranges are reported to the left, and the source sentence, the reference translation (that from the Bible), and the resulting translation that the given system produced are to the right.[2]

[1]Translation-accuracy is defined in section 5.4, on page 188.

[2]All text is shown in lower-case, since for the results reported in the dissertation, the systems ignored case distinctions (as well as punctuation).

.00–.19

1)	.13	*Source*	and eleazar the son of aaron the priest shall be chief over the chief of the levites and have the oversight of them that keep the charge of the sanctuary
		Reference	el principal de los jefes de los levitas era eleazar hijo del sacerdote aarón dirigente de los que estaban a cargo del santuario
		Result	eleazar hijo del sacerdote aarón jefe sobre jefe de levitas y devolved personalmente de los que cumpliréis la ordenanza de
2)	.17	*Source*	todo su fruto servirá de comida a tu ganado y a los animales que hay en tu tierra
		Reference	and for thy cattle and for the beast that are in thy land shall all the increase thereof be meat
		Result	all fruit thereof anoint eaten thy cattle and abram beast that is in thy land

.20–.39

3)	.21	*Source*	and if it be any unclean beast of which they do not offer a sacrifice unto the lord then he shall present the beast before the priest
		Reference	si se trata de algún animal inmundo que no se puede presentar como sacrificio a jehovah entonces el animal será puesto delante del sacerdote
		Result	si trae un sacrificio alguna inmunda animal del cual no jehovah presente animales delante sacerdote
4)	.39	*Source*	también ofrecerá a jehovah el carnero como sacrificio de paz junto con la cesta de tortas sin levadura luego presentará su ofrenda vegetal y su libación
		Reference	and he shall offer the ram for a sacrifice of peace offerings unto the lord with the basket of unleavened bread the priest shall offer also his meat offering and his drink offering
		Result	and offer unto the lord the ram for a sacrifice of peace besides basket of cakes unleavened and bring his offering meat and his libación

.40–.59

5)	.43	*Source*	este mes os será el principio de los meses será para vosotros el primero de los meses del año
		Reference	this month shall be unto you the beginning of months it shall be the first month of the year to you
		Result	this month be to you beginning of be holy unto you months first the year
6)	.50	*Source*	and he said unto his people behold the people of the children of israel are more and mightier than we
		Reference	he aquí el pueblo de los hijos de israel es más numeroso y fuerte que nosotros
		Result	sus pueblo de los hijos de israel has hecho más poderoso que nosotros

.60–.79

7)	.61	*Source*	then will i remember my covenant with jacob and also my covenant with isaac and also my covenant with abraham will i remember and i will remember the land
		Reference	yo me acordaré de mi pacto con jacob y me acordaré de mi pacto con isaac y de mi pacto con abraham y me acordaré de la tierra
		Result	entonces yo me acordaré de mi pacto con jacob también mi pacto con isaac mi pacto con abraham yo acuérdate me acordaré tierra
8)	.79	*Source*	nuestros niños nuestras mujeres nuestros rebaños y todo nuestro ganado quedarán allí en las ciudades de galaad
		Reference	our little ones our wives our flocks and all our cattle shall be there in the cities of gilead
		Result	our little our wives our flocks and all our cattle quedarán there in the cities of galaad

.80–1.00

9)	.83	*Source*	manda a los hijos de israel que te traigan aceite de olivas claro y puro para la iluminación a fin de hacer arder continuamente las lámparas
		Reference	command the children of israel that they bring unto thee pure oil olive beaten for the light to cause the lamps to burn continually
		Result	command children of israel that they bring thee pure oil olive beaten for the light to cause the lamp to burn always
10)	1.00	*Source*	and every raven after his kind
		Reference	todo cuervo según su especie
		Result	todo cuervo según su especie

BIBLIOGRAPHY

AHO, ALFRED V., & JEFFREY D. ULLMAN. 1972. *The Theory of Parsing, Translation, and Compiling, Volume I.: Parsing*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

AHRENBERG, LARS, MAGNUS MERKEL, ANNA SÅGVALL HEIN, & JÖRG TIEDEMANN. 1999. Evaluation of LWA and UWA. *Working Papers in Computational Linguistics and Language Engineering, Department of Linguistics, Uppsala University* 15.

——, MAGNUS MERKEL, ANNA SÅGVALL HEIN, & JÖRG TIEDEMANN. 2000. Evaluation of word alignment systems. In *Proceedings of the Second International Conference on Linguistic Resources and Evaluation (LREC-2000)*, 1255–1261, Athens, Greece.

AKIBA, YASUHIRO, KENJI IMAMURA, & EIICHIRO SUMITA. 2001. Using multiple edit distances to automatically rank machine translation output. In *Proceedings of the MT Summit VIII*, Santiago de Compostela, Spain.

AL-ONAIZAN, YASER, JAN CURIN, MICHAEL JAHR, KEVIN KNIGHT, JOHN D. LAFFERTY, I. DAN MELAMED, FRANZ-JOSEF OCH, DAVID PURDY, NOAH A. SMITH, & DAVID YAROWSKY. 1999. Statistical machine translation, final report, JHU workshop. (available at http://www.clsp.jhu.edu/ws99/).

ALSHAWI, HIYAN, SRINIVAS BANGALORE, & SHONA DOUGLAS. 1998. Learning phrase-base head transduction models for translation of spoken utterances. In *Proceedings of the fifth International Conference on Spoken Language Processing (ICSLP98*, 2767–2770, Sydney.

——, ——, & ——. 2000. Learning dependency translation models as collections of finite-state head transducers. *Computational Linguistics* 26.45–60.

——, & ADAM BUCHSBAUM. 1997. A comparison of head transducers and transfer for a limited domain translation application. In *Proceedings of the 35th Annual Meeting of the ACL*, 360–365, Madrid. ACL.

——, & Shona Douglas. 2000. Learning dependency transduction models from unannotated examples. *Philosophical Transactions of the Royal Society, Series A: Mathematical, Physical, and Engineering Sciences* 358.1357–1372.

Amengual, Juan Carlos, Asunción Castaño, Antonio Castellanos, Victor Manuel Jiménez, David Llorens, Andrés Marzal, Federico Prat, Juan Miguel Vilar, José Miguel Benedí, Francisco Casacuberta, Moisés Pastor, & Enrique Vidal. 2000. The EuTrans-I spoken language translation system. *Machine Translation* 15.75–103.

——, & Enrique Vidal. 1996. Two different approches for cost-efficient viterbi parsing with error correction. In *Advances in Structural and Syntactical Pattern Recognition: 6th International Workshop, SSPR '96*, ed. by Petra Perner, Patrick Wang, & Azriel Rosenfeld, 30–39. Berlin: Springer.

Arnold, D., L. Balkan, R. Lee Humphreys, S. Meijer, & L. Sadler. 1994. *Machine Translation: An Introductory Guide*. Manchester, UK: NCC Blackwell.

Bangalore, Srinivas, & Giuseppe Riccardi. 2001. A finite-state approach to machine translation. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics*, 135–142, Pittsburgh, PA.

Baum, L.E. 1972. An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process. In *Inequalities 3: Proceedings of the 3^{rd} Symposium on Inequalities*, 1–8, Los Angeles, CA. Academic Press.

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

——, Stephen A. Della Pietra, & Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics* 22.37–71.

Booth, A. Donald, & William N. Locke. 1955. Historical introduction. In *Machine Translation of Languages*, ed. by William N. Locke & A. Donald Booth, 1–14. Cambridge, MA: The Technology Press of MIT.

BRANTS, THORSTEN. 2000. TnT – a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000)*, Seattle, WA.

BREW, CHRIS, & HENRY S. THOMPSON. 1994. Automatic evaluation of computer generated text: A progress report on the TextEval project. In *Human Language Technology: Proceedings of the Workshop*, ed. by C. Weinstein, 108–113. ARPA/ISTO.

BROWN, PETER F., STEPHEN A. DELLA PIETRA, VINCENT J. DELLA PIETRA, & ROBERT L. MERCER. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics* 19.263–311.

BROWN, RALF D. 1997. Automated dictionary extraction for "knowledge-free" example-based translation. In *Proceedings from the Seventh International Conference on Theoretical and Methodological Issues in Machine Translation, TMI '97*, 111–118, Santa Fe, NM.

—— 1999. Adding linguistic knowledge to a lexical example-based translation system. In *Proceedings from the Eighth International Conference on Theoretical and Methodological Issues in Machine Translation, TMI '99*, 22–32, Chester, UK.

—— 2000. Automated generalization of translation examples. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING-2000)*, 125–131, Saarbrücken, Germany.

CASTELLANOS, ANTONIO, ISABEL GALIANO, & ENRIQUE VIDAL. 1994. Application of OSTIA to machine translation tasks. In *Grammatical Inference and Applications: Second International Colloquium, ICGI-94*, ed. by Rafael C. Carrasco & José Oncina, 93–105. Berlin: Springer-Verlag.

CHEN, STANLEY F. 1993. Aligning Sentences in Bilingual Corpora Using Lexical Information. In *Proceedings of the 31st Annual Meeting of the ACL*, 9–16, Columbus, Ohio.

CORSTON-OLIVER, SIMON, MICHAEL GAMON, & CHRIS BROCKETT. 2001. A machine learing approach to the automatic evaluation of machine translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, 140–147, Toulouse, France.

DUNNING, TED. 1993. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics* 19.61–74.

Gale, William A., & Kenneth W. Church. 1993. A program for aligning sentences in bilingual corpora. *Computational Linguistics* 19.75–102.

Germann, Ulrich, Michael Jahr, Kevin Knight, Daniel Marcu, & Kenji Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, 228–235, Toulouse, France.

Gold, E. Mark. 1967. Language identification in the limit. *Information and Control* 10.447–474.

Grefenstette, Gregory. 1999. The world wide web as a resource for example-based machine translation tasks. In *Translating and the Computer 21: Proceedings from the 21st International Conference on Translation and the Computer*, London. ASLIB.

Gross, Alex. 1992. Limitations of computers as translation tools. In *Computers in Translation: A Practical Appraisal*, ed. by John Newton, 96–130. London: Routledge.

Hovy, Eduard. 1999. Toward finely differentiated evaluation metrics for machine translation. In *Proceedings of the EAGLES Workshop on Standards and Evaluation*, 127–133, Pisa, Italy. draft of January 1999.

Hutchins, W. John. 1986. *Machine Translation: Past, Present, Future*. Chichester, UK: Ellis Horwood Limited.

—— (ed.) 2000a. *Early Years in Machine Translation*, volume 97 of *Studies in the History of the Language Sciences*. Amsterdam: John Benjamins Publishing Co.

——. 2000b. The first decades of machine translation. In *Early Years in Machine Translation*, ed. by W. John Hutchins, volume 97 of *Studies in the History of the Language Sciences*, 1–15. Amsterdam: John Benjamins Publishing Co.

——. 2000c. Yehoshua Bar-Hillel: A philospher's contribution to machine translation. In *Early Years in Machine Translation*, ed. by W. John Hutchins, volume 97 of *Studies in the History of the Language Sciences*, 299–312. Amsterdam: John Benjamins Publishing Co.

——, & Harold L. Somers. 1992. *An Introduction to Machine Translation*. London, UK: Academic Press Limited.

JOHNSTON, MICHAEL. 1998. Unification-based multimodal parsing. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, 624–630, Montreal, Quebec, Canada.

JONES, DANIEL. 1996. *Analogical Natural Language Processing*. London, England: UCL Press.

KALLMEYER, LAURA. 2000. A query tool for syntactically annotated corpora. In *Proceedings of the Joint SIGDAT conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 190–198, Honk Kong.

KAY, MARTIN. 1982. Machine translation. *American Journal of Computational Linguistics* 8.74–78.

——. 1997a. It's still the proper place. *Machine Translation* 12.35–38.

——. 1997b. The proper place of men and machines in language translation. *Machine Translation* 12.3–23. reprint of 1980 Xerox PARC working paper.

——, JEAN MARK GAWRON, & PETER NORVIG. 1994. *Verbmobil: A Translation System for Face-to-Face Dialog*. CSLI Lecture Notes Number 33. Stanford, CA: CSLI.

KING, MARGARET. 1997. Evaluating translation. In *Machine Translation and Translation Theory*, ed. by Christa Hauenschild & Susanne Heizmann, 251–263. Berlin: Mouton de Gruyter.

KNIGHT, KEVIN. 1997. Automating knowledge acquisition for machine translation. *AI Magazine* 18.81–96.

——. 1999. A statistical machine translation tutorial workbook. (available at http://www.isi.edu/natural-language/mt/wkbk.rtf).

——, & YASER AL-ONAIZAN. 1998. Translation with finite-state devices. In *Machine Translation and the Information Soup: Proceedings of the Third Conference of the Association of Machine Translation in the Americas, AMTA '98*, ed. by David Farwell, Laurie Gerber, & Eduard Hovy. Berlin: Springer-Verlag.

KRUSKAL, JOSEPH. 1999. An overview of sequence comparison. In *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, ed. by David Sankoff & Joseph Kruskal, 1–44. Stanford: CSLI Publications, Reissued edition with an introduction by John Nerbonne.

Macklovitch, Elliott, & Marie-Louise Hannan. 1998. Line 'em up: Advances in alignment technology and their impact on translation support tools. *Machine Translation* 13.41–57.

Manning, Christopher D., & Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

Marcu, Daniel. 2001. Towards a unified approach to memory- and statistical-based machine translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, 378–385, Toulouse, France.

McTait, Kevin. 2001. Linguistic knowledge and complexity in an EBMT system based on translation patterns. In *Proceedings of the MT Summit VIII*, Santiago de Compostela, Spain.

Melamed, I. Dan. 1998. Manual annotation of translational equivalence: The Blinker project. Technical Report #98-07, IRCS, University of Pennsylvania.

——. 2001. *Empirical Methods for Exploiting Parallel Texts*. Cambridge, MA: MIT Press.

Melby, Alan. 1997. Some notes on the proper place of men and machines in language translation. *Machine Translation* 12.29–34.

Mel'čuk, Igor A. 1988. *Dependency Syntax: Theory and Practice*. Albany, NY: State University of New York Press.

—— 2000. Machine translation and formal linguistics in the USSR. In *Early Years in Machine Translation*, ed. by W. John Hutchins, volume 97 of *Studies in the History of the Language Sciences*, 205–226. Amsterdam: John Benjamins Publishing Co.

Müller, Stefan. to appear. Continuous or discontinuous constituents? A comparison between syntactic analyses for constituent order and their processing systems. *Language and Computation* .

Nagao, Makoto. 1984. A framework of a mechanical translation between Japanese and English by analogy principle. In *Artificial and Human Intelligence: Edited Review Papers presented at the International NATO Symposium on Artificial and Human Intelligence, held in Lyon, France, October, 1981*, ed. by A. Elithorn & R. Banerji, 173–180. Amsterdam: North-Holland.

NIESSEN, SONJA, FRANZ JOSEF OCH, GREGOR LEUSCH, & HERMANN NEY. 2000. An evaluation tool for machine translation: Fast evaluation for MT research. In *Proceedings of the 2nd International Conference of Language Resources and Evaluation*, 39–45, Athens, Greece.

NIRENBURG, SERGEI, CONSTANTINE DOMASHNEV, & DEAN J. GRANNES. 1993. Two approaches to matching in example-based machine translation. In *Fifth International Conference on Theoretical and Methodological Issues in Machine Translation, TMI '93: MT in the next generation*, 47–57, Kyoto, Japan.

OCH, FRANZ JOSEF, & HERMANN NEY. 2000. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting of the ACL*, 440–447, Hong Kong, China.

OETTINGER, ANTHONY G. 2000. Machine translation at harvard. In *Early Years in Machine Translation*, ed. by W. John Hutchins, volume 97 of *Studies in the History of the Language Sciences*, 73–86. Amsterdam: John Benjamins Publishing Co.

ONCINA, JOSÉ, PEDRO GARCÍA, & ENRIQUE VIDAL. 1993. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.448–458.

——, & MIGUEL ANGEL VARÓ. 1996. Using domain information during the learning of a subsequential transducer. In *Grammatical Inference: Learning Syntax from Sentences: Third International Colloquium, ICGI-96*, ed. by Laurent Miclet & Colin de la Higuera, 301–312. Berlin: Springer.

PEREIRA, FERNANDO C. N., & REBECCA N. WRIGHT. 1997. Finite-state approximation of phrase-structure grammars. In *Finite-State Language Processing*, ed. by Emmanuel Roche & Yves Schabes, 149–173. Cambridge, MA: MIT Press.

PIERCE, JOHN R., JOHN B. CARROLL, ERIC. P. HAMP, DAVID G. HAYS, CHARLES F. HOCKETT, ANTHONY G. OETTINGER, & ALAN PERLIS. 1966. *Language and Machines: Computers in Translation and Linguistics: A Report by the Automatic Language Processing Advisory Committee (ALPAC), Divison of Behavioral Sciences, National Research Council, Publication 1416*. Washington, DC: National Academy of Sciences.

RHODES, IDA. 1967. The importance of the glossary storage in machine translation. In *Machine Translation*, ed. by A. D. Booth, 429–449. Amsterdam: North-Holland Publishing Co.

Roche, Emmanuel, & Yves Schabes. 1997. Introduction. In *Finite-State Language Processing*, ed. by Emmanuel Roche & Yves Schabes, 1–65. Cambridge, MA: MIT Press.

Rosetta, M.T. 1994. *Compositional Translation*. Dordrecht, The Netherlands: Kluwer.

Sanchis, Alberto, David Picó, Joan Miquel del Val, Ferran Fabregat, Jesús Tomás, Moisés Pastor, Francisco Casacuberta, & Enrique Vidal. 2001. A morphological analyser for machine translation based on finite-state transducers. In *Proceedings of the MT Summit VIII*, Santiago de Compostela, Spain.

Sato, Satoshi, & Makoto Nagao. 1990. Toward memory-based translation. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*, 247–252, Helsinki, Finland.

Schiehlen, Michael. 1998. Learning tense translation from bilingual corpora. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, 1183–1187, Montreal, Quebec, Canada.

Shannon, Claude E. 1948. A mathematical theory of communication. *Bell System Technical Journal* 27.379–423,623–656.

Somers, Harold L. 1999. Review article: Example-based machine translation. *Machine Translation* 14.113–157.

Vanni, Michelle, & Florence Reeder. 2000. How are you doing? a look at MT evaluation. In *Envisioning Machine Translation in the Information Future: 4th Conference of the Association for Machine Translation in the Americas, AMTA 2000*, ed. by John S. White, Lecture Notes in Artificial Intelligence #1934, 109–116. Berlin: Springer.

Vasconcellos, Muriel. 2000. The Georgetown project and Léon Dostert: Recollections of a young assistant. In *Early Years in Machine Translation*, ed. by W. John Hutchins, volume 97 of *Studies in the History of the Language Sciences*, 87–96. Amsterdam: John Benjamins Publishing Co.

Veale, Tony, & Andy Way. 1997. Gaijin: A bootstrapping, template-drive approach to example-based MT. In *Proceedings of NeMNLP '97: New Methods in Natural Language Processing*, Sofia, Bulgaria.

Vilar, Juan Miguel, Victor Manuel Jiménez, Juan Carlos Amengual, Antonio Castellanos, David Llorens, & Enrique Vidal. 1999. Text and speech translation by means of subsequential transducers. In *Extended Finite State Models of Language*, ed. by Andras Kornai, 121–139. Cambridge: Cambridge University Press.

Vogel, Stephan, & Hermann Ney. 2000. Translation with cascaded finite state transducers. In *Proceedings of the 38th Annual Meeting of the ACL*, 23–30, Hong Kong, China.

——, Franz Josef Och, Christof Tillmann, Sonja Niessen, Hassan Sawaf, & Hermann Ney. 2000. Statistical methods for machine translation. In *Verbmobil: Foundations of Speech-to-Speech Translation*, ed. by Wolfgang Wahlster, 377–393. Berlin: Springer-Verlag.

Weaver, Warren. 1955. Translation. In *Machine Translation of Languages*, ed. by William N. Locke & A. Donald Booth, 15–23. Cambridge, MA: The Technology Press of MIT. Reprint of memorandum of July 15, 1949.

White, John S. 2000. Contemplating automatic MT evaluation. In *Envisioning Machine Translation in the Information Future: 4th Conference of the Association for Machine Translation in the Americas, AMTA 2000*, ed. by John S. White, Lecture Notes in Artificial Intelligence #1934, 100–108. Berlin: Springer.

——, & A. O'Connell, Theresa. 1993. Evaluation of machine translation. In *Proceedings of the 1993 ARPA Workshop on Human Language Technology*, 206–210, Princeton, NJ.

Wilks, Yorick. 1992. SYSTRAN: It obviously works but how much can it be improved? In *Computers in Translation: A Practical Appraisal*, ed. by John Newton, 166–188. London: Routledge.

Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics* 23.377–403.

Yamada, Kenji, & Kevin Knight. 2001. A syntax-based translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, 523–529, Toulouse, France.

Yngve, Victor H. 1955. Syntax and the problem of multiple meaning. In *Machine Translation of Languages*, ed. by William N. Locke & A. Donald Booth, 208–226. Cambridge, MA: The Technology Press of MIT.

—— 1967. Mt at M.I.T. 1965. In *Machine Translation*, ed. by A.D. Booth, 451–523. Amsterdam: North-Holland Publishing Company.

—— 2000. Early reseach at M.I.T.: In search of adequate theory. In *Early Years in Machine Translation*, ed. by W. John Hutchins, volume 97 of *Studies in the History of the Language Sciences*, 39–72. Amsterdam: John Benjamins Publishing Co.

Subject Index

Author Index