Exact and Approximate Methods for Machine Translation Decoding

Yin-Wen Chang

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2015

ABSTRACT

Exact and Approximate Methods for Machine Translation Decoding

Yin-Wen Chang

Statistical methods have been the major force driving the advance of machine translation in recent years. Complex models are designed to improve translation performance, but the added complexity also makes decoding more challenging. In this thesis, we focus on designing exact and approximate algorithms for machine translation decoding. More specifically, we will discuss the decoding problems for phrase-based translation models and bidirectional word alignment.

The techniques explored in this thesis are Lagrangian relaxation and local search. Lagrangian relaxation based algorithms give us exact methods that have formal guarantees while being efficient in practice. We study extensions to Lagrangian relaxation that improve the convergence rate on machine translation decoding problems. The extensions include a tightening technique that adds constraints incrementally, optimality-preserving pruning to manage the search space size and utilizing the bounding properties of Lagrangian relaxation to develop an exact beam search algorithm. In addition to having the potential to improve translation accuracy, exact decoding deepens our understanding of the model that we are using, since it separates model errors from optimization errors.

The observation leads to the question of designing models that improve the translation quality. We design a syntactic phrase-based model that incorporates a dependency language model to evaluate the fluency level of the target language. By employing local search, an approximate method, to decode this richer model, we discuss the trade-off between the complexity of a model and the decoding efficiency with the model.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor Michael Collins, for guiding me through my studies and research. I still remember the excitement from our first research conversation. This work would not be possible without him. I also want to thank my committee members David Blei, Tommi Jaakkola, Daniel Hsu, and Clifford Stein for their invaluable comments and feedback.

I also thank my long time officemate and collaborator Alexander Rush. From our discussion and coding sessions, I have enhanced my knowledge in natural language processing and sharpened my programming skills. I am fortunate to know my labmates Karl Stratos, Avner May, Andrei Simion, and Mohammad Sadegh Rasooli.

I am grateful to have the opportunities to work with Marius Pasca, Terry Koo, and Kuzman Ganchev as an intern in Google Research. The experience has broaden my perspective.

My gratitude also goes to my former advisor, Chih-Jen Lin, for introducing me to the area of machine learning, which applied well to natural language processing.

Thanks to my parents for their love and support, which makes me who I am today, and my sister for being a role model throughout my childhood. Finally, thanks to Peng-Jen for accompanying me through my journey abroad.

Chapter 1

Introduction

Machine translation is an important subfield in natural language processing (NLP). The goal of a machine translation system is to translate a document in one language into another language, allowing readers to understand content written in a language other than his or her own. The importance of machine translation has been signified by the availability and popularity of free online translation systems. However, producing translations of good quality remains a challenging problem.

Recent progress in machine translation has been powered by the advance of statistical methods, which, in turn, benefit from the wide availability of large scale *parallel corpora*. These corpora consist of example translations between two languages, and statistical machine translation systems extract translation rules automatically from these data sets.

Statistical methods involve the design of a model, estimating model parameters from the data, and making predictions on new data instances using the model. In the context of machine translation, the model determines what is a good translation according to certain mathematical formulae. The parameter estimation step learns the specific parameters of the mathematical formula to score a translation. In the prediction step, the system takes a sentence in one language as its input, and outputs a translation in another language. In this thesis, we will focus on the prediction step, which is also known as the decoding problem.

The final translation step is not the only decoding problem involved. A machine translation system is usually a pipeline consisting of many steps. One important building block of translation systems is word alignment. In a parallel corpus, we assume that the data is sentence-aligned, which means that the corresponding sentences in the two languages are aligned to each other. In the preparation for establishing a translation model, we want to align the words as well. The task of word alignment is to identify the matching words in different languages. Word alignment help us build a dictionary or a word translation table. The phrase translation table used in phrase-based models can also be extracted from word alignment results.

In general, the decoding problem can be formulated as a combinatorial optimization problem:

$$y^* = \arg\max_{y \in \mathcal{Y}} f(y)$$

We use y to represent any valid structure that we consider. In machine translation, y is a possible translation, and in word alignment, y is the alignment between words. The set $\mathcal{Y}$ is a set of all possible structures, and f is a scoring function of the structure, defined by the model. We aim to find y^*, the highest scoring structure under the model. Performing an exhaustive search over the set $\mathcal{Y}$ is often intractable, and a common solution is to use heuristic algorithms, such as beam search for phrase-based translation models. Beam search is efficient in practice and are used by many state-of-the-art systems. However, there is no formal guarantee on the quality of the solution in terms of the model score.

In this thesis, we focus on two decoding problems in machine translation. One is decoding of phrase-based translation models. The other is finding bidirectional word alignments. Both problems have been proved to be NP-hard if there are no further restrictions. We will design both exact algorithms and approximate algorithms for tackling these problems. The exact algorithms apply Lagrangian relaxation, a classical technique in combinatorial optimization, while the approximate algorithm is based on local search.

In the remainder of this introduction, we first introduce the two decoding problems that this

thesis focuses on: decoding with phrase-based models and the bidirectional word alignment problem. We then describe the techniques we apply to these two problems: Lagrangian relaxation and local search.

1.1 Decoding Problems

The two decoding problems we will focus on are the phrase-based decoding problem and the bidirectional word alignment problem. Phrase-based models are widely used in state-of-the-art machine translation systems and decoding is the final step toward the goal of a translation system. The aim of the decoding problem is to find the highest scoring translation under a model for an input sentence.

An important component of a phrase-based model is the phrase translation table, which is used to evaluate the translation from one phrase to another. One crucial step to build the phrase translation table is to find word alignments, which defines the set of possible correspondences between multi-word expressions in two languages. Bidirectional word alignment is the task of finding such alignment that are symmetric – that there is no distinction between directions. Either aligning from language A to language B or reverse will give the same result.

1.1.1 Phrase-Based Translation Models

Currently, many state-of-the-art statistical machine translation systems are based on phrase-based translation models (Och et al., 1999b). In phrase-based models, the basic unit is a sequence of words. The span of words is called a phrase and it does not have to correspond to any linguistic or semantic meaning. Phrase-based models are proposed to improve word-based models, since words are not always the best basic units for translation. In many cases it is useful to have lexical entries pairing a multi-word sequence in one language with a multi-word sequence in another language. For example, in translation from German to English, we might have a lexical entry pairing "die regionen in äußerster randlage" with "the outermost regions".

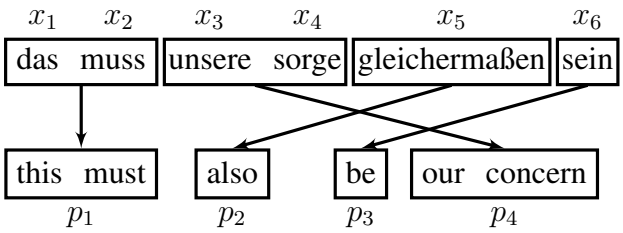


Figure 1-1: An example of phrase-based translation decoding. The source-language sentence is a German sentence: das muss unsere sorge gleichermaßen sein. One of the possible translations in English is this must also be our concern. The arrows show the mapping from German phrases to English phrases.

German phrase	English translation	score
unsere sorge	's concern	0.04
	concern	0.03
	is our concern ,	0.05
	my concern	0.01
	our concern	0.17
	our worries	0.08

Table 1.1: Example entries in the phrase translation table.

Figure 1-1 gives an example of translating a German sentence into English using a phrase-based translation model. The **source sentence** in German is segmented into phrases. Each German phrase can be translated into an English phrase, chosen from a list of candidate phrases. The translated English phrases can be reordered to form a more fluent English sentence. In the example, the verb "sein" in the German sentence is placed in the end of the sentence, while in the English sentence, the verb "be" is moved to appear directly before the phrase "our concern".

There may be several candidate translations of one German phrase. In the above example, "unsere sorge" may be translated into English phrases other than "our concern". Table 1.1 shows other possible translations, with their assigned score. The candidate translations and their scores can be looked up in a **phrase translation table**.

More formally, we define the mapping between German phrase and English phrase to be a phrase-pair p. A phrase-pair p can be represented by a tuple (s, t, e), where s and t marks the start and end of the span in the German sentence, and e is an English phrase that the German phrase can be translated into. In the example, (1, 2, this must) represents the mapping between "das

muss" and "this must". We define a derivation to be a translation and the mappings between the translated sentence and the source sentence. A derivation can be represented by a sequence of phrase-pairs. The derivation of the above example is (1, 2, this must), (5, 5, also), (6, 6, be), (3, 4, our concern).

In order to find a good translation, we need to define a scoring function that would give higher scores to better translations. In a phrase-based model, the major components include the language model, the translation model, and the distortion model. The language model is used to encourage the translation to be in good English. It is usually an n-gram language model. The translation model is represented by the phrase translation table. It makes sure that each English phrase is a suitable translation of the input German phrase it is translated from. The distortion model is usually distance-based and discourages the phrases to move too further away from their original order in the German sentence.

The scoring function f can be formally stated as follows:

$$f(y) = \mu_t \sum_{l=1}^{L} g_t(p_l) + \mu_l g_l(e(y)) + \mu_\delta \sum_{l=1}^{L-1} g_\delta(t(p_l), s(p_{l+1}))$$

where y is a derivation and f is the scoring function. We use $s(p)$, $t(p)$ and $e(p)$ to refer to each element in the tuples of the phrase-pair p. The translated English sentence given by a derivation y is $e(y)$. The derivation y consists of L phrase-pairs: $y = \langle p_1 p_2 \dots p_L \rangle$. The function g_t represents the translation model, while g_l is the language model and g_δ is the distortion model. The weights μ_t, μ_l, and μ_δ are for each component of the model.

The goal of decoding is to find the highest scoring translation of a given input German sentence using the model:

$$y^* = \arg\max_{y \in \mathcal{Y}} f(y),$$

where $\mathcal{Y}$ is the set of valid translations. A common constraint is that each German word should be translated exactly once. There are many ways we can segment the German sentence into phrases, several possible translations for each German phrase, and several ways to reorder the English

phrases. These many combinations make it hard to find the highest scoring translation of a given sentence. In general, the phrase-based decoding problem is an NP-complete problem (Knight, 1999).

Thus, many heuristic search methods are explored. One of the most commonly used methods is **beam search** (Koehn et al., 2003). Beam search is very efficient and usually produces good translations in practice. However, it does not provide any formal guarantee regarding how good a translation is compared to the optimal solution.

Exact decoding refers to methods that can find the highest scoring translation and gives a **certificate of optimality**. Some previous works have formulated the problem as an integer linear program and used off-the-shelf solvers (Germann et al., 2001; Riedel and Clarke, 2009). Other attempts include converting the decoding problem into a traveling salesman problem (TSP) and then applying a TSP solver (Zaslavskiy et al., 2009). However, these methods are usually not efficient enough to be applied to longer sentences.

1.1.2 Word Alignment

Word alignment is an important building block for many statistical machine translation systems. For example, in a phrase-based model, the phrase translation table is built by extracting phrase-pairs from a pair of word-aligned sentences. However, typical data sets have text that are sentence-aligned, but not word-aligned. Thus, in the alignment problem, the input is a pair of sentences that have the same meaning but are in two different languages. One is in the foreign language (the source-language), and the other is in English (the target-language). The goal of word alignment is to align the words in the foreign sentence to the words in the English sentence. Figure 1-2 and Figure 1-3 both illustrate word alignments between a French and an English sentence.

In this section we will assume that the source language is French and the target language is English, but the alignment problem can be applied to any pair of languages. Popular word alignment models include the IBM Models (Brown et al., 1993) and hidden Markov models for alignment (Vogel et al., 1996). We will introduce IBM Model 2 and the HMM-based model in this

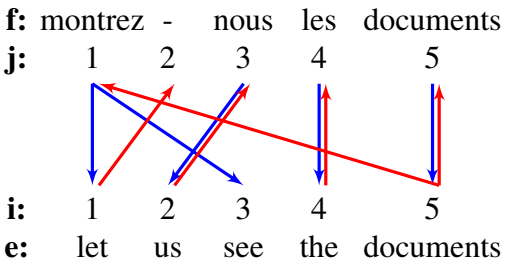
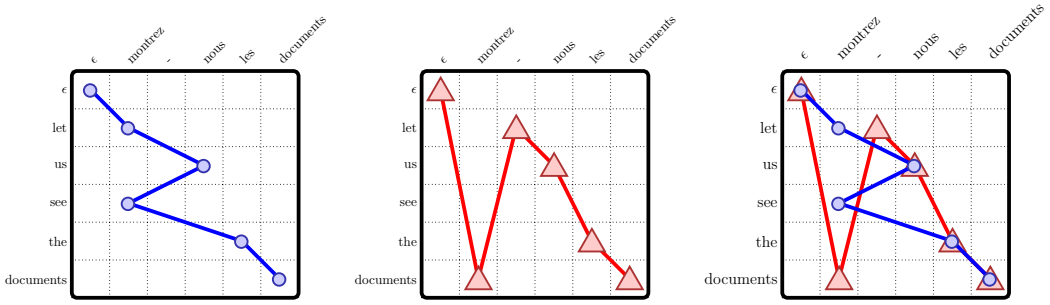


Figure 1-2: Alignment between French words and English words. This figure shows the alignment of both directions. Alignment $\mathbf{a}$ is represented by blue lines, which align English words to French words. The alignment vector is $\mathbf{a} = \langle 1, 3, 1, 4, 5 \rangle$.



(a) (b) (c)

Figure 1-3: Word alignment between a French and an English sentence can be visualized as a matrix. (a) Here both let and see are aligned to montrez, us is aligned to nous, the is aligned to les, and documents is aligned to documents. (b) Another direction. (c) The two directional alignments are not the same.

section. IBM Model 2 is a word-based translation model that captures the lexical translation and the absolute alignment model that reflects the reordering of the translated sentence. On the other hand, instead of using the absolute alignment model, the HMM-based word alignment model let the alignment probabilities to be based on the differences in the alignment positions.

To state the alignment problem more formally, let vector $\mathbf{a}$ denote an alignment, which is a mapping from an English word at position i to a French word at position j:

$$a_i = j.$$

An example English-to-French alignment vector $\mathbf{a} = \langle 1, 3, 1, 4, 5 \rangle$ is represented by blue lines in

Figure 1-2, and blue dots in Figure 1-3.

The alignment problem aims to model the probability $p(\mathbf{e}, \mathbf{a}|\mathbf{f})$ for a French sentence $\mathbf{f} = f_1 \dots f_m$ and an English sentence $\mathbf{e} = e_1 \dots e_l$. Note that m is the length of the French sentence and l is the length of the English sentence.

IBM Model 2 is improved upon IBM Model 1, which only models the lexical translation probability $p(e_i|f_j)$ between words. In IBM Model 2, the alignment probability $p(j|i, l, m)$ is introduced. The lexical translation probability and the absolute alignment probability are combined to form IBM Model 2:

$$p(\mathbf{e}, \mathbf{a}|\mathbf{f}) = \prod_{i=1}^{l} p(e_i|f_{a_i}) p(a_i|i, l, m)$$

We can estimate the parameters $p(e_i|f_{a_i})$ and $p(a_i|i, l, m)$ from a training corpus using the Expectation-Maximization (EM) algorithm. It is an iterative algorithm that uncovers the parameters from incomplete data. In IBM Model 2, the hidden variables are the alignment $\mathbf{a}$, and the observed data are the French and the English sentence.

Once we have estimated the probabilities from a training corpus, we can find the most probable alignment between a French sentence and an English sentence under the model.

$$\arg\max_{\mathbf{a}} p(\mathbf{a}|\mathbf{f}, \mathbf{e})$$

The following derivation shows how this can be done using the estimated parameters. First, we can derived the distribution:

$$p(\mathbf{e}|\mathbf{f}) = \sum_{\mathbf{a}} p(\mathbf{e}, \mathbf{a}|\mathbf{f}).$$

Then by chain rule, we will have

$$p(\mathbf{a}|\mathbf{f}, \mathbf{e}) = \frac{p(\mathbf{e}, \mathbf{a}|\mathbf{f})}{p(\mathbf{e}|\mathbf{f})}.$$

The probability $p(\mathbf{e}|\mathbf{f})$ remains the same for any alignment $\mathbf{a}$, when $\mathbf{e}$ and $\mathbf{f}$ are given. Hence,

$$\arg\max_{\mathbf{a}} p(\mathbf{a}|\mathbf{f}, \mathbf{e}) = \arg\max_{\mathbf{a}} p(\mathbf{e}, \mathbf{a}|\mathbf{f}).$$

The **HMM-based word alignment model** is very similar to IBM Model 2. The difference is that it models the alignment probability based on the distance between a_i and a_{i-1}. In an HMM, there are observed variables and hidden variables. We assume that the observed variables are generated by the hidden variables. In the word alignment case, the observed variables are the English words e_i and the hidden variables are the French words f_j. The alignment is determined by choosing the hidden French words f_j that generates the observed English words. The emission probability in HMM models the probability that a hidden variable generates an observed variable. In the alignment problem, it is the translation probability $p(e_i|f_{a_i})$. The transition probability in HMM models the probability that a hidden variable transitions from another hidden variable. It is the alignment probability $p(a_i|a_{i-1})$ in the alignment problem. Together, the emission probability and the transition probability forms the HMM:

$$p(\mathbf{e}, \mathbf{a}|\mathbf{f}) = \prod_{i=1}^{l} p(e_i|f_{a_i})p(a_i|a_{i-1}, m)$$

As for the IBM Model 2, EM algorithm can be applied to the HMM-based alignment model to estimate the parameters.

One problem of these models is that they are directional. Because each English word is aligned to at most one foreign word, we can not generate word alignments that map several English words to one foreign word. If we reverse the alignment direction to be from English to the foreign language, we will not be able to generate alignments that map several foreign words to an English word. We say that the alignments are not symmetric.

In order to handle the asymmetric problem, many systems run the directional aligners and then merge the two directional alignments heuristically. Naive methods include taking the intersection or the union. The intersection includes good alignment points, but tends to include less alignment points. The union includes most of the potential alignment points, but is more likely to include incorrect alignment points. More complex methods use other heuristics, such as grow-diag-final (Koehn et al., 2003), to explore the space between taking the intersection and the union. These

methods generally include all the alignment points given by the intersection and add some alignment points from the union heuristically.

Bidirectional models intend to find symmetric alignments directly, but decoding of many bidirectional models is NP-hard (see Appendix B.1). Previously DeNero and Macherey (2011) has proposed a bidirectional model and used dual decomposition to solve it. However, the number of cases where an exact solution is recovered is relatively low.

1.2 Techniques

The two decoding problems that are considered by this thesis are NP-hard problems if there are no further constraints. In this section, we will introduce two techniques that will be used in this thesis. The first one is Lagrangian relaxation, which is a general technique in combinatorial optimization. The Lagrangian relaxation algorithm gives us exact solutions when it converges. The second one is local search, which is a powerful tool when the model is too complex and the search space too huge to be searched exhaustively. The discussions of applying these two methods will demonstrate the trade-offs when designing a machine translation system. The trade-offs are between having a rich model, finding an optimal solution, and finding a solution efficiently. The final goal of all systems is to deliver good quality translations, and exact algorithms let us better understand the model, since we are able to separate the model error and the optimization error.

1.2.1 Lagrangian Relaxation

Lagrangian relaxation is a classical technique in combinatorial optimization for solving hard problems. The idea is to decompose the problem into a relaxed problem and some side constraints. The relaxed problem can be solved efficiently, while adding the side constraints will make the problem too complicated to be solved directly. Lagrangian relaxation incorporates the side constraints as soft constraints and encourages the constraints to be satisfied by Lagrange multipliers. The algorithm has the ability to determine if a solution is optimal, which will be described as having an

optimality certificate. When a non-optimal solution is returned, the algorithm may provide a bound on the difference between the returned solution and the optimal solution.

Lagrangian relaxation was first discussed by Held and Karp (1971), who focused on relaxation algorithms for the traveling salesman problem (TSP). It was recently introduced into the NLP field (Rush et al., 2010; Koo et al., 2010). We will discuss several algorithms for solving decoding problems in machine translation based on Lagrangian relaxation. A decoding problem can be formulated as follows:

$$\arg\max_{y \in \mathcal{Y}} f(y) = \arg\max_{y \in \mathcal{Y}} \theta^T y, \tag{1.1}$$

where $y \in \{0,1\}^{|\mathcal{I}|}$ is an indicator vector representing a structure in the decoding problem. The structure could be a translation derivation or a word alignment. The set $\mathcal{Y}$ is a collection of all valid structures under some constraints. Solving Equation 1.1 directly might be intractable. For example, the bidirectional word alignment formulation we discuss in this thesis is proved to be NP-hard (Appendix B.1).

The strategy is to define a new set $\mathcal{Y}'$ that contains the solutions to a relaxed problem that can be solved efficiently. The set $\mathcal{Y}'$ contains all structures in the original set $\mathcal{Y}$ and other structures that are not considered valid in the original decoding problem. Therefore, we have $\mathcal{Y} \subset \mathcal{Y}'$. Since the set $\mathcal{Y}'$ is derived by relaxing some constraints, we can state the set $\mathcal{Y}$ as follows:

$$\mathcal{Y} = \{y : y \in \mathcal{Y}' \text{ and } Ay = b\},$$

where $A \in \mathbb{R}^{N \times |\mathcal{I}|}$ is the constraint matrix and $b \in \mathbb{R}^N$ is an vector. Note that N is the number of constraints. Together, $Ay = b$ defines the side constraints that we have relaxed and that we would like to re-introduce using Lagrange multipliers.

We assign one Lagrange multiplier for each constraint. The Lagrange multipliers can be represented as a vector of real values that has the same length as the number of constraints: $\lambda \in \mathbb{R}^N$.

The Lagrangian can be defined as:

$$L(\lambda, y) = \theta^T y + \lambda^T (Ay - b)$$

The dual objective is

$$L(\lambda) = \max_{y \in \mathcal{Y}'} L(\lambda, y).$$

Note that here we optimize over the new relaxed set $\mathcal{Y}'$.

The dual problem is to find the minimum of the Lagrangian dual over the λ:

$$\min_{\lambda \in \mathbb{R}^N} L(\lambda).$$

The dual problem can be solved by using a subgradient algorithm, since $L(\lambda)$ is a convex function. If we define

$$y_\lambda = \arg\max_{y \in \mathcal{Y}'} L(\lambda, y),$$

then

$$\gamma_\lambda = Ay_\lambda - b$$

is a subgradient of $L(\lambda)$ at λ. The subgradient method is an iterative method for minimizing $L(\lambda)$. At each iteration, it performs the update

$$\lambda^t \leftarrow \lambda^{t-1} - \alpha^t \gamma_{\lambda^{t-1}},$$

where $\alpha^t > 0$ is the step size for the t'th iteration.

When we find a value for λ such that the solution y_λ satisfies all the constraints, it is guaranteed to be the optimal solution. However, the algorithm is not guaranteed to converge. In practice, we can use heuristics to produce an approximate solution or use other techniques to encourage convergence.

```
Initialization: a complete state y
for t = 1 ... T
  s = f(y)
  for y' ∈ N(y)
    if f(y') > f(y)
      y ← y'
  if f(y) = s
    break
```

Figure 1-4: A local search algorithm with hill-climbing strategy.

1.2.2 Local Search

Local search is a type of heuristic optimization algorithm. It is heuristic since it does not perform a systematic or exhaustive search, and the optimal solution is not guaranteed. In practice, it often finds a reasonable solution when the search space is huge and exhaustive search is infeasible.

Local search operates on a complete current state, and moves only to neighbors of that state. An essential part of a local search algorithm is defining the local steps and, thus, the neighborhood that can be reached by taking one local step. Let y be a complete state. A neighborhood $\mathcal{N}(y)$ is a set of states that can be reached by state y by taking a local step. There are several variants of the local search algorithm. When using a hill-climbing strategy, it only moves to state with a higher score. Figure 1-4 shows a local search algorithm with hill-climbing strategy, where f is the scoring function that maps the state y to a value. The algorithm terminates when it reaches a state where there is no neighboring state with higher score. Thus, the drawback of such strategy is that it might be stuck in a local optimum.

Other strategies might include randomness. For example, in simulated annealing, a neighboring state is generated randomly and a state with lower score is accepted with a probability that depends on the scores, while a state with higher score is always accepted.

Local search has been successfully applied in the field of artificial intelligence, bioinformatics and other engineering areas. These problems includes the travelling salesman problem and vertex cover problem. Recently, various NLP problems consider local search as the optimization method,

including decoding of the phrase-based translation model (Langlais et al., 2007), dependency parsing (Zhang et al., 2014), translation alignment (Marcu and Wong, 2002), and joing inference for segmentation (Marcu and Wong, 2002), POS tagging and dependency parsing (Zhang et al., 2015).

1.3 Outline of the Thesis

This outline summarizes the chapters in this thesis and the contributions of this thesis. The first contribution is to design exact algorithms based on Lagrangian relaxation for these two decoding problems. Since the two decoding problems discussed in this thesis are complex, plain Lagrangian relaxation algorithms only converge on a small fraction of examples. This observation indicates that the relaxations are not tight. One of the main focuses of this thesis is to design extensions to Lagrangian relaxation algorithms to promote convergence. Exact algorithms are interesting algorithmically and helps us better understand the models. The last part of the contribution of this thesis is to design a richer model that improves the translation quality.

Chapter 2 gives an exact algorithm for decoding the phrase-based translation model. The algorithm is based on Lagrangian relaxation. By relaxing the constraint that each foreign word should be translated exactly once, we obtain a relaxed problem that can be solved efficiently by dynamic programming. Then the relaxed constraints are re-introduced and enforced by Lagrangian relaxation. The algorithm also introduces an extension that utilizes a tightening technique to obtain an exact solution when the relaxation is not tight. The tightening technique adds constraints incrementally, at each step picking the constraints that are violated the most.

Chapter 3 gives another exact algorithm for decoding the phrase-based translation model. The algorithm combines beam search and Lagrangian relaxation to produce an efficient exact algorithm. It is able to produce an exact variant of the heuristic beam search because the application of the upper bound given by Lagrangian relaxation. Without adding constraints and growing the search space of the relaxed problem, this algorithm is able to find exact solutions more efficiently than the one in Chapter 2, according to our experiments.

Chapter 4 turns from exact algorithms to heuristic algorithms to allow a richer model. This chapter introduces a decoding algorithm based on local search. The algorithm incorporates the syntactic information into the phrase-based translation model by using a dependency language model. In this chapter, we will discuss the trade-off between model complexity and the efficiency of the optimization method.

Chapter 5 switches focus to the bidirectional word alignment. It presents an Lagrangian relaxation based algorithm that solves a previously proposed model. The model was previously solved by dual decomposition, but our algorithm employs a different decomposition for applying a Lagrangian relaxation based algorithm. This algorithm also uses a tightening technique that incrementally adds constraints. Furthermore, it presents an optimality-preserving pruning technique to accompany the tightening procedure. The pruning method keeps the search space manageable after adding constraints, and therefore allows more constraints to be added.

Chapter 6 concludes the thesis and provides some ideas for future work.

Chapter 2

A Lagrangian Relaxation Algorithm with Tightening Techniques

[This chapter is adapted from joint work with Michael Collins entitled "Exact Decoding of Phrase-based Translation Models through Lagrangian Relaxation" (Chang and Collins, 2011).]

This chapter focuses on an algorithm for exact decoding of phrase-based translation models. The decoding problem is NP-hard when there is no further restriction on reordering. The algorithm is based on Lagrangian relaxation, and utilizes a tightening technique that incrementally adds constraints until an exact solution is found.

2.1 Introduction

Phrase-based models (Och et al., 1999b; Koehn et al., 2003, 2007) are a widely-used approach for statistical machine translation. The decoding problem for phrase-based models is NP-hard[1]; because of this, previous work has generally focused on approximate search methods, for example variants of beam search, for decoding.

This chapter describes an algorithm for exact decoding of phrase-based models, based on La-

[1]We refer here to the phrase-based models of Koehn et al. (2003, 2007), considered in this chapter. Other variants of phrase-based models, which allow polynomial time decoding, have been proposed, see the related work section.

grangian relaxation (Lemaréchal, 2001). The core of the algorithm is a dynamic program for phrase-based translation which is efficient, but which allows some ill-formed translations. More specifically, the dynamic program searches over the space of translations where exactly N words are translated (N is the number of words in the source-language sentence), but where some source-language words may be translated zero times, or some source-language words may be translated more than once. Lagrangian relaxation is used to enforce the constraint that each source-language word should be translated exactly once. A subgradient algorithm is used to optimize the dual problem arising from the relaxation.

The first technical contribution of this chapter is the basic Lagrangian relaxation algorithm. By the usual guarantees for Lagrangian relaxation, if this algorithm converges to a solution where all constraints are satisfied (i.e., where each word is translated exactly once), then the solution is guaranteed to be optimal. For some source-language sentences however, the underlying relaxation is loose, and the algorithm will not converge. The second technical contribution of this chapter is a method that incrementally adds constraints to the underlying dynamic program, thereby tightening the relaxation until an exact solution is recovered.

We describe experiments on translation from German to English, using phrase-based models trained by MOSES (Koehn et al., 2007). The method recovers exact solutions, with certificates of optimality, on over 99% of test examples. On over 78% of examples, the method converges with zero added constraints (i.e., using the basic algorithm); 99.67% of all examples converge with 9 or fewer constraints. We compare to a linear programming (LP)/integer linear programming (ILP) based decoder. Our method is much more efficient: LP or ILP decoding is not feasible for anything other than short sentences,[2] whereas the average decoding time for our method (for sentences of length 1-50 words) is 121 seconds per sentence. We also compare our method to MOSES, and give precise estimates of the number and magnitude of search errors that MOSES makes. Even with large beam sizes, MOSES makes a significant number of search errors. As far as we are aware, previous work has not successfully recovered exact solutions for the type of phrase-based models

[2]For example ILP decoding for sentences of lengths 11-15 words takes on average 2707.8 seconds.

used in MOSES.

2.2 Related Work

Lagrangian relaxation is a classical technique for solving combinatorial optimization problems (Korte and Vygen, 2008; Lemaréchal, 2001). Dual decomposition, a special case of Lagrangian relaxation, has been applied to inference problems in NLP (Koo et al., 2010; Rush et al., 2010), and also to Markov random fields (Wainwright et al., 2005; Komodakis et al., 2007; Sontag et al., 2008). Earlier work on belief propagation (Smith and Eisner, 2008) is closely related to dual decomposition. Recently, Rush and Collins (2011) describe a Lagrangian relaxation algorithm for decoding for syntactic translation; the algorithmic construction described in the current chapter is, however, very different in nature to this work.

Beam search stack decoders (Koehn et al., 2003) are the most commonly used decoding algorithm for phrase-based models. Dynamic-programming-based beam search algorithms are discussed for both word-based and phrase-based models by Tillmann and Ney (2003) and Tillmann (2006).

Several works attempt exact decoding, but efficiency remains an issue. Exact decoding via integer linear programming (ILP) for IBM model 4 (Brown et al., 1993) has been studied by Germann et al. (2001), with experiments using a bigram language model for sentences up to eight words in length. Riedel and Clarke (2009) have improved the efficiency of this work by using a cutting-plane algorithm, and experimented with sentence lengths up to 30 words (again with a bigram LM). Zaslavskiy et al. (2009) formulate phrase-based decoding problem as a traveling salesman problem (TSP), and take advantage of existing exact and approximate approaches designed for TSP. Their translation experiment uses a bigram language model and applies an approximate algorithm for TSP. Och et al. (2001) propose an A* search algorithm for IBM model 4, and test on sentence lengths up to 14 words. Other work (Kumar and Byrne, 2005a; Blackwood et al., 2009) has considered variants of phrase-based models with restrictions on reordering that allow exact, polynomial time decoding, using finite-state transducers.

The idea of incrementally adding constraints to tighten a relaxation until it is exact is a core idea in combinatorial optimization. Previous work on this topic in NLP or machine learning includes work on inference in Markov random fields (Sontag et al., 2008); work that encodes constraints using finite-state machines (Tromble and Eisner, 2006); and work on non-projective dependency parsing (Riedel and Clarke, 2006).

2.3 The Phrase-based Translation Model

This section establishes notation for phrase-based translation models, and gives a definition of the decoding problem. The phrase-based model we use is the same as that described by Koehn et al. (2003), as implemented in MOSES (Koehn et al., 2007).

The input to a phrase-based translation system is a source-language sentence with N words, $x_1 x_2 \dots x_N$. A *phrase table* is used to define the set of possible *phrases* for the sentence: each phrase is a tuple $p = (s, t, e)$, where (s, t) are indices representing a contiguous span in the source-language sentence (we have $s \leq t$), and e is a target-language string consisting of a sequence of target-language words. For example, the phrase $p = (2, 5, \textit{the dog})$ would specify that words $x_2 \dots x_5$ have a translation in the phrase table as "the dog". Each phrase p has a score $g(p) = g(s, t, e)$: this score will typically be calculated as a log-linear combination of features (e.g., see Koehn et al. (2003)).

We use $s(p)$, $t(p)$ and $e(p)$ to refer to the three components (s, t, e) of a phrase p.

The output from a phrase-based model is a sequence of phrases $y = \langle p_1 p_2 \dots p_L \rangle$. We will often refer to an output y as a *derivation*. The derivation y defines a target-language translation $e(y)$, which is formed by concatenating the strings $e(p_1), e(p_2), \dots, e(p_L)$. For two consecutive phrases $p_k = (s, t, e)$ and $p_{k+1} = (s', t', e')$, the *distortion distance* is defined as $\delta(t, s') = |t + 1 - s'|$. The score for a translation is then defined as

$$f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times \delta(t(p_k), s(p_{k+1}))$$

where $\eta \in \mathbb{R}$ is often referred to as the distortion penalty, and typically takes a negative value. The function $h(e(y))$ is the score of the string $e(y)$ under a language model.[3]

The decoding problem is to find

$$\arg\max_{y \in \mathcal{Y}} f(y)$$

where $\mathcal{Y}$ is the set of valid derivations. The set $\mathcal{Y}$ can be defined as follows. First, for any derivation $y = \langle p_1 p_2 \ldots p_L \rangle$, define $y(i)$ to be the number of times that the source-language word x_i has been translated in y: that is, $y(i) = \sum_{k=1}^{L} [[s(p_k) \leq i \leq t(p_k)]]$, where $[[\pi]] = 1$ if π is true, and 0 otherwise. Then $\mathcal{Y}$ is defined as the set of finite length sequences $\langle p_1 p_2 \ldots p_L \rangle$ such that:

1. Each word in the input is translated exactly once: that is, $y(i) = 1$ for $i = 1 \ldots N$.

2. For each pair of consecutive phrases p_k, p_{k+1} for $k = 1 \ldots L-1$, we have $\delta(t(p_k), s(p_{k+1})) \leq d$, where d is the *distortion limit*.

An exact dynamic programming algorithm for this problem uses states (w_1, w_2, b, r), where (w_1, w_2) is a target-language bigram that the partial translation ended with, b is a bit-string denoting which source-language words have been translated, and r is the end position of the previous phrase (e.g., see Koehn et al. (2003)). The bigram (w_1, w_2) is needed for calculation of trigram language model scores; r is needed to enforce the distortion limit, and to calculate distortion costs. The bit-string b is needed to ensure that each word is translated exactly once. Since the number of possible bit-strings is exponential in the length of sentence, exhaustive dynamic programming is in general intractable. Instead, people commonly use heuristic search methods such as beam search for decoding. However, these methods have no guarantee of returning the highest scoring translation.

[3]The language model score usually includes a word insertion score that controls the length of translations. The relative weights of the $g(p)$ and $h(e(y))$ terms, and the value for η, are typically chosen using MERT training (Och, 2003).

2.4 A Decoding Algorithm based on Lagrangian Relaxation

We now describe a decoding algorithm for phrase-based translation, based on Lagrangian relaxation. We first describe a dynamic program for decoding which is efficient, but which relaxes the $y(i) = 1$ constraints described in the previous section. We then describe the Lagrangian relaxation algorithm, which introduces Lagrange multipliers for each constraint of the form $y(i) = 1$, and uses a subgradient algorithm to minimize the dual arising from the relaxation. We conclude with theorems describing formal properties of the algorithm, and with an example run of the algorithm.

2.4.1 An Efficient Dynamic Program

As described in the previous section, our goal is to find the optimal translation $y^* = \arg\max_{y \in \mathcal{Y}} f(y)$. We will approach this problem by defining a set $\mathcal{Y}'$ such that $\mathcal{Y} \subset \mathcal{Y}'$, and such that

$$\arg\max_{y \in \mathcal{Y}'} f(y)$$

can be found efficiently using dynamic programming. The set $\mathcal{Y}'$ omits some constraints—specifically, the constraints that each source-language word is translated once, i.e., that $y(i) = 1$ for $i = 1 \ldots N$—that are enforced for members of $\mathcal{Y}$. In the next section we describe how to re-introduce these constraints using Lagrangian relaxation. The set $\mathcal{Y}'$ does, however, include a looser constraint, namely that $\sum_{i=1}^{N} y(i) = N$, which requires that exactly N words are translated.

We now give the dynamic program that defines $\mathcal{Y}'$. The main idea will be to replace bit-strings (as described in the previous section) by a much smaller number of dynamic programming states. Specifically, the states of the new dynamic program will be tuples (w_1, w_2, n, l, m, r). The pair (w_1, w_2) is again a target-language bigram corresponding to the last two words in the partial translation, and the integer r is again the end position of the previous phrase. The integer n is the number of words that have been translated thus far in the dynamic programming algorithm. The integers l and m specify a contiguous span $x_l \ldots x_m$ in the source-language sentence; this span is the last contiguous span of words that have been translated thus far.

The dynamic program can be viewed as a shortest-path problem in a directed graph, with nodes in the graph corresponding to states (w_1, w_2, n, l, m, r). The transitions in the graph are defined as follows. For each state (w_1, w_2, n, l, m, r), we consider any phrase $p = (s, t, e)$ with $e = (e_0 \dots e_{M-1} e_M)$ such that: 1) $\delta(r, s) \leq d$; and 2) $t < l$ or $s > m$. The former condition states that the phrase should satisfy the distortion limit. The latter condition requires that there is no overlap of the new phrase's span (s, t) with the span (l, m). For any such phrase, we create a transition

$$(w_1, w_2, n, l, m, r) \xrightarrow{p=(s,t,e)} (w_1', w_2', n', l', m', r')$$

where

- $(w_1', w_2') = \begin{cases} (e_{M-1}, e_M) & \text{if } M \geq 2 \\ (w_2, e_1) & \text{if } M = 1 \end{cases}$
- $n' = n + t - s + 1$
- $(l', m') = \begin{cases} (l,\ t\ ) & \text{if } s = m + 1 \\ (s, m) & \text{if } t = l - 1 \\ (s,\ t\ ) & \text{otherwise} \end{cases}$
- $r' = t$

The new target-language bigram (w_1', w_2') is the last two words of the partial translation after including phrase p. It comes from either the last two words of e, or, if e consists of a single word, the last word of the previous bigram, w_2, and the first and only word, e_1, in e. (l', m') is expanded from (l, m) if the spans (l, m) and (s, t) are adjacent. Otherwise, (l', m') will be the same as (s, t).

The score of the transition is given by a sum of the phrase translation score $g(p)$, the language model score, and the distortion cost $\eta \times \delta(r, s)$. The trigram language model score is $h(e_1|w_1, w_2) + h(e_2|w_2, e_1) + \sum_{i=1}^{M-2} h(e_{i+2}|e_i, e_{i+1})$, where $h(w_3|w_1, w_2)$ is a trigram score (typically a log probability plus a word insertion score).

We also include start and end states in the directed graph. The start state is (<s>, <s>, 0, 0, 0, 0) where <s> is the start symbol in the language model. For each state (w_1, w_2, n, l, m, r), such that

$n = N$, we create a transition to the end state. This transition takes the form

$$(w_1, w_2, N, l, m, r) \xrightarrow{(N, N+1, \texttt{</s>})} \text{END}$$

For this transition, we define the score as $score = h(\texttt{</s>}|w_1, w_2)$; thus this transition incorporates the end symbol </s> in the language model.

The states and transitions we have described form a directed graph, where each path from the start state to the end state corresponds to a sequence of phrases $p_1 p_2 \ldots p_L$. We define $\mathcal{Y}'$ to be the full set of such sequences. We can use the Viterbi algorithm to solve $\arg\max_{y \in \mathcal{Y}'} f(y)$ by simply searching for the highest scoring path from the start state to the end state.

The set $\mathcal{Y}'$ clearly includes derivations that are ill-formed, in that they may include words that have been translated 0 times, or more than 1 time. The first line of Figure 2-2 shows one such derivation (corresponding to the translation *the quality and also the and the quality and also* .). For each phrase we show the English string (e.g., *the quality*) together with the span of the phrase (e.g., $3, 6$). The values for $y(i)$ are also shown. It can be verified that this derivation is a valid member of $\mathcal{Y}'$. However, $y(i) \neq 1$ for several values of i: for example, words 1 and 2 are translated 0 times, while word 3 is translated twice.

Other dynamic programs, and definitions of $\mathcal{Y}'$, are possible: for example an alternative would be to use a dynamic program with states (w_1, w_2, n, r). However, including the previous contiguous span (l, m) makes the set $\mathcal{Y}'$ a closer approximation to $\mathcal{Y}$. In experiments we have found that including the previous span (l, m) in the dynamic program leads to faster convergence of the subgradient algorithm described in the next section, and in general to more stable results. This faster convergence is in spite of the dynamic program being larger; it is no doubt due to $\mathcal{Y}'$ being a better approximation of $\mathcal{Y}$.

Initialization: $u^0(i) \leftarrow 0 \quad$ for $i = 1 \ldots N$

for $t = 1 \ldots T$

  $y^t = \arg\max_{y \in \mathcal{Y}'} L(u^{t-1}, y)$

  if $y^t(i) = 1$ for $i = 1 \ldots N$

    return y^t

  else

    for $i = 1 \ldots N$

      $u^t(i) = u^{t-1}(i) - \alpha^t \left(y^t(i) - 1\right)$

Figure 2-1: The decoding algorithm. $\alpha^t > 0$ is the step size at the t'th iteration.

Input German: dadurch können die qualität und die regelmäßige postzustellung auch weiterhin sichergestellt werden .

t	$L(u^{t-1})$	$y^t(i)$	derivation y^t
1	-10.0988	0 0 2 2 3 3 0 0 2 0 0 0 1	$3,6$ the quality and | $9,9$ also | $6,6$ the | $5,5$ and | $3,3$ the | $4,6$ quality and | $9,9$ also | $13,13$.
2	-11.1597	0 0 1 0 0 0 1 0 0 4 1 5 1	$3,3$ the | $7,7$ regular | $12,12$ will | $10,10$ continue to | $12,12$ be | $10,10$ continue to | $12,12$ be | $10,10$ continue to | $12,12$ be | $10,10$ continue to | $11,13$ be guaranteed .
3	-12.3742	3 3 1 2 2 0 0 0 1 0 0 0 1	$1,2$ in that way , | $5,5$ and | $2,2$ can | $1,1$ thus | $4,4$ quality | $1,2$ in that way , | $3,5$ the quality and | $9,9$ also | $13,13$.
4	-11.8623	0 1 0 0 0 1 1 3 3 0 3 0 1	$2,2$ can | $6,7$ the regular | $8,8$ distribution should | $9,9$ also | $11,11$ ensure | $8,8$ distribution should | $9,9$ also | $11,11$ ensure | $8,8$ distribution should | $9,9$ also | $11,11$ ensure | $13,13$.
5	-13.9916	0 0 1 1 3 2 4 0 0 0 1 0 1	$3,3$ the | $7,7$ regular | $5,5$ and | $7,7$ regular | $5,5$ and | $7,7$ regular | $6,6$ the | $4,4$ quality | $5,7$ and the regular | $11,11$ ensured | $13,13$.
6	-15.6558	1 1 1 2 0 2 0 1 1 1 1 1 1	$1,2$ in that way , | $3,4$ the quality of | $6,6$ the | $4,4$ quality of | $6,6$ the | $8,8$ distribution should | $9,10$ continue to | $11,13$ be guaranteed .
7	-16.1022	1 1 1 1 1 1 1 1 1 1 1 1 1	$1,2$ in that way , | $3,4$ the quality | $5,7$ and the regular | $8,8$ distribution should | $9,10$ continue to | $11,13$ be guaranteed .

Figure 2-2: An example run of the algorithm in Figure 2-1. For each value of t we show the dual value $L(u^{t-1})$, the derivation y^t, and the number of times each word is translated, $y^t(i)$ for $i = 1 \ldots N$. For each phrase in a derivation we show the English string e, together with the span (s, t): for example, the first phrase in the first derivation has English string *the quality and*, and span $(3, 6)$. At iteration 7 we have $y^t(i) = 1$ for $i = 1 \ldots N$, and the translation is returned, with a guarantee that it is optimal.

2.4.2 The Lagrangian Relaxation Algorithm

We now describe the Lagrangian relaxation decoding algorithm for the phrase-based model. Recall that in the previous section, we defined a set $\mathcal{Y}'$ that allowed efficient dynamic programming, and such that $\mathcal{Y} \subset \mathcal{Y}'$. It is easy to see that $\mathcal{Y} = \{y : y \in \mathcal{Y}', \quad \text{and} \quad \forall i,\ y(i) = 1\}$. The original decoding problem can therefore be stated as:

$$\arg\max_{y \in \mathcal{Y}'} f(y) \ \text{ such that } \forall i,\ y(i) = 1$$

We use Lagrangian relaxation (Korte and Vygen, 2008) to deal with the $y(i) = 1$ constraints. We introduce Lagrange multipliers $u(i)$ for each such constraint. The Lagrange multipliers $u(i)$

can take any positive or negative value. The Lagrangian is

$$L(u, y) = f(y) + \sum_i u(i)(y(i) - 1)$$

The dual objective is then

$$L(u) = \max_{y \in \mathcal{Y}'} L(u, y).$$

and the dual problem is to solve

$$\min_u L(u).$$

The next section gives a number of formal results describing how solving the dual problem will be useful in solving the original optimization problem.

We now describe an algorithm that solves the dual problem. By standard results for Lagrangian relaxation (Korte and Vygen, 2008), $L(u)$ is a convex function; it can be minimized by a subgradient method. If we define

$$y_u = \arg\max_{y \in \mathcal{Y}'} L(u, y)$$

and $\gamma_u(i) = y_u(i) - 1$ for $i = 1 \ldots N$, then γ_u is a subgradient of $L(u)$ at u. A subgradient method is an iterative method for minimizing $L(u)$, which perfoms updates $u^t \leftarrow u^{t-1} - \alpha^t \gamma_{u^{t-1}}$ where $\alpha^t > 0$ is the step size for the t'th subgradient step. In our experiment, we follow Koo et al. (2010) and set the step size at the tth iteration to be $\alpha^t = 1/(1 + \lambda^t)$, where λ^t is the number of times that $L(u^{(t')}) > L(u^{(t'-1)})$ for all $t' \leq t$.

Figure 2-1 depicts the resulting algorithm. At each iteration, we solve

$$\begin{aligned} &\arg\max_{y \in \mathcal{Y}'} \left(f(y) + \sum_i u(i)(y(i) - 1) \right) \\ =&\arg\max_{y \in \mathcal{Y}'} \left(f(y) + \sum_i u(i) y(i) \right) \end{aligned}$$

by the dynamic program described in the previous section. Incorporating the $\sum_i u(i)y(i)$ terms in the dynamic program is straightforward: we simply redefine the phrase scores as

$$g'(s,t,e) = g(s,t,e) + \sum_{i=s}^{t} u(i)$$

Intuitively, each Lagrange multiplier $u(i)$ penalizes or rewards phrases that translate word i; the algorithm attempts to adjust the Lagrange multipliers in such a way that each word is translated exactly once. The updates $u^t(i) = u^{t-1}(i) - \alpha^t(y^t(i) - 1)$ will decrease the value for $u(i)$ if $y^t(i) > 1$, increase the value for $u(i)$ if $y^t(i) = 0$, and leave $u(i)$ unchanged if $y^t(i) = 1$.

2.4.3 Properties

We now give some theorems stating formal properties of the Lagrangian relaxation algorithm. First, define y^* to be the optimal solution for our original problem:

Definition 1. $y^* = \arg\max_{y \in \mathcal{Y}} f(y)$

Our first theorem states that the dual function provides an upper bound on the score for the optimal translation, $f(y^*)$:

Theorem 1. *For any value of* $u \in \mathbb{R}^N$, $L(u) \geq f(y^*)$.

Proof.

$$\begin{aligned} L(u) &= \max_{y \in \mathcal{Y}'} \left(f(y) + \sum_i u(i)(y(i) - 1) \right) \\ &\geq \max_{y \in \mathcal{Y}} \left(f(y) + \sum_i u(i)(y(i) - 1) \right) \\ &= \max_{y \in \mathcal{Y}} f(y) \end{aligned}$$

The first inequality follows because $\mathcal{Y} \subset \mathcal{Y}'$. The final equality is true since any $y \in \mathcal{Y}$ has $y(i) = 1$ for all i, implying that $\sum_i u(i)(y(i) - 1) = 0$. □

The second theorem states that under an appropriate choice of the step sizes α^t, the method converges to the minimum of $L(u)$. Hence we will successfully find the tightest possible upper

bound defined by the dual $L(u)$.

Theorem 2. *For any sequence* $\alpha^1, \alpha^2, \ldots$ *If 1)* $\lim_{t\to\infty} \alpha^t \to 0$*; 2)* $\sum_{t=1}^{\infty} \alpha^t = \infty$*, then* $\lim_{t\to\infty} L(u^t) = \min_u L(u)$

Proof. See Korte and Vygen (2008). □

Our final theorem states that if at any iteration the algorithm finds a solution y^t such that $y^t(i) = 1$ for $i = 1 \ldots N$, then this is guaranteed to be the optimal solution to our original problem. First, define

Definition 2. $y_u = \arg\max_{y\in\mathcal{Y}'} L(u, y)$

We then have the theorem

Theorem 3. *If* $\exists\ u$*, s.t.* $y_u(i) = 1$ *for* $i = 1 \ldots N$*, then* $f(y_u) = f(y^*)$*, i.e.* y_u *is optimal.*

Proof. We have

$$\begin{aligned} L(u) &= \max_{y\in\mathcal{Y}'} \left(f(y) + \sum_i u(i)(y(i) - 1) \right) \\ &= f(y_u) + \sum_i u(i)(y_u(i) - 1) \\ &= f(y_u) \end{aligned}$$

The second equality is true because of the definition of y_u. The third equality follows because by assumption $y_u(i) = 1$ for $i = 1 \ldots N$. Because $L(u) = f(y_u)$ and $L(u) \geq f(y^*)$ for all u, we have $f(y_u) \geq f(y^*)$. But $y^* = \arg\max_{y\in\mathcal{Y}} f(y)$, and $y_u \in \mathcal{Y}$, hence we must also have $f(y_u) \leq f(y^*)$ hence $f(y_u) = f(y^*)$. □

In summary, we have shown that if the algorithm in Figure 2-1 returns a solution y^t such that $y^t(i) = 1$ for all i, then this solution is guaranteed to be optimal.

In some cases, however, the algorithm in Figure 2-1 may not return a solution y^t such that $y^t(i) = 1$ for all i. There could be two reasons for this. In the first case, we may not have

# iter.	1-10 words	11-20 words	21-30 words	31-40 words	41-50 words	All sentences	
0-7	166 (89.7 %)	219 (39.2 %)	34 (6.0 %)	2 (0.6 %)	0 (0.0 %)	421 (23.1 %)	23.1 %
8-15	17 (9.2 %)	187 (33.5 %)	161 (28.4 %)	30 (8.6 %)	3 (1.8 %)	398 (21.8 %)	44.9 %
16-30	1 (0.5 %)	93 (16.7 %)	208 (36.7 %)	112 (32.3 %)	22 (13.1 %)	436 (23.9 %)	68.8 %
31-60	1 (0.5 %)	52 (9.3 %)	105 (18.6 %)	99 (28.5 %)	62 (36.9 %)	319 (17.5 %)	86.3 %
61-120	0 (0.0 %)	7 (1.3 %)	54 (9.5 %)	89 (25.6 %)	45 (26.8 %)	195 (10.7 %)	97.0 %
121-250	0 (0.0 %)	0 (0.0 %)	4 (0.7 %)	14 (4.0 %)	31 (18.5 %)	49 (2.7 %)	99.7 %
x	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)	1 (0.3 %)	5 (3.0 %)	6 (0.3 %)	100.0 %

Table 2.1: Table showing the number of iterations taken for the algorithm to converge. x indicates sentences that fail to converge after 250 iterations. 97% of the examples converge within 120 iterations.

run the algorithm for enough iterations T to see convergence. In the second case, the underlying relaxation may not be tight, in that there may not be *any* settings u for the Lagrange multipliers such that $y_u(i) = 1$ for all i.

Section 2.5 describes a method for tightening the underlying relaxation by introducing hard constraints (of the form $y(i) = 1$ for selected values of i). We will see that this method is highly effective in tightening the relaxation until the algorithm converges to an optimal solution.

2.4.4 An Example of the Algorithm

Figure 2-2 shows an example of how the algorithm works when translating a German sentence into an English sentence. After the first iteration, there are words that have been translated two or three times, and words that have not been translated. At each iteration, the Lagrangian multipliers are updated to encourage each word to be translated once. On this example, the algorithm converges to a solution where all words are translated exactly once, and the solution is guaranteed to be optimal.

2.5 Tightening the Relaxation

In some cases the algorithm in Figure 2-1 will not converge to $y(i) = 1$ for $i = 1 \dots N$ because the underlying relaxation is not tight. We now describe a method that incrementally tightens the Lagrangian relaxation algorithm until it provides an exact answer. In cases that do not converge, we introduce hard constraints to force certain words to be translated exactly once in the dynamic programming solver. In experiments we show that typically only a few constraints are necessary.

$Optimize(\mathcal{C}, u)$
while (dual value still improving)
$y^* = \arg\max_{y \in \mathcal{Y}'_{\mathcal{C}}} L(u, y)$
if $y^*(i) = 1$ for $i = 1 \ldots N$ **return** y^*
else for $i = 1 \ldots N$
$u(i) = u(i) - \alpha\,(y^*(i) - 1)$
$count(i) = 0$ for $i = 1 \ldots N$
for $k = 1 \ldots K$
$y^* = \arg\max_{y \in \mathcal{Y}'_{\mathcal{C}}} L(u, y)$
if $y^*(i) = 1$ for $i = 1 \ldots N$ **return** y^*
else for $i = 1 \ldots N$
$u(i) = u(i) - \alpha\,(y^*(i) - 1)$
$count(i) = count(i) + [[y^*(i) \neq 1]]$
Let $\mathcal{C}' =$ set of G i's that have the largest value for $count(i)$ and that are not in $\mathcal{C}$
return $Optimize(\mathcal{C} \cup \mathcal{C}', u)$

Figure 2-3: A decoding algorithm with incremental addition of constraints. The function $Optimize(\mathcal{C}, u)$ is a recursive function, which takes as input a set of constraints $\mathcal{C}$, and a vector of Lagrange multipliers, u. The initial call to the algorithm is with $\mathcal{C} = \emptyset$, and $u = 0$. $\alpha > 0$ is the step size. In our experiments, the step size decreases with the number of iteration; see Appendix A.1.

Given a set $\mathcal{C} \subseteq \{1, 2, \ldots, N\}$, we define

$$\mathcal{Y}'_{\mathcal{C}} = \{y : y \in \mathcal{Y}', \text{ and } \forall\, i \in \mathcal{C},\ \ y(i) = 1\}$$

Thus $\mathcal{Y}'_{\mathcal{C}}$ is a subset of $\mathcal{Y}'$, formed by adding hard constraints of the form $y(i) = 1$ to $\mathcal{Y}'$. Note that $\mathcal{Y}'_{\mathcal{C}}$ remains as a superset of $\mathcal{Y}$, which enforces $y(i) = 1$ for all i. Finding $\arg\max_{y \in \mathcal{Y}'_{\mathcal{C}}} f(y)$ can again be achieved using dynamic programming, with the number of dynamic programming states increased by a factor of $2^{|\mathcal{C}|}$: dynamic programming states of the form (w_1, w_2, n, l, m, r) are replaced by states $(w_1, w_2, n, l, m, r, b_{\mathcal{C}})$ where $b_{\mathcal{C}}$ is a bit-string of length $|\mathcal{C}|$, which records which words in the set $\mathcal{C}$ have or haven't been translated in a hypothesis (partial derivation). Note that if $\mathcal{C} = \{1 \ldots N\}$, we have $\mathcal{Y}'_{\mathcal{C}} = \mathcal{Y}$, and the dynamic program will correspond to exhaustive dynamic programming.

We can again run a Lagrangian relaxation algorithm, using the set $\mathcal{Y}'_{\mathcal{C}}$ in place of $\mathcal{Y}'$. We will use Lagrange multipliers $u(i)$ to enforce the constraints $y(i) = 1$ for $i \notin \mathcal{C}$. Our goal will be to find

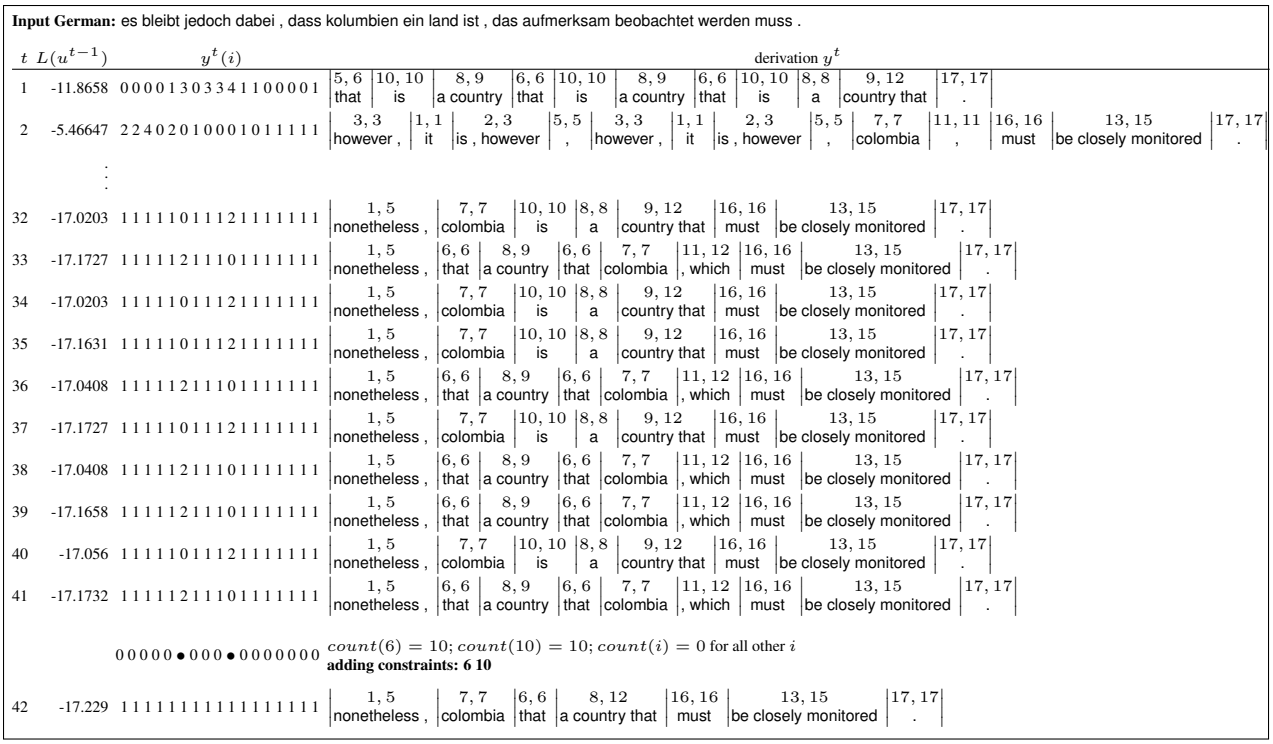


Figure 2-4: An example run of the algorithm in Figure 2-3. At iteration 32, we start the $K = 10$ iterations to count which constraints are violated most often. After K iterations, the count for 6 and 10 is 10, and all other constraints have not been violated during the K iterations. Thus, hard constraints for word 6 and 10 are added. After adding the constraints, we have $y^t(i) = 1$ for $i = 1 \dots N$, and the translation is returned, with a guarantee that it is optimal.

a small set of constraints $\mathcal{C}$, such that Lagrangian relaxation will successfully recover an optimal solution. We will do this by incrementally adding elements to $\mathcal{C}$; that is, by incrementally adding constraints that tighten the relaxation.

The intuition behind our approach is as follows. Say we run the original algorithm, with the set $\mathcal{Y}'$, for several iterations, so that $L(u)$ is close to convergence (i.e., $L(u)$ is close to its minimal value). However, assume that we have not yet generated a solution y^t such that $y^t(i) = 1$ for all i. In this case we have some evidence that the relaxation may not be tight, and that we need to add some constraints. The question is, which constraints to add? To answer this question, we run the subgradient algorithm for K more iterations (e.g., $K = 10$), and at each iteration track which constraints of the form $y(i) = 1$ are violated. We then choose $\mathcal{C}$ to be the G constraints (e.g., $G = 3$) that are violated most often during the K additional iterations, and are not adjacent to each

other. We recursively call the algorithm, replacing $\mathcal{Y}'$ by $\mathcal{Y}'_{\mathcal{C}}$; the recursive call may then return an exact solution, or alternatively again add more constraints and make a recursive call.[4]

Figure 2-3 depicts the resulting algorithm. We initially make a call to the algorithm $Optimize(\mathcal{C}, u)$ with $\mathcal{C}$ equal to the empty set (i.e., no hard constraints), and with $u(i) = 0$ for all i. In an initial phase the algorithm runs subgradient steps, while the dual is still improving. In a second step, if a solution has not been found, the algorithm runs for K more iterations, thereby choosing G additional constraints, then recursing.

If at any stage the algorithm finds a solution y^* such that $y^*(i) = 1$ for all i, then this is the solution to our original problem, $\arg\max_{y \in \mathcal{Y}} f(y)$. This follows because for any $\mathcal{C} \subseteq \{1 \dots N\}$ we have $\mathcal{Y} \subseteq \mathcal{Y}'_{\mathcal{C}}$; hence the theorems in Section 2.4.3 go through for $\mathcal{Y}'_{\mathcal{C}}$ in place of $\mathcal{Y}'$, with trivial modifications. Note also that the algorithm is guaranteed to eventually find the optimal solution, because eventually $\mathcal{C} = \{1 \dots N\}$, and $\mathcal{Y} = \mathcal{Y}'_{\mathcal{C}}$.

The remaining question concerns the "dual still improving" condition; i.e., how to determine that the first phase of the algorithm should terminate. We do this by recording the first and second best dual values $L(u')$ and $L(u'')$ in the sequence of Lagrange multipliers $u^1, u^2, \dots$ generated by the algorithm. Suppose that $L(u'')$ first occurs at iteration t''. If $\frac{L(u') - L(u'')}{t - t''} < \epsilon$, we say that the dual value does not decrease enough. The value for ϵ is a parameter of the approach: in experiments we used $\epsilon = 0.002$.

Figure 2-4 gives an example run of the algorithm. After 31 iterations the algorithm detects that the dual is no longer decreasing rapidly enough, and runs for $K = 10$ additional iterations, tracking which constraints are violated. Constraints $y(6) = 1$ and $y(10) = 1$ are each violated 10 times, while other constraints are not violated. A recursive call to the algorithm is made with $\mathcal{C} = \{6, 10\}$, and the algorithm converges in a single iteration, to a solution that is guaranteed to be optimal.

When $\mathcal{C} \neq \emptyset$, A* search can be used for decoding, with the dynamic program for $\mathcal{Y}'$ providing

[4]Formal justification for the method comes from the relationship between Lagrangian relaxation and linear programming relaxations. In cases where the relaxation is not tight, the subgradient method will essentially move between solutions whose convex combination form a fractional solution to an underlying LP relaxation (Nedić and Ozdaglar, 2009). Our method eliminates the fractional solution through the introduction of hard constraints.

# cons.	1-10 words	11-20 words	21-30 words	31-40 words	41-50 words	All sentences	
0-0	183 (98.9 %)	511 (91.6 %)	438 (77.4 %)	222 (64.0 %)	82 (48.8 %)	1,436 (78.7 %)	78.7 %
1-3	2 (1.1 %)	45 (8.1 %)	94 (16.6 %)	87 (25.1 %)	50 (29.8 %)	278 (15.2 %)	94.0 %
4-6	0 (0.0 %)	2 (0.4 %)	27 (4.8 %)	24 (6.9 %)	19 (11.3 %)	72 (3.9 %)	97.9 %
7-9	0 (0.0 %)	0 (0.0 %)	7 (1.2 %)	13 (3.7 %)	12 (7.1 %)	32 (1.8 %)	99.7 %
x	0 (0.0 %)	0 (0.0 %)	0 (0.0 %)	1 (0.3 %)	5 (3.0 %)	6 (0.3 %)	100.0 %

Table 2.2: Table showing the number of constraints added before convergence of the algorithm in Figure 2-3, broken down by sentence length. Note that a maximum of 3 constraints are added at each recursive call, but that fewer than 3 constraints are added in cases where fewer than 3 constraints have $count(i) > 0$. x indicates the sentences that fail to converge after 250 iterations. 78.7% of the examples converge without adding any constraints.

# cons.	1-10 words		11-20 words		21-30 words		31-40 words		41-50 words		All sentences	
	A*	w/o	A*	w/o	A*	w/o	A*	w/o	A*	w/o	A*	w/o
0-0	0.8	0.8	9.7	10.7	47.0	53.7	153.6	178.6	402.6	492.4	64.6	76.1
1-3	2.4	2.9	23.2	28.0	80.9	102.3	277.4	360.8	686.0	877.7	241.3	309.7
4-6	0.0	0.0	28.2	38.8	111.7	163.7	309.5	575.2	1,552.8	1,709.2	555.6	699.5
7-9	0.0	0.0	0.0	0.0	166.1	500.4	361.0	1,467.6	1,167.2	3,222.4	620.7	1,914.1
mean	0.8	0.9	10.9	12.3	57.2	72.6	203.4	299.2	679.9	953.4	120.9	168.9
median	0.7	0.7	8.9	9.9	48.3	54.6	169.7	202.6	484.0	606.5	35.2	40.0

Table 2.3: The average time (in seconds) for decoding using the algorithm in Figure 2-3, with and without A* algorithm, broken down by sentence length and the number of constraints that are added. A* indicates speeding up using A* search; w/o denotes without using A*.

admissible estimates for the dynamic program for $\mathcal{Y}'_{\mathcal{C}}$. Experiments show that A* gives significant improvements in efficiency. The next section contains a full description of the A* algorithm.

2.6 Speeding up the DP: A* Search

In the algorithm depicted in Figure 2-3, each time we call $Optimize(\mathcal{C} \cup \mathcal{C}', u)$, we expand the number of states in the dynamic program by adding hard constraints. On the graph level, adding hard constraints can be viewed as expanding an original state in $\mathcal{Y}'$ to $2^{|\mathcal{C}|}$ states in $\mathcal{Y}'_{\mathcal{C}}$, since now we keep a bit-string $b_{\mathcal{C}}$ of length $|\mathcal{C}|$ in the states to record which words in $\mathcal{C}$ have or haven't been translated. We now show how this observation leads to an A* algorithm that can significantly improve efficiency when decoding with $\mathcal{C} \neq \emptyset$.

For any state $s = (w_1, w_2, n, l, m, r, b_{\mathcal{C}})$ and Lagrange multiplier values $u \in \mathbb{R}^N$, define $\beta_{\mathcal{C}}(s, u)$ to be the maximum score for any path from the state s to the end state, under Lagrange multipliers u, in the graph created using constraint set $\mathcal{C}$. Define $\pi(s) = (w_1, w_2, n, l, m, r)$, that is, the corresponding state in the graph with no constraints ($\mathcal{C} = \emptyset$). Then for any values of s and u, we

have

$$\beta_{\mathcal{C}}(s, u) \leq \beta_{\emptyset}(\pi(s), u)$$

That is, the maximum score for any path to the end state in the graph with no constraints, forms an upper bound on the value for $\beta_{\mathcal{C}}(s, u)$.

This observation leads directly to an A* algorithm, which is exact in finding the optimum solution, since we can use $\beta_{\emptyset}(\pi(s), u)$ as the admissible estimates for the score from state s to the goal (the end state). The $\beta_{\emptyset}(s', u)$ values for all s' can be calculated by running the Viterbi algorithm using a backwards path. With only $1/2^{|\mathcal{C}|}$ states, calculating $\beta_{\emptyset}(s', u)$ is much cheaper than calculating $\beta_{\mathcal{C}}(s, u)$ directly. Guided by $\beta_{\emptyset}(s', u)$, $\beta_{\mathcal{C}}(s, u)$ can be calculated efficiently by using A* search.

Using the A* algorithm leads to significant improvements in efficiency when constraints are added. Section 4.7 presents comparison of the running time with and without A* algorithm.

2.7 Experiments

In this section, we present experimental results to demonstrate the efficiency of the decoding algorithm. We compare to MOSES (Koehn et al., 2007), a phrase-based decoder using beam search, and to a general purpose integer linear programming (ILP) solver, which solves the problem exactly.

The experiments focus on translation from German to English, using the Europarl data (Koehn, 2005). We tested on 1,824 sentences of length at most 50 words. The experiments use the algorithm shown in Figure 2-3. We limit the algorithm to a maximum of 250 iterations and a maximum of 9 hard constraints. The distortion limit d is set to be four, and we prune the phrase translation table to have 10 English phrases per German phrase.

Our method finds exact solutions on 1,818 out of 1,824 sentences (99.67%). (6 examples do not converge within 250 iterations.) Table 2.1 shows the number of iterations required for convergence, and Table 2.2 shows the number of constraints required for convergence, broken down by sentence

length. In 1,436/1,818 (78.7%) sentences, the method converges without adding hard constraints to tighten the relaxation. For sentences with 1-10 words, the vast majority (183 out of 185 examples) converge with 0 constraints added. As sentences get longer, more constraints are often required. However most examples converge with 9 or fewer constraints.

Table 2.3 shows the average times for decoding, broken down by sentence length, and by the number of constraints that are added. As expected, decoding times increase as the length of sentences, and the number of constraints required, increase. The average run time across all sentences is 120.9 seconds. Table 2.3 also shows the run time of the method without the A* algorithm for decoding. The A* algorithm gives significant reductions in runtime.

2.7.1 An alternative dynamic program

In the end of Section 2.4.1, we mention that an alternative dynamic program with states (w_1, w_2, n, r) is possible. We experiment with the algorithm depicted in Figure 2-3 using this dynamic program. Experiments shows that it requires more constraints and more iterations, but each iteration is cheaper than using $\mathcal{Y}'$. The average runtime is 1.2 seconds for sentences with 1–10 words, 19.7 seconds for sentences with 11-20 words, and 102.4 seconds for those with 21-30 words. However, the average runtime shows that using set $\mathcal{Y}'$, which is a closer approximate of the set $\mathcal{Y}$, is more stable.

2.7.2 Comparison to an LP/ILP solver

To compare to a linear programming (LP) or integer linear programming (ILP) solver, we can implement the dynamic program (search over the set $\mathcal{Y}'$) through linear constraints, with a linear objective. The $y(i) = 1$ constraints are also linear. Hence we can encode our relaxation within an LP or ILP. Having done this, we tested the resulting LP or ILP using Gurobi, a high-performance commercial grade solver. We also compare to an LP or ILP where the dynamic program makes use of states (w_1, w_2, n, r)—i.e., the span (l, m) is dropped, making the dynamic program smaller. Table 2.4 shows the average time taken by the LP/ILP solver. Both the LP and the ILP require

method		ILP		LP		
set	length	mean	median	mean	median	% frac.
$\mathcal{Y}''$	1-10	275.2	132.9	10.9	4.4	12.4 %
	11-15	2,707.8	1,138.5	177.4	66.1	40.8 %
	16-20	20,583.1	3,692.6	1,374.6	637.0	59.7 %
$\mathcal{Y}'$	1-10	257.2	157.7	18.4	8.9	1.1 %
	11-15	3607.3	1838.7	476.8	161.1	3.0 %

Table 2.4: Average and median time of the LP/ILP solver (in seconds). % frac. indicates how often the LP gives a fractional answer. $\mathcal{Y}'$ indicates the dynamic program using set $\mathcal{Y}'$ as defined in Section 2.4.1, and $\mathcal{Y}''$ indicates the dynamic program using states (w_1, w_2, n, r). The statistics for ILP for length 16-20 are based on 50 sentences.

very long running times on these shorter sentences, and running times on longer sentences are prohibitive. Our algorithm is more efficient because it leverages the structure of the problem, by directly using a combinatorial algorithm (dynamic programming).

2.7.3 Comparison to MOSES

We now describe comparisons to the phrase-based decoder implemented in MOSES. MOSES uses beam search to find approximate solutions.

The distortion limit described in Section 4.3 is the same as that in Koehn et al. (2003), and is the same as that described in the user manual for MOSES (Koehn et al., 2007). However, a complicating factor for our comparisons is that MOSES uses an additional distortion constraint, not documented in the manual, which we describe here.[5] We call this constraint the *gap constraint*. We will show in experiments that without the gap constraint, MOSES fails to produce translations on many examples. In our experiments we will compare to MOSES both with and without the gap constraint (in the latter case, we discard examples where MOSES fails).

We now describe the gap constraint. For a sequence of phrases $p_1, \ldots, p_k$ define $\theta(p_1 \ldots p_k)$ to be the index of the left-most source-language word not translated in this sequence. For example, if the bit-string for $p_1 \ldots p_k$ is 111001101000, then $\theta(p_1 \ldots p_k) = 4$. A sequence of phrases $p_1 \ldots p_L$ satisfies the gap constraint if and only if for $k = 2 \ldots L$, $|t(p_k) + 1 - \theta(p_1 \ldots p_k)| \leq d$. where d is the distortion limit. We will call MOSES without this restriction MOSES-nogc, and MOSES with this restriction MOSES-gc.

[5] Personal communication from Philipp Koehn; see also the software for MOSES.

Beam size	Fails	# search errors	percentage
100	650/1,818	214/1,168	18.32 %
200	531/1,818	207/1,287	16.08 %
1000	342/1,818	115/1,476	7.79 %
10000	169/1,818	68/1,649	4.12 %

Table 2.5: Table showing the number of examples where MOSES-nogc fails to give a translation, and the number/percentage of search errors for cases where it does give a translation.

Diff.	MOSES-gc s =100	MOSES-gc s =200	MOSES-nogc s=1000
0.000 – 0.125	66 (24.26%)	65 (24.07%)	32 (27.83%)
0.125 – 0.250	59 (21.69%)	58 (21.48%)	25 (21.74%)
0.250 – 0.500	65 (23.90%)	65 (24.07%)	25 (21.74%)
0.500 – 1.000	49 (18.01%)	49 (18.15%)	23 (20.00%)
1.000 – 2.000	31 (11.40%)	31 (11.48%)	5 (4.35%)
2.000 – 4.000	2 (0.74%)	2 (0.74%)	3 (2.61%)
4.000 –13.000	0 (0.00%)	0 (0.00%)	2 (1.74%)

Table 2.6: Table showing statistics for the difference between the translation score from MOSES, and from the optimal derivation, for those sentences where a search error is made. For MOSES-gc we include cases where the translation produced by our system is not reachable by MOSES-gc. The average score of the optimal derivations is -23.4.

Results for MOSES-nogc Table 2.5 shows the number of examples where MOSES-nogc fails to give a translation, and the number of search errors for those cases where it does give a translation, for a range of beam sizes. A search error is defined as a case where our algorithm produces an exact solution that has higher score than the output from MOSES-nogc. The number of search errors is significant, even for large beam sizes.

Results for MOSES-gc MOSES-gc uses the gap constraint, and thus in some cases our decoder will produce derivations which MOSES-gc cannot reach. Among the 1,818 sentences where we produce a solution, there are 270 such derivations. For the remaining 1,548 sentences, MOSES-gc makes search errors on 2 sentences (0.13%) when the beam size is 100, and no search errors when the beam size is 200, 1,000, or 10,000.

Finally, table 2.6 shows statistics for the magnitude of the search errors that MOSES-gc and MOSES-nogc make.

type of Moses	beam size	# sentence	Moses	our code
MOSES-gc	100	1,818	24.4773	24.5395
	200	1,818	24.4765	24.5395
	1,000	1,818	24.4765	24.5395
	10,000	1,818	24.4765	24.5395
MOSES-nogc	100	1,168	27.3546	27.3249
	200	1,287	27.0591	26.9907
	1,000	1,476	26.5734	26.6128
	10,000	1,649	25.6531	25.6620

Table 2.7: Comparison of BLEU score. We only consider the sentences that both programs produce an answer.

2.8 Conclusions

We have described an exact decoding algorithm for phrase-based translation models, using Lagrangian relaxation. The algorithmic construction we have described may also be useful in other areas of NLP, for example natural language generation. Possible extensions to the approach include methods that incorporate the Lagrangian relaxation formulation within learning algorithms for statistical MT: we see this as an interesting avenue for future research.

Chapter 3

Optimal Beam Search

[This chapter is adapted from joint work with Alexander Rush and Michael Collins entitled "Optimal Beam Search for Machine Translation" (Rush et al., 2013).]

In this chapter, we will present another algorithm for decoding of phrase-based translation models. The algorithm is still based on Lagrangian relaxation. Instead of using the tightening technique to achieve an exact solution, this algorithm combines the Lagrangian relaxation method with beam search. We use the Lagrangian relaxation method to provide a good admissible heuristic, and use beam search to obtain a primal feasible solution. Similar to the A* algorithm described in Section 2.6, the admissible heuristic guarantees that the solution is optimal if no pruning by beam size is applied.

The algorithm has several differences and advantages over the algorithm presented in Chapter 2:

- It provides a feasible solution even when the algorithm does not converge, and gives a bound for the difference between the solution and the optimal solution.

- It can utilize Polyak's rule to compute step size since an lower bound and an upper bound are available at all iterations.

- Overall, it is more efficient in practice.

3.1 Introduction

Beam search Koehn et al. (2003) and cube pruning Chiang (2007) have become the *de facto* decoding algorithms for phrase- and syntax-based translation. The algorithms are central to large-scale machine translation systems due to their efficiency and tendency to produce high-quality translations Koehn (2004); Koehn et al. (2007); Dyer et al. (2010). However despite practical effectiveness, neither algorithm provides any bound on possible decoding error.

In this work we present a variant of beam search decoding for phrase- and syntax-based translation. The motivation is to exploit the effectiveness and efficiency of beam search, but still maintain formal guarantees. The algorithm has the following benefits:

- In theory, it can provide a certificate of optimality; in practice, we show that it produces optimal hypotheses, with certificates of optimality, on the vast majority of examples.
- It utilizes well-studied algorithms and extends off-the-shelf beam search decoders.
- Empirically it is very fast, results show that it is 3.5 times faster than heavily-optimized relaxation-based solvers. (The two solvers are solving exactly the same problem and produce exact solutions on the same number of example sentences.)

While our focus is on fast decoding for machine translation, the algorithm we present can be applied to a variety of dynamic programming-based decoding problems. The method only relies on having a fast constrained beam search algorithm and a fast unconstrained dual search algorithm. Similar algorithms exist for many NLP tasks.

We begin in Section 3.2 by describing constrained hypergraph search and showing how it generalizes translation decoding. Section 3.3 introduces a variant of beam search that is, in theory, able to produce a certificate of optimality. Section 3.4 shows how to improve the chance that beam search produces a certificate by using bounds derived from Lagrangian relaxation. Section 3.5 puts everything together to derive a fast beam search algorithm that is often optimal in practice.

Experiments compare the new algorithm with several varieties of beam search, cube pruning, A* search, and relaxation-based decoders on two translation tasks. The optimal beam search

algorithm is able to find exact solutions with certificates of optimality on 99% of phrase-based translation examples and 98% of syntax-based translation examples, significantly more than other baselines. Additionally the optimal beam search algorithm is much faster than other exact methods.

3.2 Background

The focus of this work is decoding for statistical machine translation. Given a source sentence, the goal is to find the target sentence that maximizes a combination of translation model and language model scores. In order to analyze this decoding problem, we first abstract away from the specifics of translation into a general form, known as a hypergraph. In this section, we describe the hypergraph formalism and its relation to machine translation.

3.2.1 Notation

Throughout the chapter, scalars and vectors are written in lowercase, matrices are written in uppercase, and sets are written in script-case, e.g. $\mathcal{X}$. All vectors are assumed to be column vectors. The function $\delta(j)$ yields an indicator vector with $\delta(j)_j = 1$ and $\delta(j)_i = 0$ for all $i \neq j$.

3.2.2 Hypergraphs and Search

A directed hypergraph is a pair $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{1 \dots |\mathcal{V}|\}$ is a set of vertices, and $\mathcal{E}$ is a set of directed hyperedges. Each hyperedge $e \in \mathcal{E}$ is a tuple $\langle \langle v_2, \dots, v_k \rangle, v_1 \rangle$ where $v_i \in \mathcal{V}$ for $i \in \{1 \dots k\}$. The *head* of the hyperedge is $h(e) = v_1$. The *tail* of the hypergraph is the ordered sequence $t(e) = \langle v_2, \dots, v_k \rangle$. The size of the tail $|t(e)|$ may vary across different edges, but $|t(e)| \geq 1$ for all edges. A directed graph is a directed hypergraph with $|t(e)| = 1$ for all edges $e \in \mathcal{E}$.

Each vertex $v \in \mathcal{V}$ is either a *non-terminal* or a *terminal* in the hypergraph. The set of non-terminals is $\mathcal{N} = \{v \in \mathcal{V} : h(e) = v \text{ for some } e \in \mathcal{E}\}$. Conversely, the set of terminals is defined as $\mathcal{T} = \mathcal{V} \setminus \mathcal{N}$.

All directed hypergraphs used in this work are acyclic: informally this implies that no hyperpath (as defined below) contains the same vertex more than once (see Martin et al. (1990) for a full definition). Acyclicity implies a partial topological ordering of the vertices. Let this partial order be given by the inequality operator $\prec$. We also assume there is a distinguished *root* vertex 1 with the property that $1 \prec v$ for all $v \in \mathcal{V} \setminus \{1\}$. For hyperedges, we use $e \prec e'$ as shorthand for $h(e) \prec h(e')$.

Next we define a hyperpath as a tuple $(x, y) \in \{0,1\}^{|\mathcal{V}|} \times \{0,1\}^{|\mathcal{E}|}$ where $x_v = 1$ if vertex v is used in the hyperpath, $x_v = 0$ otherwise (similarly $y_e = 1$ if edge e is used in the hyperpath, $y_e = 0$ otherwise). The set of valid hyperpaths starting at the root is

$$\begin{aligned}
\mathcal{X} = \{(x,y) : x_1 &= 1, \\
x_v &= \sum_{e \in \mathcal{E}: h(e) = v} y_e \quad \forall\, v \in \mathcal{N}, \\
x_v &= \sum_{e \in \mathcal{E}: v \in t(e)} y_e \quad \forall\, v \in \mathcal{V} \setminus \{1\}\}
\end{aligned}$$

The first problem we consider is *unconstrained* hypergraph search. Let $\theta \in R^{|\mathcal{E}|}$ be the weight vector for the hypergraph. The unconstrained search problem is to find

$$\max_{(x,y) \in \mathcal{X}} \sum_{e \in \mathcal{E}} \theta_e y_e = \max_{(x,y) \in \mathcal{X}} \theta^\top y$$

This maximization can be computed for any weight vector and directed acyclic hypergraph in time $O(|\mathcal{E}|)$ using dynamic programming. Figure 3-1 shows this algorithm which is simply a version of the CKY algorithm.

Next consider a variant of this problem: *constrained* hypergraph search. Constraints will be necessary for both phrase- and syntax-based decoding. In phrase-based models, the constraints will ensure that each source word is translated exactly once. In syntax-based models, the constraints will be used to intersect a translation forest with a language model.

In the constrained hypergraph problem, hyperpaths must fulfill additional linear hyperedge

procedure BESTPATHSCORE( )
  $\pi[v] \leftarrow 0$ for all $v \in \mathcal{T}$
  for $e \in \mathcal{E}$ in $\succ$ order **do**
    $\langle \langle v_2, \ldots, v_k \rangle, v_1 \rangle \leftarrow e$
    $s \leftarrow \theta_e + \sum_{i=2}^{k} \pi[v_i]$
    if $s > \pi[v_1]$ **then** $\pi[v_1] \leftarrow s$
  return $\pi[1]$

Figure 3-1: Dynamic programming algorithm for unconstrained hypergraph search. Note that this version only returns the highest score: $\max_{(x,y)\in\mathcal{X}} \theta^\top y$. The optimal hyperpath can be found by including back-pointers.

constraints. Define the set of constrained hyperpaths as

$$\mathcal{X}' = \{(x, y) \in \mathcal{X} : Fy = c\}$$

where we have a constraint matrix $F \in \mathbb{R}^{|c|\times|\mathcal{E}|}$ and vector $c \in R^{|c|}$ encoding $|c|$ constraints. The optimal constrained hyperpath is $x^*, y^* = \arg\max_{(x,y)\in\mathcal{X}'} \theta^\top y$.

Note that the constrained hypergraph search problem may be NP-Hard. Crucially this is true even when the corresponding unconstrained search problem, $\max_{(x,y)\in\mathcal{X}} \theta^\top y$, is solvable in polynomial time. For instance, phrase-based decoding is known to be NP-Hard Knight (1999), but we will see that it can be expressed as a polynomial-sized hypergraph with constraints.

Example: Phrase-Based Machine Translation Consider translating a source sentence $s_1 \ldots s_{|s|}$ to a target sentence in a language with vocabulary Σ. A simple phrase-based translation model consists of a tuple $(\mathcal{P}, f, g)$ with

- $\mathcal{P}$; a set of pairs (r, t) where $r_1 \ldots r_{|r|}$ is a sequence of source words and $t_1 \ldots t_{|t|}$ is a sequence of target words.
- $f : \mathcal{P} \rightarrow \mathbb{R}$; a translation model mapping each element of $\mathcal{P}$ to a real-valued score.
- $g : \Sigma^2 \rightarrow \mathbb{R}$; a language model mapping a target language bigram to a score.

Additionally we define two helper functions. Let $g^{(i)}(t) = \sum_{i=2}^{|t|} g(t_{i-1}, t_i)$ be the internal language model score of a word sequence. Let $g^{(o)}(t, t') = g(t_{|t|}, t'_1)$ be the overlapping language model score of two words adjacent sequences.

A phrase-based derivation is a sequence of phrases $p_1 \dots p_{|p|}$. A phrase p consists of a span in the source sentence $(j(p), k(p))$ with $1 \leq j(p) \leq k(p) \leq |s|$ and a sequence of target words $t(p)$. The span identifies the source sequence $r(p) = s_{j(p)} \dots s_{k(p)}$ that is *translated* by the phrase. Note that a phrase corresponds to a pair $(r(p), t(p)) \in \mathcal{P}$. For a derivation to be valid, it must translate every source word *exactly* once. Call the set of valid derivations $\mathcal{D}$.

The decoding problem for phrase-based translation is to find the highest scoring valid derivation

$$\max_{p \in \mathcal{D}} \sum_{l=1}^{|p|} f(r(p_l), t(p_l)) + g^{(i)}(t(p_l)) + g^{(o)}(t(p_{l-1}), t(p_l))$$

where we define $t(p_0)$ to be the start symbol $*$.

This problem can be expressed as a hypergraph using the construction of Chang and Collins (2011). For conciseness we write vertices as tuples and assume an implied mapping to integers:

- The vertices track the current target word w and the number of source words translated, b.

  Let $\mathcal{V} = \{(b, w) : w \in \Sigma, b \in \{1 \dots |s|\}\}$.

- The edges indicate using a phrase p, thereby increasing the source word count by $|r(p)|$ and setting w to the final word in $t(p)$.

  Let $\mathcal{E} = \left\{ \begin{array}{l} (b, w) \rightarrow (b + |r(p)|, t(p)_{|t(p)|}) : \\ \quad w \in \Sigma, b \in \{1 \dots |s|\}, \text{phrase } p \end{array} \right\}$

- The edge weights are the cost of using phrase p when the current word was w.

  $\theta_e = f(r(p), t(p)) + g^{(i)}(t(p)) + g^{(o)}(w, t(p))$

Any valid derivation corresponds to a path in this graph. However, a path in the graph may not be valid derivation in $\mathcal{D}$, since it may not use all source words exactly once. We enforce this property through constraints.

Define a constraint matrix $F \in \{0,1\}^{|s| \times |\mathcal{E}|}$ where the rows correspond to source indices and the columns correspond to edges. For column $e \in \mathcal{E}$ and row $i \in 1 \ldots |s|$ we have $F_{i,e} = 1$ if the corresponding phrase $p = p(e)$ translates the source word i, i.e. has $j(p) \leq i \leq k(p)$. For all other positions $F_{i,e} = 0$. The product Fy is a vector where $(Fy)_i$ is the number of times word i has been translated.

The constrained hypergraph problem for phrase-based translation ensures that each word is translated exactly once

$$\mathcal{X}' = \{(x, y) \in \mathcal{X} : Fy = \mathbf{1}\}$$

The best derivation under this phrase-based translation model has score $\max_{(x,y) \in \mathcal{X}'} \theta^\top y$.

Example: Syntax-Based Machine Translation Syntax-based machine translation with a language model can also be expressed as a constrained hypergraph problem. For the sake of space, we omit the definition. See Rush and Collins (2011) for an in-depth description of the constraint matrix used for syntax-based translation.

3.3 A Variant of Beam Search

This section describes a variant of the beam search algorithm for finding the highest-scoring constrained hyperpath. The algorithm uses three main techniques: (1) dynamic programming with additional signature information to satisfy the hypergraph constraints, (2) beam pruning where some, possibly optimal, hypotheses are discarded, and (3) branch-and-bound-style application of upper and lower bounds to discard provably non-optimal hypotheses.

Any solution returned by the algorithm will be a valid constrained hyperpath and a member of $\mathcal{X}'$. Additionally the algorithm returns a certificate flag opt that if true means that no beam pruning was used, implying the solution returned is optimal. Generally it will be hard to produce a certificate even by reducing the amount of beam pruning; however in the next section we will introduce a method based on Lagrangian relaxation to tighten the upper bounds. These bounds

will help eliminate most solutions before they trigger pruning.

3.3.0.1 Algorithm

Figure 3-2 shows the complete beam search algorithm. At its core it is a dynamic programming algorithm filling in the chart π. The beam search chart indexes hypotheses by their top vertex $v \in \mathcal{V}$ as well as a signature $sig \in \mathcal{S}$ where $\mathcal{S} \subset R^{|c|}$ and $|c|$ is the number of constraints. A new hypothesis is constructed from each hyperedge and all possible signatures of tail nodes. We define the function SIGS to take the tail of an edge and return the set of possible signature combinations

$$\textsc{Sigs}(\langle v_2, \dots v_k \rangle) = \prod_{i=2}^{k} \{sig : \pi[v_i, sig] \neq -\infty\}$$

Line 7 loops over this entire set. For hypothesis (x, y), the algorithm ensures that its signature sig is equal to Fy. This property is updated on line 8.

The signature provides proof that a hypothesis is still valid. Let the function $\textsc{Check}(sig)$ return true if the hypothesis can still fulfill the constraints. For example, in phrase-based decoding, we will define $\textsc{Check}(sig) = (sig \leq \mathbf{1})$, this ensures that each word has been translated 0 or 1 times. This check is applied on line 10.

Unfortunately maintaining all signatures is inefficient. For example we will see that in phrase-based decoding the signature is a bit-string recording which source words have been translated; the number of possible bit-strings is exponential in the length of the sentence. The algorithm includes two methods for removing potential paths, bounding and pruning.

Bounding allows us to discard provably non-optimal solutions. The algorithm takes as arguments a lower bound on the optimal score $\mathrm{lb} \leq \theta^\top y^*$, and upper bounds on the outside score for all vertices v: $\mathrm{ubs}[v]$, i.e. an overestimate of the score for completing the hyperpath from v. If a hypothesis has score s, it can only be optimal if $s + \mathrm{ubs}[v] \geq \mathrm{lb}$. This bound check is performed on line 10.

Pruning removes weak partial solutions based on problem-specific checks. The algorithm in-

1: **procedure** BEAMSEARCH($\mathrm{lb}, \mathrm{ubs}, m$)
2: $\quad \mathrm{opt} \leftarrow \mathrm{true}$
3: $\quad \pi[v, sig] \leftarrow -\infty$ for all $v \in \mathcal{V}, sig \in \mathcal{S}$
4: $\quad \pi[v, 0] \leftarrow 0$ for all $v \in \mathcal{T}$
5: $\quad$ **for** $e \in \mathcal{E}$ in $\succ$ order **do**
6: $\quad\quad \langle \langle v_2, \ldots, v_k \rangle, v_1 \rangle \leftarrow e$
7: $\quad\quad$ **for** $sig^{(2)} \ldots sig^{(k)} \in \textsc{Sigs}(\langle v_2, \ldots, v_k \rangle)$ **do**
8: $\quad\quad\quad sig \leftarrow F\delta(e) + \sum_{i=2}^{k} sig^{(i)}$
9: $\quad\quad\quad s \leftarrow \theta_e + \sum_{i=2}^{k} \pi[v_i, sig^{(i)}]$
10: $\quad\quad\quad$ **if** $\begin{pmatrix} s > \pi[v_1, sig] \wedge \\ \textsc{Check}(sig) \wedge \\ s + \mathrm{ubs}[v_1] \geq \mathrm{lb} \end{pmatrix}$ **then**
11: $\quad\quad\quad\quad \pi[v_1, sig] \leftarrow s$
12: $\quad\quad\quad\quad$ **if** $\textsc{Prune}(v_1, sig, m)$ **then** $\mathrm{opt} \leftarrow$ false
13: $\quad \mathrm{lb}' \leftarrow \pi[1, c]$
14: $\quad$ **return** $\mathrm{lb}', \mathrm{opt}$

Input:

$(\mathcal{V}, \mathcal{E}, \theta)$	hypergraph with weights
(F, c)	matrix and vector for constraints
$\mathrm{lb} \in R$	lower bound
$\mathrm{ubs} \in R^{\lvert\mathcal{V}\rvert}$	upper bounds on outside scores
m	a pruning parameter

Output:

lb'	resulting lower bound score
opt	certificate of optimality

Figure 3-2: A variant of the beam search algorithm. Uses dynamic programming to produce a lower bound on the optimal constrained solution and, possibly, a certificate of optimality. Function SIGS enumerates all possible tail signatures. Function CHECK identifies signatures that do not violate constraints. Bounds lb and ubs are used to remove provably non-optimal solutions. Function PRUNE, taking parameter m, returns true if it prunes hypotheses from π that could be optimal.

vokes the black-box function, PRUNE, on line 12, passing it a pruning parameter m and a vertex-signature pair. The pruner returns true if it prunes from the chart. Note that pruning may remove optimal hypotheses, so we set the certificate flag opt to false if the chart is modified.

This variant on beam search satisfies the following two properties (recall (x^*, y^*) is the optimal constrained solution)

Property 3.3.1 (Primal Feasibility)**.** *The returned score* lb' *lower bounds the optimal constrained score, that is* $\mathrm{lb}' \leq \theta^\top y^*$.

Property 3.3.2 (Dual Certificate)**.** *If beam search returns with* $\mathrm{opt} = \mathrm{true}$*, then the returned score*

procedure PRUNE(v, sig, m)
$\mathcal{B} \leftarrow \{(u, sig') : ||sig'||_1 = ||sig||_1,$
$\pi[u, sig'] \neq -\infty\}$
$\mathcal{P} \leftarrow \mathcal{B} \setminus \text{mBEST}(m, \mathcal{B}, \pi)$
$\pi[u, sig'] \leftarrow -\infty$ for all $u, sig' \in \mathcal{P}$
if $\mathcal{P} = \emptyset$ **then return** true
else return false

Input: $\begin{bmatrix} (v, sig) & \text{the last hypothesis added to the chart} \\ m \in Z^+ & \text{\# of hypotheses to retain} \end{bmatrix}$
Output: true, if π is modified

Figure 3-3: Pruning function for phrase-based translation. Set $\mathcal{B}$ contains all hypotheses with $||sig||_1$ source words translated. The function prunes all but the top-m scoring hypotheses in this set.

is optimal, i.e. $\text{lb}' = \theta^\top y^*$.

An immediate consequence of Property 3.3.1 is that the output of beam search, lb', can be used as the input lb for future runs of the algorithm. Furthermore, if we loosen the amount of beam pruning by adjusting the parameter m we can produce tighter lower bounds and discard more hypotheses. We can then iteratively apply this idea with a sequence of parameters $m_1 \dots m_K$ producing lower bounds $\text{lb}^{(1)}$ through $\text{lb}^{(K)}$. We return to this idea in Section 3.5.

Example: Phrase-based Beam Search. Recall that the constraints for phrase-based translation consist of a binary matrix $F \in \{0, 1\}^{|s| \times |\mathcal{E}|}$ and vector $c = \mathbf{1}$. The value sig_i is therefore the number of times source word i has been translated in the hypothesis. The predicate $\text{CHECK}(sig) = (sig \leq \mathbf{1})$ removes hypotheses that translate source words more than once. For this reason, phrase-based signatures are called *bit-strings*.

A common beam pruning strategy is to group together items into a set $\mathcal{B}$ and retain a (possibly complete) subset. An example phrase-based beam pruner is given in Figure 3-3. It groups together hypotheses based on $||sig_i||_1$, i.e. the number of source words translated, and applies a hard pruning filter that retains only the m highest-scoring items $(v, sig) \in \mathcal{B}$ based on $\pi[v, sig]$.

3.4 Finding Tighter Bounds with Lagrangian Relaxation

Beam search produces a certificate only if beam pruning is never used. In the case of phrase-based translation, the certificate is dependent on all groups $\mathcal{B}$ having less than $m+1$ items. The only way to ensure this is to bound out enough hypotheses to avoid pruning. The effectiveness of the bounding inequality, $s+\mathrm{ubs}[v] < \mathrm{lb}$, in removing hypotheses is directly dependent on the tightness of the bounds lb and ubs.

In this section we propose using Lagrangian relaxation to get tight upper bounds ubs. We first give a brief overview of the method and then apply it to computing bounds. Our experiments show that these upper bounds are very effective at finding certificates in practice.

3.4.1 Algorithm

In Lagrangian relaxation, instead of solving the constrained search problem, we relax the constraints and solve an unconstrained hypergraph problem with modified weights. Recall the constrained hypergraph problem: $\max_{(x,y)\in\mathcal{X}:Fy=c} \theta^\top y$. The Lagrangian dual of this optimization problem is

$$\begin{aligned} L(\lambda) &= \max_{(x,y)\in\mathcal{X}} \theta^\top y - \lambda^\top (Fy - c) \\ &= \left(\max_{(x,y)\in\mathcal{X}} (\theta^\top - \lambda^\top F) y \right) + \lambda^\top c \end{aligned}$$

where $\lambda \in R^{|c|}$ is a vector of dual variables. This maximization is over $\mathcal{X}$, so for any value of λ, $L(\lambda)$ can be calculated using dynamic programming.

The Lagrangian dual has the following two properties, where $(x,y) \in \mathcal{X}$ is the hyperpath computed within the max

Property 3.4.1 (Dual Feasibility)**.** *The value $L(\lambda)$ upper bounds the optimal solution, that is $L(\lambda) \geq \theta^\top y^*$*

Property 3.4.2 (Primal Certificate)**.** *If the hyperpath (x,y) is a member of $\mathcal{X}'$, i.e. $Fy = c$, then*

procedure LRROUND(α_k, λ)
$(x, y) \leftarrow \arg \max_{(x,y) \in \mathcal{X}} \theta^\top y - \lambda^\top (Fy - c)$
$\lambda' \leftarrow \lambda - \alpha_k (Fy - c)$
$\text{opt} \leftarrow Fy = c$
$\text{ub} \leftarrow \theta^\top y$
return $\lambda, \text{ub}, \text{opt}$
procedure LAGRANGIANRELAXATION(α)
$\lambda^{(0)} \leftarrow 0$
for k in $1 \dots K$ **do**
$\lambda^{(k)}, \text{ub}, \text{opt} \leftarrow \text{LRROUND}(\alpha_k, \lambda^{(k-1)})$
if opt **then return** $\lambda^{(k)}, \text{ub}, \text{opt}$
return $\lambda^{(K)}, \text{ub}, \text{opt}$

Input: $\alpha_1 \dots \alpha_K$ sequence of subgradient rates

Output:
- λ final dual vector
- ub upper bound on optimal constrained solution
- opt certificate of optimality

Figure 3-4: Lagrangian relaxation algorithm. The algorithm repeatedly calls LRROUND to compute the subgradient, update the dual vector, and check for a certificate.

$L(\lambda) = \theta^\top y^*$.

Property 3.4.1 states that $L(\lambda)$ always produces some upper bound; however, to help beam search, we want as tight an bound as possible: $\min_\lambda L(\lambda)$.

The Lagrangian relaxation algorithm, shown in Figure 3-4, uses subgradient descent to find this minimum. The subgradient of $L(\lambda)$ is $Fy - c$ where y is the argmax of the modified objective $(x, y) = \arg\max_{(x,y) \in \mathcal{X}} \theta^\top y - \lambda^\top (Fy - c)$. Subgradient descent iteratively solves unconstrained hypergraph search problems to compute these subgradients and updates λ. See Rush and Collins (2012) for an extensive discussion of this style of optimization in natural language processing.

Example: Phrase-based Relaxation. For phrase-based translation, we expand out the Lagrangian to

$$\begin{aligned} L(\lambda) &= \max_{(x,y) \in \mathcal{X}} \theta^\top y - \lambda^\top (Fy - c) \\ &= \max_{(x,y) \in \mathcal{X}} \sum_{e \in \mathcal{E}} (\theta_e - \sum_{i=j(p(e))}^{k(p(e))} \lambda_i) y_e + \sum_{i=1}^{|s|} \lambda_i \end{aligned}$$

The weight of each edge θ_e is modified by the dual variables λ_i that correspond to each source word used by the phrase $p(e)$, $j(p(e)) \leq i \leq k(p(e))$. The relaxed solution is the best unconstrained derivation under this new set of weights. This solution may use source words multiple times or not at all. However if the solution uses each source word exactly once ($Fy = \mathbf{1}$), then we have a certificate and the solution is optimal.

3.4.2 Computing Upper Bounds

The table $\mathrm{ubs}[v]$ passed to BEAMSEARCH must contain an overestimate of the outside score for all vertices $v \in \mathcal{V}$. Define the set $\mathcal{O}(v)$ to contain all outside edges of vertex v (informally, edges that cannot appear in a hypothesis ending in v). It is required that, for all $v \in \mathcal{V}$,

$$\max_{(x,y) \in \mathcal{X}' : x_v = 1} \sum_{e \in \mathcal{O}(v)} \theta_e y_e \leq \mathrm{ubs}[v]$$

Solving this constrained maximization is difficult; however, we can again relax the constraint and compute the Lagrangian dual. Property 3.4.1 implies that for any dual vector λ,

$$\max_{(x,y) \in \mathcal{X}' : x_v = 1} \sum_{e \in \mathcal{O}(v)} \theta_e y_e \leq$$
$$\max_{(x,y) \in \mathcal{X} : x_v = 1} \sum_{e \in \mathcal{O}(v) : y_e = 1} (\theta^\top - \lambda^T F)\delta(e) + \lambda^T c$$

We take this last term as the upper bound for all v.

If we have a fixed dual vector λ, these upper bounds can be efficiently computed for all vertices using the standard outside dynamic programming algorithm with modified weights. We will refer to this algorithm as DUALOUTSIDE(λ).

3.5 Optimal Beam Search

Whether the beam search algorithm can return an optimal solution or not is dependent on the tightness of the upper and lower bounds. We can produce better lower bounds by varying the pruning parameter m; we can produce better upper bounds by running Lagrangian relaxation and computing outside scores based on λ. In this section we combine these two ideas and present a complete optimal beam search algorithm.

One simple method for computing bounds, shown at the top of Figure 3-5, is to compute bounds in stages. The algorithm first runs Lagrangian relaxation to compute the best λ vector. If, in the process, it finds a primal certificate, it returns the result. The algorithm then iteratively runs beam search using the parameter sequence m_k. These parameters allow the algorithm to loosen the amount of beam pruning until it finds an optimal solution. For example in phrase based pruning, we would raise the number of hypotheses stored per group until no beam pruning occurs.

The disadvantage of the staged approach is that it needs to wait until Lagrangian relaxation is completed before even running beam search. Often beam search will be able to quickly find an optimal solution even with good but non-optimal λ. In the worst case, beam search may still improve the lower bound lb.

This observation motivates the alternating algorithm OPTBEAM shown Figure 3-5. In each round, the algorithm alternates between computing subgradients to tighten ubs and running beam search to maximize lb. In early rounds we set m for aggressive beam pruning, and as the upper bounds get tighter, we loosen pruning to try to get a certificate. If at any point either a primal or dual certificate is found, the algorithm returns the optimal solution.

3.6 Related Work

Approximate methods based on beam search and cube-pruning have been widely studied for phrase-based Koehn et al. (2003); Tillmann and Ney (2003); Tillmann (2006) and syntax-based translation models Chiang (2007); Huang and Chiang (2007); Watanabe et al. (2006); Huang and

procedure OPTBEAMSTAGED(α, m)
  $\lambda, \text{ub}, \text{opt} \leftarrow$LAGRANGIANRELAXATION($\alpha$)
  if opt **then return** ub
  $\text{ubs} \leftarrow$ DUALOUTSIDE(λ)
  $\text{lb}^{(0)} \leftarrow -\infty$
  for k in $1 \dots K$ **do**
    $\text{lb}^{(k)}, \text{opt} \leftarrow$ BEAMSEARCH($\text{ubs}, \text{lb}^{(k-1)}, m_k$)
    if opt **then return** $\text{lb}^{(k)}$
  return $\max_{k \in \{1 \dots K\}} \text{lb}^{(k)}$

procedure OPTBEAM(α, m)
  $\lambda^{(0)} \leftarrow 0$
  $\text{lb}^{(0)} \leftarrow -\infty$
  for k in $1 \dots K$ **do**
    $\lambda^{(k)}, \text{ub}^{(k)}, \text{opt} \leftarrow$ LRROUND($\alpha_k, \lambda^{(k-1)}$)
    if opt **then return** $\text{ub}^{(k)}$
    $\text{ubs}^{(k)} \leftarrow$ DUALOUTSIDE($\lambda^{(k)}$)
    $\text{lb}^{(k)}, \text{opt} \leftarrow$ BEAMSEARCH($\text{ubs}^{(k)}, \text{lb}^{(k-1)}, m_k$)
    if opt **then return** $\text{lb}^{(k)}$
  return $\max_{k \in \{1 \dots K\}} \text{lb}^{(k)}$

Input: $\begin{bmatrix} \alpha_1 \dots \alpha_K & \text{sequence of subgradient rates} \\ m_1 \dots m_K & \text{sequence of pruning parameters} \end{bmatrix}$
Output: optimal constrained score or lower bound

Figure 3-5: Two versions of optimal beam search: staged and alternating. Staged runs Lagrangian relaxation to find the optimal λ, uses λ to compute upper bounds, and then repeatedly runs beam search with pruning sequence $m_1 \dots m_k$. Alternating switches between running a round a Lagrangian relaxation and a round of beam search with the updated λ. If either the Lagrangian relaxation or the beam search produces a certificate, it returns the result.

Mi (2010).

There is also a line of work studying exact algorithms for machine translation decoding. Often times exact decoders are slow to use in practice, but help quantify the errors made by approximate methods. Exact algorithms proposed for IBM model 4 include ILP Germann et al. (2001), cutting plane Riedel and Clarke (2009), and multi-pass A* search Och et al. (2001). Zaslavskiy et al. (2009) formulate phrase-based decoding as a traveling salesman problem (TSP) and use a TSP decoder. Exact decoding algorithms based on finite state transducers (FST) Iglesias et al. (2009) have been studied on phrase-based models with limited reordering Kumar and Byrne (2005b). Exact decoding based on FST is also feasible for certain hierarchical grammars de Gispert et al. (2010).

Chang and Collins (2011) and Rush and Collins (2011) develop Lagrangian relaxation-based approaches for exact machine translation. Both algorithms rely on involved tightening procedures to recover the exact solution. The method presented in this chapter can be seen as an alternative to this process.

Apart from translation decoding, this chapter is closely related to work on column generation for NLP. Riedel et al. (2012) and Belanger et al. (2012) relate column generation to beam search and produce exact solutions for parsing and tagging. They also discuss the relationship between beam width and duality gap.

3.7 Results

To evaluate the effectiveness of optimal beam search for translation decoding, we implemented decoders for phrase- and syntax-based models. In this section we compare the speed and optimality of these decoders to several baseline methods.

3.7.1 Setup and Implementation

For phrase-based translation we used a German-to-English data set taken from Europarl (Koehn, 2005). We tested on 1,824 sentences of length at most 50 words. For experiments the phrase-based model uses a trigram language model and also includes distortion penalties. Additionally the unconstrained hypergraph includes additional derivation information similar to the graph described in Chang and Collins (2011).

For syntax-based translation we used a Chinese-to-English data set. The model and hypergraphs come from the work of Huang and Mi (2010). We tested on 691 sentences from the newswire portion of the 2008 NIST MT evaluation test set. For experiments, the syntax-based model uses a trigram language model. The translation model is tree-to-string syntax-based model with a standard context-free translation forest. The constraint matrix F is based on the constraints described by Rush and Collins (2011).

	11-20 (558)			21-30 (566)			31-40 (347)			41-50 (168)			all (1824)		
	time	cert	exact	time	cert	exact	time	cert	exact	time	cert	exact	time	cert	exact
BEAM (100)	2.33	19.5	38.0	8.37	1.6	7.2	21.59	0.3	1.4	47.06	0.0	0.0	11.82	15.3	23.2
BEAM (1000)	2.33	37.8	66.3	8.42	3.4	18.9	21.60	0.6	3.2	53.99	0.6	1.2	12.44	22.6	36.9
MOSES (100)	0.18	0.0	81.0	0.36	0.0	45.6	0.53	0.0	14.1	0.74	0.0	6.0	0.34	0.0	52.3
MOSES (1000)	2.29	0.0	97.8	4.39	0.0	78.8	6.52	0.0	43.5	9.00	0.0	19.6	4.20	0.0	74.6
ASTAR (cap)	11.02	99.3	99.3	91.59	53.9	53.9	124.9	7.8	7.8	264.7	1.2	1.2	86.99	58.8	58.8
LR-TIGHT	4.20	100.0	100.0	23.25	100.0	100.0	88.16	99.7	99.7	377.9	97.0	97.0	60.11	99.7	99.7
OPTBEAM	2.85	100.0	100.0	10.33	100.0	100.0	28.29	100.0	100.0	84.34	97.0	97.0	17.27	99.7	99.7
ChangCollins	10.90	100.0	100.0	57.20	100.0	100.0	203.4	99.7	99.7	679.9	97.0	97.0	120.9	99.7	99.7
MOSES-GC (100)	0.14	0.0	89.4	0.27	0.0	84.1	0.41	0.0	75.8	0.58	0.0	78.6	0.26	0.0	84.9
MOSES-GC (1000)	1.33	0.0	89.4	2.62	0.0	84.3	4.15	0.0	75.8	6.19	0.0	79.2	2.61	0.0	85.0

Table 3.1: Experimental results for phrase-based translation. Column *time* is the mean time per sentence in seconds, *cert* is the percentage of sentences solved with a certificate of optimality, *exact* is the percentage of sentences solved exactly, i.e. $\theta^\top y = \theta^\top y^*$. Results are grouped by sentence length (group 1-10 is omitted for space).

	11-20 (192)			21-30 (159)			31-40 (136)			41-100 (123)			all (691)		
	time	cert	exact	time	cert	exact	time	cert	exact	time	cert	exact	time	cert	exact
BEAM (100)	0.40	4.7	75.9	0.40	0.0	66.0	0.75	0.0	43.4	1.66	0.0	25.8	0.68	5.72	58.7
BEAM (1000)	0.78	16.9	79.4	2.65	0.6	67.1	6.20	0.0	47.5	15.5	0.00	36.4	4.16	12.5	65.5
CUBE (100)	0.08	0.0	77.6	0.16	0.0	66.7	0.23	0.0	43.9	0.41	0.0	26.3	0.19	0.0	59.0
CUBE (1000)	1.76	0.0	91.7	4.06	0.0	95.0	5.71	0.0	82.9	10.69	0.0	60.9	4.66	0.0	85.0
LR-TIGHT	0.37	100.0	100.0	1.76	100.0	100.0	4.79	100.00	100.00	30.85	94.5	94.5	7.25	99.0	99.0
OPTBEAM	0.25	100.0	100.0	0.59	99.4	99.4	2.08	98.5	98.5	6.56	92.7	92.7	1.82	98.2	98.2
ILP	9.15	100.0	100.0	32.35	100.0	100.0	49.6	100.0	100.0	108.6	100.0	100.0	40.1	100.0	100.0

Table 3.2: Experimental results for syntax-based translation. See Table 3.1 for column descriptions.

Our decoders use a two-pass architecture. For each sentence the first pass constructs the hypergraph in memory, and the second pass runs search. Wherever possible the models share optimized construction and search code.

The performance of optimal beam search is dependent on the sequences α and m. For the step-size α we used a variant of Polyak's rule Polyak (1987); Boyd and Mutapcic (2007), substituting the unknown optimal score for the last computed lower bound: $\alpha_k \leftarrow \frac{\theta^\top y^{(k)} - \mathrm{lb}^{(k)}}{||Fy^{(k)} - c||_2^2}$. We adjust the order of the pruning parameter m based on a function μ of the current duality gap: $m_k \leftarrow 10^{\mu(\mathrm{ub}^{(k)} - \mathrm{lb}^{(k)})}$.

Previous work on these data sets has shown that exact algorithms do not result in a significant increase in translation accuracy. We focus on the efficiency and model score of the algorithms.

3.7.2 Baseline Methods

We compare optimal beam search (OPTBEAM) to several different decoding methods. For both systems we compare to: BEAM, the beam search decoder from Figure 3-2 using outside scores generated with $\lambda = 0$ and $m \in \{100, 1000\}$; LR-TIGHT, Lagrangian relaxation followed by incremental tightening constraints, which is a reimplementation of Chang and Collins (2011) and Rush and Collins (2011).

For phrase-based translation we compare with: MOSES-GC, the standard Moses beam search decoder with $m \in \{100, 1000\}$ Koehn et al. (2007); MOSES, a version of Moses without gap constraints more similar to BEAM; ASTAR, an implementation of A* search using future scores generated with $\lambda = 0$, and capped at 20,000,000 queue pops.

For syntax-based translation we compare with: ILP, a general-purpose integer linear programming solver Gurobi Optimization (2013) and CUBEPRUNING, an approximate decoding method similar to beam search Chiang (2007), tested with $m \in \{100, 1000\}$.

3.7.3 Experiments

Table 3.1 shows the main results for phrase-based translation. OPTBEAM decodes the optimal translation with certificate in 99% of sentences with an average time of 17.27 seconds per sentence. This speed is seven times faster than the decoder of Chang and Collins (2011) and 3.5 times faster then our reimplementation, LR-TIGHT. ASTAR performs poorly, taking lots of time on a few difficult sentences. BEAM is fast, but rarely finds an exact solution. MOSES is very fast, but less exact than optimal beam search and unable to produce a certificate.

Table 3.2 shows the main results for syntax-based translation. OPTBEAM finds a certificate on 98% of solutions with an average time of 1.82 seconds per sentence, and is four times faster than LR-TIGHT. CUBE (100) is an order of magnitude faster, but is rarely exact on longer sentences. CUBE (1000) finds more exact solutions, but is comparable in speed to optimal beam search. BEAM performs better than in the phrase-based model, but is not much faster than OPTBEAM.

Table 3.3 breaks down the amount of time spent in each part of the algorithm. For both meth-

		≥ 30		all	
		mean	median	mean	median
PB	Hypergraph	56.6%	69.8%	59.6%	69.6%
	Lag. Relaxation	10.0%	5.5%	9.4%	7.6%
	Beam Search	33.4%	24.6%	30.9%	22.8%
SB	Hypergraph	0.5%	1.6%	0.8%	2.4%
	Lag. Relaxation	15.0%	35.2%	17.3%	41.4%
	Beam Search	84.4%	63.1%	81.9 %	56.1%

Table 3.3: Distribution of time within optimal beam search, including: hypergraph construction (including language model), Lagrangian relaxation, and beam search. *Mean* is the percentage of total time. *Median* is the distribution over the median values for each row.

ods, beam search has the most time variance. Beam search also uses more time on longer sentences. For phrase-based sentences, Lagrangian relaxation is fast, and hypergraph construction dominates. If not for this cost, OPTBEAM would be comparable in speed to MOSES (1000).

3.8 Conclusion

In this work we develop an optimal variant of beam search and apply it to machine translation decoding. The algorithm exploits the fact that beam search produces constrained solutions, and uses bounds from Lagrangian relaxation to eliminate non-optimal solutions. Results show that this form of optimal beam search is able to find exact solutions significantly faster than other exact methods.

This chapter utilizes Lagrangian relaxation to build an variant of beam search. The algorithm uses beam search to give valid solutions and take advantage of the upper bound obtained from Lagrangian relaxation to prune the search space. The pruning becomes more and more effective when the upper bound from Lagrangian relaxation becomes tighter. Combined with the optimality-preserving pruning, the beam search is able to find the optimal solution.

The same idea that utilizes the bounding property of the Lagrangian relaxation to prune the search space can be applied to other search method, such as A^* search. The drawback of A^* search is that it terminates only when it finds the optimal solution. The process might be time consuming when the upper bound given by Lagrangian relaxation is not tight enough. Whereas beam search

has the advantage of trying it quickly. One obvious modification to A^* search is to cap the size of the queue storing all the hypotheses. The modified A^* search can be described as a variant of beam search that has only one beam.

One reason that we choose beam search is because we would like to search by building the partial translation from left to right, which is how the core dynamic programming algorithm of the Lagrangian relaxation works. This way we can utilize the upper bound to prune the search space effectively.

Chapter 4

A Local Search Algorithm

In this chapter, we propose to use local search to approach the decoding problem for phrase-based translation models. Local search is a heuristic method that does not provide any guarantee on the solution. However, local search can be appealing due to its ability to work with a more complex model. For example, when applying Lagrangian relaxation, the construction discussed in Section 2.4 is difficult to scale to higher order language model because of the size of dynamic programming (Section 2.4.1) will increase exponentially with the order of the language model. Whereas computing a four-gram language model is of the same order of complexity as computing a trigram language model when it is computed for the whole sentence. Hence, local search can easily a higher order language model without adding too much computation cost.

4.1 Introduction

The design of a machine translation model depends on several factors. In addition to translation accuracy, we have to consider the efficiency of the decoding problem. The goal of the decoding problem is to find the highest scoring translation in the target language, which in term involves the scoring function and a search method. The design of the translation model affects which search methods can be applied efficiently. On the other hand, we can also say that the search method we employ decides the restriction on the scoring function. For example, dynamic-programming based

methods, which are easily extended to the widely-used method of beam search, require the scoring function to be decomposable according to the dynamic programming states.

In this work, we propose a dependency language model that aims to improve translation quality. We incorporate the dependency language model as one more component to the log-linear combination of scores in a phrase-based translation model. We first parse a complete sentence into a dependency tree and use the head-modifier relations. The trigram dependency language model looks at either the previous sibling or the grandparent of the current word. Then, we use local search as our method to optimize the model score.

The work by Langlais et al. (2007) discussed using local search as the optimization method in the decoding process. The proposed local search algorithm is a greedy method that scores a whole sentence at a time and moves toward the higher scoring translation until no improvement can be made. In a local search algorithm, we always start with a valid derivation, and at each iteration, we maintain a valid derivation, which includes a complete target-language sentence. Then we search the neighborhood for a higher scoring translation. A neighborhood is defined to be a set of valid translations that can be reached by the current translation via taking a local step. The operations consist of swapping, moving, splitting, merging and re-translating one or two phrases. A detailed description of the local steps will be given in Section 4.5.1.

We use BLEU score, as well as a syntactic evaluation metric to measure the translation quality. Adding the dependency language model does not affect the BLEU score significantly; however, we do observe a significant improvement in the syntactic evaluation metric.

The rest of this chapter is structured as follows. In Section 4.2 we survey related works on local search and syntactic phrase-based translation models. We introduce phrase-based translation models in Section 4.3, and detail the dependency language model in Section 4.4. Section 4.5 gives the local search algorithm. In Section 4.6, we describe the syntactic evaluation metric. Section 4.7 shows the experimental results.

4.2 Related Work

The local search algorithm used in our work is based on the work by Langlais et al. (2007), who proposed a local search algorithm for the phrase-based system. We use a similar local search algorithm and the same set of operations. The difference is that we incorporate a dependency language model to improve the translation quality, whereas they focus on the study of local search on phrase-based translation models. They used a reversed n-gram language model to demonstrate that local search can work with a non-incremental scoring function, but the improvement is not very significant.

Previously, a greedy decoding algorithm has been studied for decoding of word-based models by Germann et al. (2001), and a faster approach was discussed in Germann (2003).

Local search has also been applied to document-wide decoding of the phrase-based translation model (Hardmeier et al., 2012). The proposed operations are very similar to other work.

There are also works that apply local search to other natural language processing (NLP) tasks. Zhang et al. (2014) use local search for dependency parsing, and Marcu and Wong (2002) use it for translation alignment. Zhang et al. (2015) use randomized greedy inference for joint segmentation, POS tagging and dependency parsing.

There are various efforts of incorporating the syntactic information into phrase-based translation models. To avoid the requirement of a complete target-language sentence, various approaches have been proposed and studied: string-to-tree (Yamada and Knight, 2001; Huang et al., 2006; Shen et al., 2008), tree-to-string (Liu et al., 2006; Huang and Mi, 2010), tree-to-tree approach (Cowan et al., 2006; DeNeefe and Knight, 2009; Chiang, 2010), incremental syntactic language model (Schwartz et al., 2011)

In our work, we use an evaluation metric proposed by Liu and Gildea (2005). It is a modified BLEU score on head-word chains in a dependency tree. Several other works proposed syntax-based automatic evaluation measures for machine translation. Popović and Ney (2009) proposed BLEU score, precision, recall and F-measure on part-of-speech (POS) tags and on words and POS tags together. Mehay and Brew (2006) used a Combinatory Categorial Grammar (CCG) parser to

extract word-word dependencies.

4.3 The Phrase-based Translation Model

In phrase-based translation models, the basic unit is a phrase pair consisting of a source-language phrase and a target-language phrase. A phrase is defined to be any continuous span of words. The goal of the decoding problem of the phrase-based translation model is to find the highest scoring translation in the target-language, given a source-language sentence. Here we define $x = x_1 \dots x_N$ to be the source-language sentence that consists of N words. Each x_i represents the ith word. We also define a phrase pair p to be a tuple of (s, t, e) where s and t are indices into the source-language sentence, and e is a target-language phrase that $x_s \dots x_t$ can be translated into. We use $s(p), t(p), e(p)$ to refer to the three components of phrase pair p. We describe a derivation to be a sequence of L phrase pairs that covers the all words in the source-language sentence once:

$$y = \langle p_1 \dots p_L \rangle.$$

The translated target-language sentence can be represented by $e(y)$, and we use $|e(y)|$ to denote the number of words in the target-language sentence.

A scoring function is used to score the derivation. The usual phrase-based model has three parts: the translation model, the language model and the distortion model.

$$f(y) = \mu_t \sum_{l=1}^{L} g_t(p_l) + \mu_l g_l(e(y)) + \mu_b |e(y)| + \mu_\delta \sum_{l=1}^{L-1} g_\delta(t(p_l), s(p_{l+1}))$$

We use g_t to represent the translation score, g_l the language model score, and g_δ the distortion score. Each component has weight μ.

The decoding problem is to find the highest scoring derivation for a given scoring function f, within the set of valid derivations:

$$y^* = \max_{y \in \mathcal{Y}} f(y),$$

where $\mathcal{Y}$ is the set of valid derivations. Here we define a derivation to be valid if each source-language word is translated exactly once and the distortion limit is satisfied. The distortion limit is a hard constraint on how phrases can be reordered.

4.3.1 The Proposed Scoring Function

Using local search gives us the flexibility to use a scoring function that takes the whole sentence into consideration. Unlike optimization methods based on dynamic programming, we will not require the scoring function to be decomposable.

Our scoring function consists of the usual components of the phrase-based translation models: the translation scores, the language model, and the distortion penalties. In addition, we add a dependency language model to allow scoring of long-distance syntactic relationships. We define the scoring function to be

$$\hat{f}(y) = f(y) + \mu_d g_d(y),$$

where $g_d(y)$ is the dependency language model score that will be defined the Section 4.4

4.4 The Dependency Language Model

In order to include the syntactic structure in the scoring function, we utilize a trigram dependency language model. For modifiers adjacent to its head, the model looks at the grandparent, and for non-adjacent modifiers, it looks at the previous sibling. The dependency language model is built upon tags and an unlabeled dependency tree of the sentence.

We first parse the translated target-language sentence into a dependency tree using an existing parser. We use the notation $d(y)$ to represent the parse tree derived from a derivation y. The dependency tree can be viewed as a set of head-modifier pairs (h, m), where h is the head and m is the modifier. Every word in the sentence will appear as modifier in exactly one head-modifier relation.

In our model, we consider several features including direction, adjacency to the head, the head

word and its tag, the previous sibling word and its tag, and occasionally the grandparent word (head of the head word) and its tag. We only look at the grandparent when the current word is adjacent to the head. In that situation, there is no previous sibling. We use a four-layer back-off model that eventually goes to the probability of the word.

The dependency language model score of a sentence is the sum of log probabilities of each word under the model.

$$g_d(y) = \sum_{(h,m) \in d(y)} \log(p(w_m, t_m | w_h, t_h, dir, adj, w_p, t_p))$$

We define the following symbols:

- dir: whether the modifier is on the left (L) or on the right (R)
- adj: whether the modifier is adjacent to the head or not (True, False)
- w_h: the head word
- t_h: the tag of the head word
- w_m: the modifier word
- t_m: the tag of the modifier word
- w_p: $\begin{cases} \text{the previous sibling word} & \text{if } adj \text{ is False} \\ \text{the grandparent word (head of the head)} & \text{if } adj \text{ is True} \end{cases}$
- t_p: $\begin{cases} \text{the tag of the previous sibling word} & \text{if } adj \text{ is False} \\ \text{the tag of the grandparent word} & \text{if } adj \text{ is True} \end{cases}$

Figure 4-1 shows an example dependency tree. The dependency language model score of this sentence will be as follows (for brevity we omit probabilities involving the STOP symbol, which

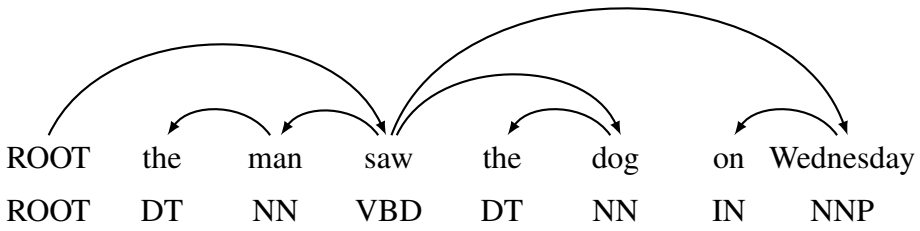


Figure 4-1: An example dependency tree of the sentence: the man saw a dog on Wednesday.

is always the final modifier to any head):

$$
\begin{aligned}
& p(\mathsf{the}, \mathsf{DT} \mid man, NN, L, True, saw, VBD) \\
\times & p(\mathsf{man}, \mathsf{NN} \mid \mathsf{saw}, \mathsf{VBD}, L, True, \mathsf{ROOT}, \mathsf{ROOT}) \\
\times & p(\mathsf{saw}, \mathsf{ROOT} \mid \mathsf{ROOT}, \mathsf{ROOT}, R, True, \mathsf{NULL}, \mathsf{NULL}) \\
\times & p(\mathsf{the}, \mathsf{DT} \mid \mathsf{dog}, \mathsf{NN}, L, True, \mathsf{saw}, \mathsf{VBD}) \\
\times & p(\mathsf{dog}, \mathsf{NN} \mid \mathsf{saw}, \mathsf{VBD}, R, True, \mathsf{ROOT}, \mathsf{ROOT}) \\
\times & p(\mathsf{on}, \mathsf{IN} \mid \mathsf{Wednesday}, \mathsf{NNP}, L, True, \mathsf{saw}, \mathsf{VBD}) \\
\times & p(\mathsf{Wednesday}, \mathsf{NNP} \mid \mathsf{saw}, \mathsf{VBD}, R, False, \mathsf{dog}, \mathsf{NN})
\end{aligned}
$$

Similar to the works by Eisner (1996); Collins (1997), we decompose probability of each word-tag pair into the product of two terms using the chain rule.

$$
\begin{aligned}
& p(w_m, t_m | w_h, t_h, dir, adj, w_p, t_p) \\
= & \, p(\mathbf{w}_m | \mathbf{t}_m, w_h, t_h, dir, adj, w_p, t_p) \times p(\mathbf{t}_m | w_h, t_h, dir, adj, w_p, t_p)
\end{aligned}
$$

Following the back-off models in Collins (1997), we use a 4-level back-off model for smoothing. The conditioning variables for each level of back-off is summarized in Table 4.1. For the trigram probability estimate of the word given its tag, we use smoothing parameters λ_1, λ_2 and λ_3

Back-off level	$p(w_m\|\cdots)$	$p(t_m\|\cdots)$
1	$t_m, w_h, t_h, dir, adj, w_p, t_p$	$w_h, t_h, dir, adj, w_p, t_p$
2	t_m, w_h, t_h, dir, adj	w_h, t_h, dir, adj
3	t_m	t_h, dir, adj
4	–	–

Table 4.1: The conditioning variables for each level of back-off.

to produce the interpolated estimate.

$$
\begin{aligned}
p(w_m|t_m, w_h, t_h, dir, adj, w_p, t_p) =& \lambda_1 p_{ML}(w_m|t_m, w_h, t_h, dir, adj, w_p, t_p) + (1-\lambda_1)p(w_m|t_m, w_h, t_h, dir, adj) \\
p(w_m|t_m, w_h, t_h, dir, adj) =& \lambda_2 p_{ML}(w_m|t_m, w_h, t_h, dir, adj) + (1-\lambda_2)p(w_m|t_m) \\
p(w_m|t_m) =& \lambda_3 p_{ML}(w_m|t_m) + (1-\lambda_3)p(w_m) \\
p(w_m) =& p_{ML}(w_m)
\end{aligned}
$$

For the tag, we use smoothing parameters λ_4, λ_5 and λ_6 to interpolate the lower gram probability.

$$
\begin{aligned}
p(t_m|w_h, t_h, dir, adj, w_p, t_p) =& \lambda_4 p_{ML}(t_m|w_h, t_h, dir, adj, w_p, t_p) + (1-\lambda_4)p(t_m|w_h, t_h, dir, adj) \\
p(t_m|w_h, t_h, dir, adj) =& \lambda_5 p_{ML}(t_m|w_h, t_h, dir, adj) + (1-\lambda_5)p(t_m|t_h, dir, adj) \\
p(t_m|t_h, dir, adj) =& \lambda_6 p_{ML}(t_m|t_h, dir, adj) + (1-\lambda_6)p(t_m) \\
p(t_m) =& p_{ML}(t_m)
\end{aligned}
$$

Each λ_i is derived using the following method:

$$\lambda_{event} = \frac{count(event)}{count(event) + \lambda u(event)}$$

The $event$ for each λ_i is the conditioning variable for that level. It is described here:

$$\lambda_1 = \lambda_{(t_m, w_h, t_h, dir, adj, w_p, t_p)}$$

$$\lambda_2 = \lambda_{(t_m, w_h, t_h, dir, adj)}$$

$$\lambda_3 = \lambda_{(t_m)}$$

$$\lambda_4 = \lambda_{(w_h, t_h, dir, adj, w_p, t_p)}$$

$$\lambda_5 = \lambda_{(w_h, t_h, dir, adj)}$$

$$\lambda_6 = \lambda_{(t_h, dir, adj)}$$

For example,

$$\lambda_1 = \frac{count(t_m, w_h, t_h, dir, adj, w_p, t_p)}{count(t_m, w_h, t_h, dir, adj, w_p, t_p) + \lambda u(t_m, w_h, t_h, dir, adj, w_p, t_p)}$$

Note that $u(event)$ returns the number of unique word following event, and λ is a constant.

Also, p_{ML} is the maximum likelihood estimator of the probability, for example:

$$p_{ML}(w_m | t_m, w_h, t_h, dir, adj, w_p, t_p) = \frac{count(t_m, w_h, t_h, dir, adj, w_p, t_p, w_m)}{count(t_m, w_h, t_h, dir, adj, w_p, t_p)} \tag{4.1}$$

4.5 Local Search Algorithm

We use a local search algorithm with hill-climbing strategy. It starts from an initial state, and moves to a higher scoring neighboring state at each iteration, until there is no higher scoring state in its neighborhood. In our algorithm, each state is a valid derivation. A state can transition to another state by taking a local step. Section 4.5.1 gives the definition of all local steps we use. We can described each type of local step as a function that maps one state to a set of states. Each type of local steps defines a partial neighborhood.

Initialization: Get a initial derivation $y = p_1 \dots p_L$ using beam search with beam size 1000, with no dependency language model
for $t = 1 \dots T$
  $s = f(y)$
  for $\mathcal{N} \in \{\mathcal{N}_{Swap}, \mathcal{N}_{Move}, \mathcal{N}_{Split}, \mathcal{N}_{Merge}\}$
    for $y' \in \mathcal{N}(y)$
      Parse the complete sentence $e(y')$ into a tree $d(y')$
      if $f(y') > f(y)$
        $y \leftarrow y'$
  if $f(y) = s$
    break

Figure 4-2: The local search algorithm for phrase-based decoding.

In our algorithm, we explore the neighborhood and find the highest scoring state in the neighborhood. Each state is a complete derivation, and we need to score all states in the neighborhood. Each time a derivation is proposed, we need to parse it into a dependency tree and score it using the scoring function introduced in Section 4.3.1. If the score is higher than the current state, we transition to this new state. Otherwise, we have found a local maximum and the algorithm stops. The algorithm is described in Figure 4-2.

4.5.1 Local Steps

There are four types of local steps in our local search algorithm. They are swap, move, split, and merge. We assume that the current derivation we have is

$$y = \langle p_1 p_2 \dots p_L \rangle$$

We will describe each type of the local steps in the following sections. In each operation, there are one or two phrases involved. Note that for all operations, we consider the alternative target-language translations of the involved phrases.

4.5.1.1 Swap

With the Swap operation, we pick two phrases and swap their locations. The resulting derivation of swapping phrase i and phrase j can be defined as

$$y_{i:j} = \langle p_1 \dots p_{i-1} p_j p_{i+1} \dots p_{j-1} p_i p_{j+1} \dots p_L \rangle,$$

assuming that $i < j$.

4.5.1.2 Move

With the move operation, we pick one phrase and move it to another position. The resulting derivation of moving phrase i to the position of phrase j can be defined as

$$y_{i \to j} = \langle p_1 \dots p_{j-1} p_i p_j p_{j+1} \dots p_L \rangle.$$

4.5.1.3 Split

The split operation takes one phrase $p = (s, t, e)$ such that $t - s \geq 1$ and replaces it by two phrases $p' = (s, u, e')$ and $p'' = (u + 1, t, e'')$ such that $s \leq u$ and $u + 1 \leq t$. The resulting derivation is

$$y_{i:p'_i p''_i} = \langle p_1 \dots p_{i-1} p'_i p''_i p_{i+1} \dots p_L \rangle$$

4.5.1.4 Merge

The merge operation takes two neighboring phrases that are together covering a continuous span in the source-language and replace the two phrases by one phrase that cover the same span. Suppose the two neighboring phrases are at position i and $i + 1$: $p_i = (s, t, e)$ and $p_{i+1} = (s', t', e')$. There are two cases that p_i and p_{i+1} are covering the same span:

1. $s' - t = 1$
2. $s - t' = 1$

We replace them by a phrase $\hat{p} = (\hat{s}, \hat{t}, \hat{e})$ such that either of the following is true:

1. $\hat{s} = s$ and $\hat{t} = t'$ if $s' - t = 1$

2. $\hat{s} = s'$ and $\hat{t} = t$ if $s - t' = 1$

The resulting derivation is

$$y_{i,i+1:\hat{p}} = \langle p_1 \dots p_{i-1} \hat{p} p_{i+2} \dots p_L \rangle.$$

4.5.2 Neighborhood

The neighborhood $\mathcal{N}(y)$ of the currently derivation y is a set of valid derivations that can be reached by local steps. We define four types of neighborhood according to the four types of local steps. In our local search algorithm, we iterate through the four types of neighborhood. Each time we move to the highest scoring state in one neighborhood. We stop iterating when there is no improvement in all four types of neighborhood.

Each neighborhood contains only the valid derivation, and we use a function $\textsf{feasible}(y)$ to check if a new derivation y satisfies the distortion limit. The partial neighborhood that is defined by each local step can be defined as follows:

- $\mathcal{N}_{swap}(y) = \{y_{i:j} : i \in \{1 \dots L\}, j \in \{i+1 \dots L\},\ \textsf{feasible}(y_{i:j})\}$

- $\mathcal{N}_{move}(y) = \{y_{i \to j} : i, j \in \{1 \dots L,\ \textsf{feasible}(y_{i \to j})\}$

- $\mathcal{N}_{split}(y) = \{y_{i:p'_i p''_i} : t(p_i) - s(p_i) \geq 1,\ p'_i = (s(p_i), u, e'),\ p''_i = (u+1, t(p_i), e'')\ \forall i \in \{1 \dots L\},\ \textsf{feasible}(y_{i:p'_i p''_i})\}$

- $\mathcal{N}_{merge}(y) = \{y_{i,i+1:\hat{p}}$: constraints in Section 4.5.1.4 satisfied $\forall\ i \in \{1 \dots L-1\},\ \textsf{feasible}(y_{i,i+1:\hat{p}})\}$

4.6 Dependency BLEU score

In addition to BLEU score (Papineni et al., 2002), we use HWCM (Head-Word Chain based Metric), an evaluation metric proposed by Liu and Gildea (2005), to evaluate the translation quality. HWCM is a dependency based BLEU score that reflects the similarity of the syntactic structures between the proposed translation and the reference translation. We modified it slightly to incorporate the brevity penalty (BP) as in the computation of BLEU score. We will use HWCM$_{BLEU}$ to denote it.

HWCM replaces the word n-gram by dependency n-gram. The dependency n-gram are determined by tracing the head-modifier relations in the dependency tree. The unigram consists of single words, the same as in the usual language model. The bigram, which is a 2-word head word chain, is defined to be the head-modifier pair. The trigram, which is a 3-word headword chain, is a triple of grandparent, head, and modifier, where grandparent is the head of the head. The four-gram (4-word headword chain) is a tuple of grand-grandparent, grandparent, head, and modifier. If for any word, the length of a chain is less than the desired length, we use NULL to fill it. Figure 4-3 shows an example of a dependency tree and lists the 4-word headword chains.

The sentences in both the proposed translation and the reference translation are parsed into dependency trees. The headword chains for both trees are extracted. The rest of the computation of the score is the same as the original BLEU score. In our experiment, the maximum length of the headword chains is four. Let p_n be the modified dependency n-gram precision, then the HWCM$_{BLEU}$ score is defined as follows:

$$\text{HWCM}_{BLEU} = BP \times \exp\left(\sum_{n=1}^{4} \frac{1}{4} \log p_n\right),$$

where $BP = \exp(1 - \frac{\text{reference translation length}}{\text{proposed translation length}})$

Following the original BLEU score, the modified precision score p_n only count the smaller

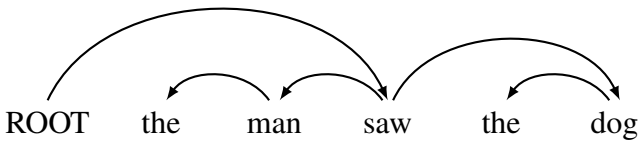


Figure 4-3: An example dependency tree. The 4-word headword chains are (ROOT, saw, man, the), (NULL, ROOT, saw, man), (NULL, NULL, ROOT, saw), (ROOT, saw, dog, the), (NULL, ROOT, saw, dog).

number of the appearance of a token as hit.

$$p_n = \frac{\min(count(t) \text{ in proposed translation}, count(t) \text{ in reference translation})}{count(t) \text{ in proposed translation}}$$

4.7 Experiments

Our experiments compare the translation quality of our method to the phrase-based model.

We test the model on the German-to-English Europarl data set (Koehn, 2005). When decoding, we use reordered German sentences (Collins et al., 2005) as our input. The reordering was utilizing the source-language (German) syntactic structure, while our method is incorporating the target-language (English) dependency structure. We use a set of 500 sentences for parameter tuning (see below). The standard test set consists of 2000 sentences. We use the first half as the development set and the second half as the test set.

In our experiments, we use the output from the Moses decoder with beam size 1000 to seed our local search. We compare our model which include the dependency language model and the usual phrase-based model. We use MERT (Och, 2003) to obtain the weights for each component of the phrase-based model. Then, the simplex method Nelder and Mead (1965) is used to tune the three parameters of language model (μ_l), dependency language (μ_d) and the word bonus (μ_b).

Each time we score a complete sentence, we parse the sentence using the Yara Parser (Rasooli and Tetreault, 2015), an implementation of the arc-eager dependency model (Nivre, 2004) with averaged perceptron training (Collins, 2002). The parser was trained on the Wall Street Journal

	Development		Test	
	phrase-based	+deplm	phrase-based	+deplm
BLEU	**26.29**	26.20	26.36	**26.53**
HWCM_{BLEU}	21.51	**22.27**	22.31	**23.52**

Table 4.2: Development and test set results show the effectiveness of adding the dependency language model in terms of the syntactic evaluation metric HWCM_{BLEU}. Here phrase-based is the original phrase-based model, and +deplm is our method that includes the dependency language model. Both methods are optimized using local search algorithm seeded with the same beam search results.

Local step	%
Swap	38
Move	4
Split	42
Merge	16

Table 4.3: The percentage of how often each type of local steps is taken to reach a higher scoring state. The number is based on the test data consisting of 1000 sentences.

data set. We set beam size to one for speed reason. The same parser is employed to build the dependency language model using the Europarl training set.

Table 4.2 shows the results on two metrics: BLEU and the syntactic evaluation metric HWCM_{BLEU} described in Section 4.6. The baseline is the phrase-based model phrase-based. Our method that includes the dependency language model into the scoring function is denoted by +deplm. Both methods use outputs from Moses as the initial state. The original phrase-based model is also optimized by the local search. Having the dependency language model, the HWCM_{BLEU} is increased by 0.7 on the development set and by 1.2 on the test set.

On the test set, we see improvement of scores on 93% of sentences after applying local search. It is not surprising since the initial state uses translation output from Moses which does not take the dependency language model score into account. The average number of local steps taken is 2.5 steps. Table 4.3 shows how often each type of local steps is taken. The Swap and Split are applied most often.

4.8 Conclusion

This chapter focuses on improving the translation quality. We include a dependency language model into the phrase-based translation model and use local search to optimize the model. The experimental results show that the translation quality is improved in terms of a syntactic evaluation metric.

In addition to consider the one-best tree when computing the dependency language model score, we can also consider multiple trees at the same time. One idea is to obtain the k-best parse trees and sum over the k trees, which can be viewed as marginalizing the probability. This strategy might be able to reduce the noise given by the parser.

Moreover, our method could be extended to include other information, since local search allows a more flexible design of the scoring function. In addition to syntactic information, we could also consider semantic information of the proposed translation.

Local search can be explored in more depth. One obvious strategy is to include randomization in the local search algorithm, such as random restart, which has shown improvement in various other applications. Alternative variants of local search are worth exploring as well. One example is stimulated annealing, which utilize randomness to avoid being stuck at local optima. We can also consider maintaining more than one state at a time.

Other improvement to the local search include different strategies to enlarge the search space of each local step. We can replace the exhaustive search in our algorithm by other optimization algorithms. For example, Lagrangian relaxation might be used to do efficient search over the search space of a neighborhood.

Chapter 5

Bidirectional Word Alignment

[This chapter is adapted from joint work with Alexander Rush, John DeNero, and Michael Collins entitled "A Constrained Viterbi Relaxation for Bidirectional Word Alignment" (Chang et al., 2014).]

In this chapter, we turn from phrase-based translation to bidirectional word alignment. The bidirectional word alignment formulation we work with is NP-hard. Our focus is on exact decoding algorithm based on Lagrangian relaxation. To achieve higher convergence rate, we use a tightening technique that incrementally re-introduce the constraints. To compensate for the cost of adding constraints, we use a optimality-preserving pruning to reduce the size of the search space.

5.1 Introduction

Word alignment is a critical first step for building statistical machine translation systems. In order to ensure accurate word alignments, most systems employ a post-hoc symmetrization step to combine directional word aligners, such as IBM Model 4 (Brown et al., 1993) or hidden Markov model (HMM) based aligners (Vogel et al., 1996). Several authors have proposed bidirectional models that incorporate this step directly, but decoding under many bidirectional models is NP-Hard and finding exact solutions has proven difficult.

In this chapter, we describe a novel Lagrangian-relaxation based decoder for the bidirectional model proposed by DeNero and Macherey (2011), with the goal of improving search accuracy.

In that work, the authors implement a dual decomposition-based decoder for the problem, but are only able to find exact solutions for around 6% of instances.

Our decoder uses a simple variant of the Viterbi algorithm for solving a relaxed version of this model. The algorithm makes it easy to re-introduce constraints for difficult instances, at the cost of increasing run-time complexity. To offset this cost, we employ optimality-preserving coarse-to-fine pruning to reduce the search space. The pruning method utilizes lower bounds on the cost of valid bidirectional alignments, which we obtain from a fast, greedy decoder.

The method has the following properties:

- It is based on a novel relaxation for the model of DeNero and Macherey (2011), solvable with a variant of the Viterbi algorithm.
- To find optimal solutions, it employs an efficient strategy that alternates between adding constraints and applying pruning.
- Empirically, it is able to find exact solutions on 86% of sentence pairs and is significantly faster than general-purpose solvers.

We begin in Section 5.2 by formally describing the directional word alignment problem. Section 5.3 describes a preliminary bidirectional model using full agreement constraints and a Lagrangian relaxation-based solver. Section 5.4 modifies this model to include adjacency constraints. Section 5.5 describes an extension to the relaxed algorithm to explicitly enforce constraints, and Section 5.6 gives a pruning method for improving the efficiency of the algorithm.

Experiments compare the search error and accuracy of the new bidirectional algorithm to several directional combiners and other bidirectional algorithms. Results show that the new relaxation is much more effective at finding exact solutions and is able to produce comparable alignment accuracy.

Notation We use lower- and upper-case letters for scalars and vectors, and script-case for sets e.g. $\mathcal{X}$. For vectors, such as $v \in \{0,1\}^{[(\mathcal{I}\times\mathcal{J})\cup\mathcal{J}]}$, where $\mathcal{I}$ and $\mathcal{J}$ are finite sets, we use the notation

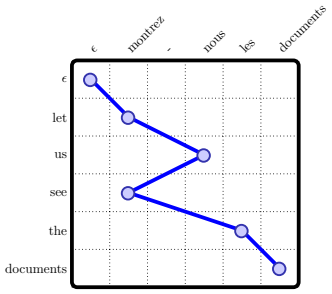


Figure 5-1: An example $\mathbf{e}\rightarrow\mathbf{f}$ directional alignment for the sentences `let us see the documents` and `montrez - nous les documents`, with $I = 5$ and $J = 5$. The indices $i \in [I]_0$ are rows, and the indices $j \in [J]_0$ are columns. The HMM alignment shown has transitions $x(0, 1, 1) = x(1, 2, 3) = x(3, 3, 1) = x(1, 4, 4) = x(4, 5, 5) = 1$.

$v(i, j)$ and $v(j)$ to represent elements of the vector. Define $d = \delta(i)$ to be the indicator vector with $d(i) = 1$ and $d(i') = 0$ for all $i' \neq i$. Finally define the notation $[J]$ to refer to $\{1 \dots J\}$ and $[J]_0$ to refer to $\{0 \dots J\}$.

5.2 Background

The focus of this work is on the word alignment decoding problem. Given a sentence $\mathbf{e}$ of length $|\mathbf{e}| = I$ and a sentence $\mathbf{f}$ of length $|\mathbf{f}| = J$, our goal is to find the best bidirectional alignment between the two sentences under a given objective function. Before turning to the model of interest, we first introduce directional word alignment.

5.2.1 Word Alignment

In the $\mathbf{e}\rightarrow\mathbf{f}$ word alignment problem, each word in $\mathbf{e}$ is aligned to a word in $\mathbf{f}$ or to the null word ϵ. This alignment is a mapping from each index $i \in [I]$ to an index $j \in [J]_0$ (where $j = 0$ represents alignment to ϵ). We refer to a single word alignment as a *link*.

A first-order HMM alignment model (Vogel et al., 1996) is an HMM of length $I + 1$ where the hidden state at position $i \in [I]_0$ is the aligned index $j \in [J]_0$, and the transition score takes into account the previously aligned index $j' \in [J]_0$.[1] Formally, define the set of possible HMM

[1] Our definition differs slightly from other HMM-based aligners in that it does not track the last ϵ alignment.

alignments as $\mathcal{X} \subset \{0,1\}^{[([I]_0 \times [J]_0) \cup ([I] \times [J]_0 \times [J]_0)]}$ with

$$\mathcal{X} = \begin{cases} x : x(0,0) = 1, \\ x(i,j) = \sum_{j'=0}^{J} x(j',i,j) & \forall i \in [I], j \in [J]_0, \\ x(i,j) = \sum_{j'=0}^{J} x(j,i+1,j') & \forall i \in [I-1]_0, j \in [J]_0 \end{cases}$$

where $x(i,j) = 1$ indicates that there is a link between index i and index j, and $x(j',i,j) = 1$ indicates that index $i-1$ aligns to index j' and index i aligns to j. Figure 5-1 shows an example member of $\mathcal{X}$.

The constraints of $\mathcal{X}$ enforce *backward* and *forward* consistency respectively. If $x(i,j) = 1$, backward consistency enforces that there is a transition from $(i-1,j')$ to (i,j) for some $j' \in [J]_0$, whereas forward consistency enforces a transition from (i,j) to $(i+1,j')$ for some $j' \in [J]_0$. Informally the constraints "chain" together the links.

The HMM objective function $f : \mathcal{X} \to \mathbb{R}$ can be written as a linear function of x

$$f(x;\theta) = \sum_{i=1}^{I} \sum_{j=0}^{J} \sum_{j'=0}^{J} \theta(j',i,j) x(j',i,j)$$

where the vector $\theta \in \mathbb{R}^{[I] \times [J]_0 \times [J]_0}$ includes the transition and alignment scores. For a generative model of alignment, we might define $\theta(j',i,j) = \log(p(\mathbf{e}_i|\mathbf{f}_j)p(j|j'))$. For a discriminative model of alignment, we might define $\theta(j',i,j) = w \cdot \phi(i,j',j,\mathbf{f},\mathbf{e})$ for a feature function ϕ and weights w (Moore, 2005; Lacoste-Julien et al., 2006).

Now reverse the direction of the model and consider the $\mathbf{f}\to\mathbf{e}$ alignment problem. An $\mathbf{f}\to\mathbf{e}$ alignment is a binary vector $y \in \mathcal{Y}$ where for each $j \in [J]$, $y(i,j) = 1$ for exactly one $i \in [I]_0$. Define the set of HMM alignments $\mathcal{Y} \subset \{0,1\}^{[([I]_0 \times [J]_0) \cup ([I]_0 \times [I]_0 \times [J])]}$ as

$$\mathcal{Y} = \left\{ \begin{array}{ll} y : y(0,0) = 1, & \\ y(i,j) = \sum_{i'=0}^{I} y(i',i,j) & \forall i \in [I]_0, j \in [J], \\ y(i,j) = \sum_{i'=0}^{I} y(i,i',j+1) & \forall i \in [I]_0, j \in [J-1]_0 \end{array} \right\}$$

Similarly define the objective function

$$g(y;\omega) = \sum_{j=1}^{J} \sum_{i=0}^{I} \sum_{i'=0}^{I} \omega(i',i,j) y(i',i,j)$$

with vector $\omega \in \mathbb{R}^{[I]_0 \times [I]_0 \times [J]}$.

Note that for both of these models we can solve the optimization problem exactly using the standard Viterbi algorithm for HMM decoding. The first can be solved in $O(IJ^2)$ time and the second in $O(I^2J)$ time.

5.3 Bidirectional Alignment

The directional bias of the $\mathbf{e}\rightarrow\mathbf{f}$ and $\mathbf{f}\rightarrow\mathbf{e}$ alignment models may cause them to produce differing alignments. To obtain the best single alignment, it is common practice to use a post-hoc algorithm to merge these directional alignments (Och et al., 1999a). First, a directional alignment is found from each word in $\mathbf{e}$ to a word $\mathbf{f}$. Next an alignment is produced in the reverse direction from $\mathbf{f}$ to $\mathbf{e}$. Finally, these alignments are merged, either through intersection, union, or with an interpolation algorithm such as grow-diag-final (Koehn et al., 2003).

In this work, we instead consider a bidirectional alignment model that jointly considers both directional models. We begin in this section by introducing a simple bidirectional model that enforces *full* agreement between directional models and giving a relaxation for decoding. Section 5.4 loosens this model to *adjacent* agreement.

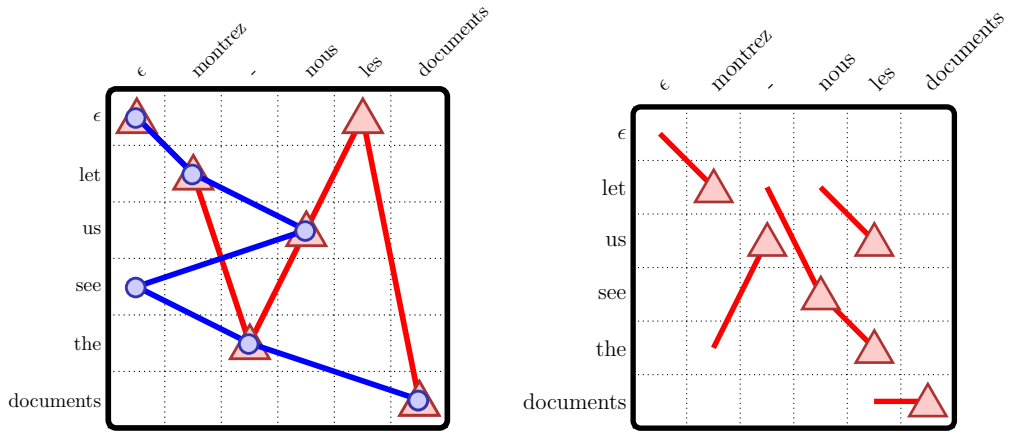


Figure 5-2: (a) An example alignment pair (x, y) satisfying the full agreement conditions. The x alignment is represented with circles and the y alignment with triangles. (b) An example $\mathbf{f}{\rightarrow}\mathbf{e}$ alignment $y \in \mathcal{Y}'$ with relaxed forward constraints. Note that unlike an alignment from $\mathcal{Y}$ multiple words may be aligned in a column and words may transition from non-aligned positions.

5.3.1 Enforcing Full Agreement

Perhaps the simplest post-hoc merging strategy is to retain the intersection of the two directional models. The analogous bidirectional model enforces *full* agreement to ensure the two alignments select the same non-null links i.e.

$$x^*, y^* = \underset{x \in \mathcal{X}, y \in \mathcal{Y}}{\arg\max}\, f(x) + g(y) \text{ s.t.}$$
$$x(i,j) = y(i,j) \quad \forall i \in [I], j \in [J]$$

We refer to the optimal alignments for this problem as x^* and y^*.

Unfortunately this bidirectional decoding model is NP-Hard (a proof is given in Appendix B.1). As it is common for alignment pairs to have $|\mathbf{f}|$ or $|\mathbf{e}|$ over 40, exact decoding algorithms are intractable in the worst-case.

Instead we will use Lagrangian relaxation for this model. At a high level, we will remove a subset of the constraints from the original problem and replace them with Lagrange multipliers. If we can solve this new problem efficiently, we may be able to get optimal solutions to the original

problem. (See the tutorial by Rush and Collins (2012) describing the method.)

There are many possible subsets of constraints to consider relaxing. The relaxation we use preserves the agreement constraints while relaxing the Markov structure of the $\mathbf{f}\rightarrow\mathbf{e}$ alignment. This relaxation will make it simple to later re-introduce constraints in Section 5.5.

We relax the forward constraints of set $\mathcal{Y}$. Without these constraints the y links are no longer chained together. This relaxation has two consequences: (1) for index j there may be any number of indices i, such that $y(i,j)=1$, (2) if $y(i',i,j)=1$ it is no longer required that $y(i',j-1)=1$. The relaxation gives a set $\mathcal{Y}'$ which is a superset of $\mathcal{Y}$

$$\mathcal{Y}' = \left\{ \begin{array}{l} y : y(0,0)=1, \\ y(i,j)=\sum_{i'=0}^{I} y(i',i,j) \quad \forall i \in [I]_0, j \in [J] \end{array} \right\}$$

Figure 5-2b shows a possible $y \in \mathcal{Y}'$ and a valid unchained structure.

To form the Lagrangian dual with relaxed forward constraints, we introduce a vector of Lagrange multipliers, $\lambda \in \mathbb{R}^{[I-1]_0 \times [J]_0}$, with one multiplier for each original constraint. The Lagrangian dual $L(\lambda)$ is defined as

$$\begin{aligned}
& \max_{\substack{x\in\mathcal{X}, y\in\mathcal{Y}', \\ x(i,j)=y(i,j)}} f(x) + \sum_{i=1}^{I}\sum_{j=0}^{J}\sum_{i'=0}^{I} y(i',i,j)\omega(i',i,j) && (5.1) \\
& \quad -\sum_{i=0}^{I}\sum_{j=0}^{J-1} \lambda(i,j)\left(y(i,j) - \sum_{i'=0}^{I} y(i,i',j+1)\right) \\
= & \max_{\substack{x\in\mathcal{X}, y\in\mathcal{Y}', \\ x(i,j)=y(i,j)}} f(x) + \sum_{i=1}^{I}\sum_{j=0}^{J}\sum_{i'=0}^{I} y(i',i,j)\omega'(i',i,j) && (5.2) \\
= & \max_{\substack{x\in\mathcal{X}, y\in\mathcal{Y}', \\ x(i,j)=y(i,j)}} f(x) + \sum_{i=1}^{I}\sum_{j=0}^{J} y(i,j) \max_{i'\in[I]_0} \omega'(i',i,j) && (5.3) \\
= & \max_{\substack{x\in\mathcal{X}, y\in\mathcal{Y}', \\ x(i,j)=y(i,j)}} f(x) + g'(y;\omega,\lambda) && (5.4)
\end{aligned}$$

Line 5.2 distributes the λ's and introduces a modified potential vector ω' defined as

$$\omega'(i', i, j) = \omega(i', i, j) - \lambda(i, j) + \lambda(i', j-1)$$

for all $i' \in [I]_0, i \in [I]_0, j \in [J]$. Line 5.3 utilizes the relaxed set $\mathcal{Y}'$ which allows each $y(i, j)$ to select the best possible previous link $(i', j-1)$. Line 5.4 introduces the modified directional objective

$$g'(y; \omega, \lambda) = \sum_{i=1}^{I} \sum_{j=0}^{J} y(i, j) \max_{i' \in [I]_0} \omega'(i', i, j)$$

The Lagrangian dual is guaranteed to be an upper bound on the optimal solution, i.e. for all λ, $L(\lambda) \geq f(x^*) + g(y^*)$. Lagrangian relaxation attempts to find the tighest possible upper bound by minimizing the Lagrangian dual, $\min_\lambda L(\lambda)$, using subgradient descent. Briefly, subgradient descent is an iterative algorithm, with two steps. Starting with $\lambda = 0$, we iteratively

1. Set (x, y) to the $\arg\max$ of $L(\lambda)$.

2. Update $\lambda(i, j)$ for all $i \in [I-1]_0, j \in [J]_0$,

$$\lambda(i, j) \leftarrow \lambda(i, j) - \eta_t \big(y(i, j) - \sum_{i'=0}^{I} y(i, i', j+1)\big)$$

.

where $\eta_t > 0$ is a step size for the t'th update. If at any iteration of the algorithm the forward constraints are satisfied for (x, y), then $f(x) + g(y) = f(x^*) + g(x^*)$ and we say this gives a *certificate of optimality* for the underlying problem.

To run this algorithm, we need to be able to efficiently compute the (x, y) pair that is the $\arg\max$ of $L(\lambda)$ for any value of λ. Fortunately, since the y alignments are no longer constrained to valid transitions, we can compute these alignments by first picking the best $\mathbf{f} \rightarrow \mathbf{e}$ transitions for each possible link, and then running an $\mathbf{e} \rightarrow \mathbf{f}$ Viterbi-style algorithm to find the bidirectional alignment.

procedure VITERBIFULL(θ, ω')
Let π, ρ be dynamic programming charts.
$\rho[i, j] \leftarrow \max_{i' \in [I]_0} \omega'(i', i, j) \; \forall \, i \in [I], j \in [J]_0$
$\pi[0, 0] \leftarrow \sum_{j=1}^{J} \max\{0, \rho[0, j]\}$
for $i \in [I], j \in [J]_0$ in order **do**
$\pi[i, j] \leftarrow \max_{j' \in [J]_0} \theta(j', i, j) + \pi[i - 1, j']$
if j $\neq 0$ **then** $\pi[i, j] \leftarrow \pi[i, j] + \rho[i, j]$
return $\max_{j \in [J]_0} \pi[I, j]$

Figure 5-3: Viterbi-style algorithm for computing $L(\lambda)$. For simplicity the algorithm shows the $\max$ version of the algorithm, $\arg\max$ can be computed with back-pointers.

The $\max$ version of this algorithm is shown in Figure 5-3. It consists of two steps. We first compute the score for each $y(i, j)$ variable. We then use the standard Viterbi update for computing the x variables, adding in the score of the $y(i, j)$ necessary to satisfy the constraints.

5.4 Adjacent Agreement

Enforcing full agreement can be too strict an alignment criteria. DeNero and Macherey (2011) instead propose a model that allows near matches, which we call *adjacent* agreement. Adjacent agreement allows links from one direction to agree with adjacent links from the reverse alignment for a small penalty. Figure 5-4a shows an example of a valid bidirectional alignment under adjacent agreement.

In this section we formally introduce adjacent agreement, and propose a relaxation algorithm for this model. The key algorithmic idea is to extend the Viterbi algorithm in order to consider possible adjacent links in the reverse direction.

5.4.1 Enforcing Adjacency

Define the adjacency set $\mathcal{K} = \{-1, 0, 1\}$. A bidirectional alignment satisfies adjacency if for all $i \in [I], j \in [J]$,

- If $x(i, j) = 1$, it is required that $y(i + k, j) = 1$ for exactly one $k \in \mathcal{K}$ (i.e. either above,

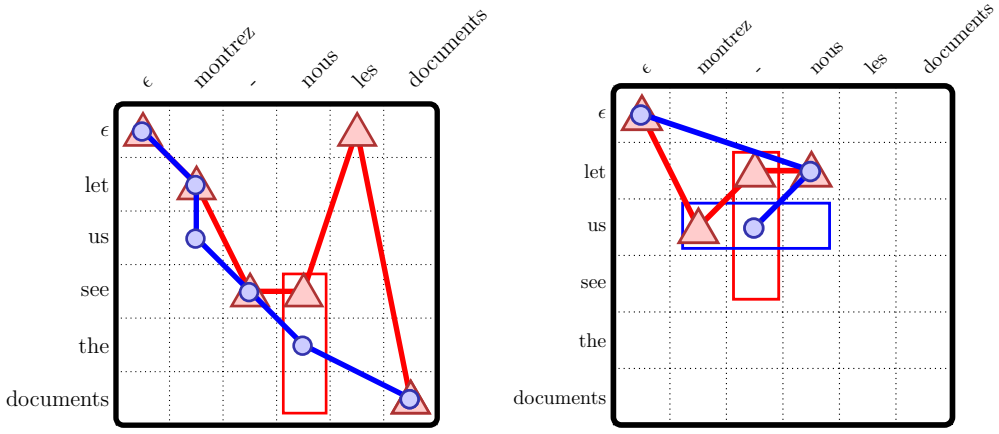


Figure 5-4: (a) An alignment satisfying the adjacency constraints. Note that $x(2,1) = 1$ is allowed because of $y(1,1) = 1$, $x(4,3) = 1$ because of $y(3,3)$, and $y(3,1)$ because of $x(3,2)$. (b) An adjacent bidirectional alignment in progress. Currently $x(2,2) = 1$ with $z^{\updownarrow}(-1) = 1$ and $z^{\leftrightarrow}(-1) = 1$. The last transition was from $x(1,3)$ with $z^{\leftrightarrow\prime}(-1) = 1, z^{\leftrightarrow\prime}(0) = 1, z^{\updownarrow\prime}(0) = 1$.

center, or below). We indicate which position with variables $z^{\updownarrow}{}_{i,j} \in \{0,1\}^{[}\mathcal{K}$

- If $x(i,j) = 1$, it is allowed that $y(i, j+k) = 1$ for any $k \in \mathcal{K}$ (i.e. either left, center, or right) and all other $y(i,j') = 0$. We indicate which positions with variables $z^{\leftrightarrow}{}_{i,j} \in \{0,1\}^{[}\mathcal{K}$

Formally for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, the pair (x,y) is feasible if there exists a z from the set $\mathcal{Z}(x,y) \subset \{0,1\}^{[\mathcal{K}^2 \times [I] \times [J]]}$ defined as

$$\mathcal{Z}(x,y) = \left\{ \begin{array}{ll} z : \forall i \in [I], j \in [J] & \\ z^{\updownarrow}{}_{i,j} \in \{0,1\}^{[\mathcal{K}]}, & z^{\leftrightarrow}{}_{i,j} \in \{0,1\}^{[\mathcal{K}]} \\ x(i,j) = \sum_{k \in \mathcal{K}} z^{\updownarrow}{}_{i,j}(k), & \sum_{k \in \mathcal{K}} z^{\leftrightarrow}{}_{i,j}(k) = y(i,j), \\ z^{\updownarrow}{}_{i,j}(k) \leq y(i+k,j) & \forall k \in \mathcal{K} : i+k > 0, \\ x(i,j) \geq z^{\leftrightarrow}{}_{i,j-k}(k) & \forall k \in \mathcal{K} : j+k > 0 \end{array} \right\}$$

Additionally adjacent, non-overlapping matches are assessed a penalty α calculated as

$$h(z) = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k \in \mathcal{K}} \alpha |k| (z^{\updownarrow}{}_{i,j}(k) + z^{\leftrightarrow}{}_{i,j}(k))$$

where $\alpha \leq 0$ is a parameter of the model. The example in Figure 5-4a includes a 3α penalty.

Adding these penalties gives the complete adjacent agreement problem

$$\arg\max_{\substack{z \in \mathcal{Z}(x,y) \\ x \in \mathcal{X}, y \in \mathcal{Y}}} f(x) + g(y) + h(z)$$

Next, apply the same relaxation from Section 5.3.1, i.e. we relax the forward constraints of the $\mathbf{f} \rightarrow \mathbf{e}$ set. This yields the following Lagrangian dual

$$L(\lambda) = \max_{\substack{z \in \mathcal{Z}(x,y) \\ x \in \mathcal{X}, y \in \mathcal{Y}'}} f(x) + g'(y; \omega, \lambda) + h(z)$$

Despite the new constraints, we can still compute $L(\lambda)$ in $O(IJ(I+J))$ time using a variant of the Viterbi algorithm. The main idea will be to consider possible adjacent settings for each link. Since each $z^{\updownarrow}{}_{i,j}$ and $z^{\leftrightarrow}{}_{i,j}$ only have a constant number of settings, this modification does not increase the asymptotic complexity of the algorithm.

Figure 5-5 shows the algorithm for computing $L(\lambda)$. The main loop of the algorithm is similar to Figure 5-3. It proceeds row-by-row, picking the best alignment $x(i,j) = 1$. The major change is that the chart π also stores a value $z \in \{0,1\}^{[\mathcal{K} \times \mathcal{K}]}$ representing a possible $z^{\updownarrow}{}_{i,j}, z^{\leftrightarrow}{}_{i,j}$ pair. Since we have the proposed $z_{i,j}$ in the inner loop, we can include the scores of the adjacent y alignments that are in neighboring columns, as well as the possible penalty for matching $x(i,j)$ to a $y(i+k,j)$ in a different row. Figure 5-4b gives an example setting of z.

In the dynamic program, we need to ensure that the transitions between the z's are consistent. The vector z' indicates the y links adjacent to $x(i-1,j')$. If j' is near to j, z' may overlap with z and vice-versa. The transition set $\mathcal{N}$ ensures these indicators match up

procedure VITERBIADJ(θ, ω')
$\rho[i,j] \leftarrow \max_{i' \in [I]_0} \omega'(i',i,j) \; \forall \, i \in [I], j \in [J]_0$
$\pi[0,0] \leftarrow \sum_{j=1}^{J} \max\{0, \rho[0,j]\}$
for $i \in [I], j \in [J]_0, z^{\updownarrow}, z^{\leftrightarrow} \in \{0,1\}^{[|\mathcal{K}|]}$ **do**
$\pi[i,j,z] \leftarrow$
$$\max_{\substack{j' \in [J]_0, \\ z' \in \mathcal{N}(z, j-j')}} \theta(j',i,j) + \pi[i-1, j', z'] + \sum_{k \in \mathcal{K}} z^{\leftrightarrow}(k)(\rho[i,j+k] + \alpha|k|) + z^{\updownarrow}(k)\alpha|k|$$
return $\max_{j \in [J]_0, z \in \{0,1\}^{[|\mathcal{K}| \times \mathcal{K}|]}} \pi[I,j,z]$

Figure 5-5: Modified Viterbi algorithm for computing the adjacent agreement $L(\lambda)$.

$$\mathcal{N}(z,k') = \left\{ \begin{array}{ll} z': & (z^{\updownarrow}(-1) \wedge k' \in \mathcal{K}) \Rightarrow z^{\leftrightarrow\prime}(k'), \\ & (z^{\updownarrow\prime}(1) \wedge k' \in \mathcal{K}) \Rightarrow z^{\leftrightarrow}(-k'), \\ & \sum_{k \in \mathcal{K}} z^{\updownarrow}(k) = 1 \end{array} \right\}$$

5.5 Adding Back Constraints

In general, it can be shown that Lagrangian relaxation is only guaranteed to solve a linear programming relaxation of the underlying combinatorial problem. For difficult instances, we will see that this relaxation often does not yield provably exact solutions. However, it is possible to "tighten" the relaxation by re-introducing constraints from the original problem.

In this section, we extend the algorithm to allow incrementally re-introducing constraints. In particular we track which constraints are most often violated in order to explicitly enforce them in the algorithm.

Define a binary vector $p \in \{0,1\}^{[[I-1]_0 \times [J]_0]}$ where $p(i,j) = 1$ indicates a previously relaxed constraint on link $y(i,j)$ that should be re-introduced into the problem. Let the new partially

constrained Lagrangian dual be defined as

$$L(\lambda; p) = \max_{\substack{z \in \mathcal{Z}(x,y) \\ x \in \mathcal{X}, y \in \mathcal{Y}'}} f(x) + g'(y; \omega, \lambda) + h(z)$$
$$y(i,j) = \sum_{i'} y(i, i', j+1) \quad \forall i, j : p(i,j) = 1$$

If $p = \vec{1}$, the problem includes all of the original constraints, whereas $p = \vec{0}$ gives our original Lagrangian dual. In between we have progressively more constrained variants.

In order to compute the $\arg\max$ of this optimization problem, we need to satisfy the constraints within the Viterbi algorithm. We augment the Viterbi chart with a count vector $d \in \mathcal{D}$ where $\mathcal{D} \subset Z^{||p||_1}$ and $d(i,j)$ is a count for the (i,j)'th constraint, i.e. $d(i,j) = y(i,j) - \sum_{i'} y(i', i, j)$. Only solutions with count 0 at the final position satisfy the active constraints. Additionally define a helper function $[\cdot]_{\mathcal{D}}$ as the projection from $Z^{[I-1]_0 \times [J]} \rightarrow \mathcal{D}$, which truncates dimensions without constraints.

Figure 5-6 shows this constrained Viterbi relaxation approach. It takes p as an argument and enforces the active constraints. For simplicity, we show the full agreement version, but the adjacent agreement version is similar. The main new addition is that the inner loop of the algorithm ensures that the count vector d is the sum of the counts of its children d' and $d - d'$.

Since each additional constraint adds a dimension to d, adding constraints has a multiplicative impact on running time. Asymptotically the new algorithm requires $O(2^{||p||_1} IJ(I+J))$ time. This is a problem in practice as even adding a few constraints can make the problem intractable. We address this issue in the next section.

5.6 Pruning

Re-introducing constraints can lead to an exponential blow-up in the search space of the Viterbi algorithm. In practice though, many alignments in this space are far from optimal, e.g. aligning a common word like `the` to `nous` instead of `les`. Since Lagrangian relaxation re-computes the

procedure CONSVITERBIFULL(θ, ω', p)
 for $i \in [I], j \in [J]_0, i' \in [I]$ **do**
  $d \leftarrow |\delta(i,j) - \delta(i', j-1)|_{\mathcal{D}}$
  $\rho[i,j,d] \leftarrow \omega'(i', i, j)$
 for $j \in [J], d \in \mathcal{D}$ **do**
  $\pi[0,0,d] \leftarrow \max_{d' \in \mathcal{D}} \pi[0,0,d'] + \rho[0,j,d-d']$
 for $i \in [I], j \in [J]_0, d \in \mathcal{D}$ **do**
  if j = 0 **then**
   $\pi[i,j,d] \leftarrow \max_{j' \in [J]_0} \theta(j', i, j) + \pi[i-1, j', d]$
  else
   $\pi[i,j,d] \leftarrow$
   $\max_{j' \in [J]_0, d' \in \mathcal{D}} \theta(j', i, j) + \pi[i-1, j', d'] + \rho[i,j,d-d']$
 return $\max_{j \in [J]_0} \pi[I, j, \mathbf{0}]$

Figure 5-6: Constrained Viterbi algorithm for finding partially-constrained, full-agreement alignments. The argument p indicates which constraints to enforce.

alignment many times, it would be preferable to skip these links in later rounds, particularly after re-introducing constraints.

In this section we describe an optimality preserving coarse-to-fine algorithm for pruning. Approximate coarse-to-fine pruning algorithms are widely used within NLP, but exact pruning is less common. Our method differs in that it only eliminates non-optimal transitions based on a lower-bound score. After introducing the pruning method, we present an algorithm to make this method effective in practice by producing high-scoring lower bounds for adjacent agreement.

5.6.1 Thresholding Max-Marginals

Our pruning method is based on removing transitions with low max-marginal values. Define the max-marginal value of an $\mathbf{e} \rightarrow \mathbf{f}$ transition in our Lagrangian dual as

$$M(j', i, j; \lambda) = \max_{\substack{z \in \mathcal{Z}(x,y), \\ x \in \mathcal{X}, y \in \mathcal{Y}'}} f(x) + g'(y; \lambda) + h(z)$$
$$\text{s.t. } x(j', i, j) = 1$$

where M gives the value of the best dual alignment that transitions from $(i-1, j')$ to (i, j). These max-marginals can be computed by running a forward-backward variant of any of the algorithms described thus far.

We make the following claim about max-marginal values and any lower-bound score

Lemma 1 (Safe Pruning)**.** *For any valid constrained alignment $x \in \mathcal{X}, y \in \mathcal{Y}, z \in \mathcal{Z}(x, y)$ and for any dual vector $\lambda \in \mathbb{R}^{[I-1]_0 \times [J]_0}$, if there exists a transition j', i, j with max-marginal value $M(j', i, j; \lambda) < f(x) + g(y) + h(z)$ then the transition will not be in the optimal alignment, i.e. $x^*(j', i, j) = 0$.*

This lemma tells us that we can prune transitions whose dual max-marginal value falls below a threshold without pruning possibly optimal transitions. Pruning these transitions can speed up Lagrangian relaxation without altering its properties.

Furthermore, the threshold is determined by any feasible lower bound on the optimal score, which means that better bounds can lead to more pruning.

5.6.2 Finding Lower Bounds

Since the effectiveness of pruning is dependent on the lower bound, it is crucial to be able to produce high-scoring alignments that satisfy the agreement constraints. Unfortunately, this problem is non-trivial. For instance, taking the union of directional alignments does not guarantee a feasible solution; whereas taking the intersection is trivially feasible but often not high-scoring.

To produce higher-scoring feasible bidirectional alignments we introduce a heuristic algorithm. The algorithm starts with any feasible alignment (x, y, z). It runs the following greedy loop:

1. Repeat until there exists no $x(i, 0) = 1$ or $y(0, j) = 1$, or there is no score increase.

   (a) For each $i \in [I], j \in [J]_0, k \in \mathcal{K} : x(i, 0) = 1$, check if $x(i, j) \leftarrow 1$ and $y(i, j+k) \leftarrow 1$ is feasible, remember score.

   (b) For each $i \in [I]_0, j \in [J], k \in \mathcal{K} : y(0, j) = 1$, check if $y(i, j) \leftarrow 1$ and $x(i+k, j) \leftarrow 1$ is feasible, remember score.

	1-20 (28%)			21-40 (45%)			41-60 (27%)			all		
	time	cert	exact	time	cert	exact	time	cert	exact	time	cert	exact
ILP	15.12	100.0	100.0	364.94	100.0	100.0	2,829.64	100.0	100.0	924.24	100.0	100.0
LR	0.55	97.6	97.6	4.76	55.9	55.9	15.06	7.5	7.5	6.33	54.7	54.7
CONS	0.43	100.0	100.0	9.86	95.6	95.6	61.86	55.0	62.5	21.08	86.0	88.0
D&M	-	6.2	-	-	0.0	-	-	0.0	-	-	6.2	-

Table 5.1: Experimental results for model accuracy of bilingual alignment. Column *time* is the mean time per sentence pair in seconds; *cert* is the percentage of sentence pairs solved with a certificate of optimality; *exact* is the percentage of sentence pairs solved exactly. Results are grouped by sentence length. The percentage of sentence pairs in each group is shown in parentheses.

(c) Let (x, y, z) be the highest-scoring feasible solution produced.

This algorithm produces feasible alignments with monotonically increasing score, starting from the intersection of the alignments. It has run-time of $O(IJ(I + J))$ since each inner loop enumerates IJ possible updates and assigns at least one index a non-zero value, limiting the outer loop to $I + J$ iterations.

In practice we initialize the heuristic based on the intersection of x and y at the current round of Lagrangian relaxation. Experiments show that running this algorithm significantly improves the lower bound compared to just taking the intersection, and consequently helps pruning significantly.

5.7 Related Work

The most common techniques for bidirectional alignment are post-hoc combinations, such as union or intersection, of directional models, (Och et al., 1999a), or more complex heuristic combiners such as grow-diag-final (Koehn et al., 2003).

Several authors have explored explicit bidirectional models in the literature. Cromieres and Kurohashi (2009) use belief propagation on a factor graph to train and decode a one-to-one word alignment problem. Qualitatively this method is similar to ours, although the model and decoding algorithm are different, and their method is not able to provide certificates of optimality.

A series of papers by Ganchev et al. (2010), Graca et al. (2008), and Ganchev et al. (2008) use posterior regularization to constrain the posterior probability of the word alignment problem to be

symmetric and bijective. This work acheives state-of-the-art performance for alignment. Instead of utilizing posteriors our model tries to decode a single best one-to-one word alignment.

A different approach is to use constraints at training time to obtain models that favor bidirectional properties. Liang et al. (2006) propose agreement-based learning, which jointly learns probabilities by maximizing a combination of likelihood and agreement between two directional models.

General linear programming approaches have also been applied to word alignment problems. Lacoste-Julien et al. (2006) formulate the word alignment problem as quadratic assignment problem and solve it using an integer linear programming solver.

Our work is most similar to DeNero and Macherey (2011), which uses dual decomposition to encourage agreement between two directional HMM aligners during decoding time.

5.8 Experiments

Our experimental results compare the accuracy and optimality of our decoding algorithm to directional alignment models and previous work on this bidirectional model.

Data and Setup The experimental setup is identical to DeNero and Macherey (2011). Evaluation is performed on a hand-aligned subset of the NIST 2002 Chinese-English dataset (Ayan and Dorr, 2006). Following past work, the first 150 sentence pairs of the training section are used for evaluation. The potential parameters θ and ω are set based on unsupervised HMM models trained on the LDC FBIS corpus (6.2 million words). Training is performed using the agreement-based learning method which encourages the directional models to overlap (Liang et al., 2006). This directional model has been shown produce state-of-the-art results with this setup (Haghighi et al., 2009).

Baselines We compare our algorithm with several baseline methods. DIR includes post-hoc combinations of the $\mathbf{e}\rightarrow\mathbf{f}$ and $\mathbf{f}\rightarrow\mathbf{e}$ HMM-based aligners. Variants include union, intersection,

and grow-diag-final. D&M is the dual decomposition algorithm for bidirectional alignment as presented by DeNero and Macherey (2011) with different final combinations. LR is the Lagrangian relaxation algorithm applied to the adjacent agreement problem without the additional constraints described in Section 5.5. CONS is our full Lagrangian relaxation algorithm including incremental constraint addition. ILP uses a highly-optimized general-purpose integer linear programming solver (Gurobi Optimization, 2013) to solve the lattice with the constraints described.

Implementation The main task of the decoder is to repeatedly compute the $\arg\max$ of $L(\lambda)$. To speed up decoding, our implementation fully instantiates the Viterbi lattice for a problem instance. This approach has several benefits: each iteration can reuse the same lattice structure; max-marginals can be easily computed with a general forward-backward algorithm; pruning corresponds to removing lattice edges; and adding constraints can be done through lattice intersection. For consistency, we implement each baseline (except for D&M) through the same lattice.

Parameter Settings We run 400 iterations of the subgradient algorithm using the rate schedule $\eta_t = 0.95^{t'}$ where t' is the count of updates for which the dual value did not improve. Every 10 iterations we run the greedy decoder to compute a lower bound. If the gap between our current dual value $L(\lambda)$ and the lower bound improves significantly we run coarse-to-fine pruning as described in Section 5.6 with the best lower bound. For CONS, if the algorithm does not converge we run 400 more iterations and incrementally add the 5 most violated constraints every 25 iterations.

Results Our first set of experiments looks at the model accuracy and the decoding time of various methods that can produce optimal solutions. Results are shown in Table 5.1. D&M is only able to find the optimal solution with certificate on 6% of instances. The relaxation algorithm used in this work is able to increase that number to 54.7%. With incremental constraints and pruning, we are able to solve over 86% of sentence pairs including many longer and more difficult pairs. Additionally the method finds these solutions with only a small increase in running time over Lagrangian relaxation, and is significantly faster than using an ILP solver.

Model	Combiner	alignment			phrase pair		
		Prec	Rec	AER	Prec	Rec	F1
DIR	union	57.6	80.0	33.4	75.1	33.5	46.3
	intersection	86.2	62.9	27.0	64.3	43.5	51.9
	grow-diag	59.7	79.5	32.1	70.1	36.9	48.4
D&M	union	63.3	81.5	29.1	63.2	44.9	52.5
	intersection	77.5	75.1	23.6	57.1	53.6	55.3
	grow-diag	65.6	80.6	28.0	60.2	47.4	53.0
CONS		72.5	74.9	26.4	53.0	52.4	52.7

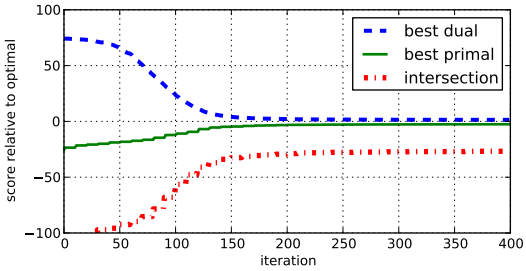
Table 5.2: Alignment accuracy and phrase pair extraction accuracy for directional and bidirectional models. `Prec` is the precision. `Rec` is the recall. `AER` is alignment error rate and `F1` is the phrase pair extraction F1 score.

	1-20	21-40	41-60	all
# cons.	20.0	32.1	39.5	35.9

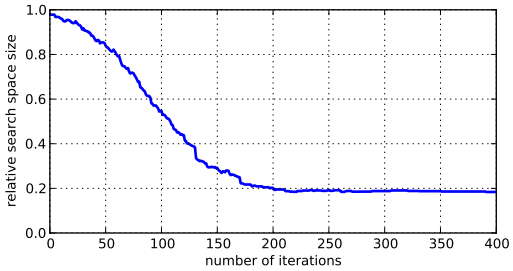
Table 5.3: The average number of constraints added for sentence pairs where Lagrangian relaxation is not able to find an exact solution.

Next we compare the models in terms of alignment accuracy. Table 5.2 shows the precision, recall and alignment error rate (AER) for word alignment. We consider union, intersection and grow-diag-final as combination procedures. The combination procedures are applied to D&M in the case when the algorithm does not converge. For CONS, we use the optimal solution for the 86% of instances that converge and the highest-scoring greedy solution for those that do not. The proposed method has an AER of 26.4, which outperforms each of the directional models. However, although CONS achieves a higher model score than D&M, it performs worse in accuracy. Table 5.2 also compares the models in terms of phrase-extraction accuracy (Ayan and Dorr, 2006). We use the phrase extraction algorithm described by DeNero and Klein (2010), accounting for possible links and ϵ alignments. CONS performs better than each of the directional models, but worse than the best D&M model.

Finally we consider the impact of constraint addition, pruning, and use of a lower bound. Table 5.3 gives the average number of constraints added for sentence pairs for which Lagrangian relaxation alone does not produce a certificate. Figure 5-7a shows the average over all sentence pairs of the best dual and best primal scores. The graph compares the use of the greedy algorithm from Section 5.6.2 with the simple intersection of x and y. The difference between these curves illustrates the benefit of the greedy algorithm. This is reflected in Figure 5-7b which shows the



(a) The best dual and the best primal score, relative to the optimal score, averaged over all sentence pairs. The `best primal` curve uses a feasible greedy algorithm, whereas the `intersection` curve is calculated by taking the intersection of x and y.



(b) A graph showing the effectiveness of coarse-to-fine pruning. Relative search space size is the size of the pruned lattice compared to the initial size. The plot shows an average over all sentence pairs.

Figure 5-7

effectiveness of coarse-to-fine pruning over time. On average, the pruning reduces the search space of each sentence pair to 20% of the initial search space after 200 iterations.

5.9 Conclusion

We have introduced a novel Lagrangian relaxation algorithm for a bidirectional alignment model that uses incremental constraint addition and coarse-to-fine pruning to find exact solutions. The algorithm increases the number of exact solution found on the model of DeNero and Macherey (2011) from 6% to 86%.

Unfortunately despite achieving higher model score, this approach does not produce more accurate alignments than the previous algorithm. This result suggests that the adjacent agreement model may still be too constrained for this underlying task. Implicitly, an approach with fewer exact solutions may allow for useful violations of these constraints. In future work, we hope to explore bidirectional models with soft-penalties to explicitly permit these violations.

The tightening technique of adding constraints is similar to the one described in Chapter 2. This chapter further introduces an extension to the tightening technique: a pruning method to prevent the search space from growing too fast and therefore allow more constraints to be added.

Chapter 6

Conclusion

This thesis has explored exact and approximate algorithms for two decoding problems in machine translation: phrase-based decoding and bidirectional word alignment. For exact algorithms, we proposed algorithms based on Lagrangian relaxation. Each algorithm involves the design of a decomposition for the specific decoding problem. We also studied extensions to the Lagrangian relaxation method that increase the convergence rate. These extensions include a tightening technique that adds constraints incrementally, an optimality-preserving pruning technique that allows more constraints to be added and a variant of beam search that utilizes the bounding property of Lagrangian relaxation. For approximate algorithm, we used local search to decode a more complex phrase-based translation model, which improves the translation quality in terms of a syntactic evaluation metric.

In this thesis, a major focus has been on Lagrangian relaxation, where a problem is decomposed into an easier problem and some side constraints. Dual decomposition is a variant of Lagrangian relaxation and is used to combine two or more models. The constraints enforced in dual decomposition are the agreement between different models. It works naturally for joint models in natural language processing (NLP) when we want to combine different models. Dual decomposition has delivered good results on several NLP tasks.

Sometimes a problem can be decomposed differently so that either Lagrangian relaxation or

dual decomposition can be applied. For example, DeNero and Macherey (2011) proposed a model for bidirectional word alignment. They decomposed the problem into two directional alignment problems and employed dual decomposition to find agreement of the two models. For the same model, we proposed a different decomposition that relaxed some constraints of the problem and used Lagrangian relaxation to enforce the constraints (Chapter 5). Different ways of decomposition might result in difference in easiness of convergence. Thus, when designing a Lagrangian relaxation based algorithm, it worths considering different ways to decompose the problem.

Lagrangian relaxation and dual decomposition have been widely applied in the field of natural language processing. It has been used in various parsers, including dependency parser (Koo et al., 2010; Martins et al., 2011), a shallow semantic parser with linguistic constraints (Martins et al., 2011), a method for combining constituency parsers with latent annotations (Roux et al., 2013). Application to other NLP areas includes biomedical event extraction (Riedel and McCallum, 2011), joint word alignment and bilingual named entity recognition (Wang et al., 2013), and multi-document summarization (Almeida and Martins, 2013).

When we design the exact algorithms based on Lagrangian relaxation, we are solving existing models, which include the standard phrase-based model (Koehn et al., 2003, 2007), and the bidirectional alignment model proposed by DeNero and Macherey (2011). Our goal is to demonstrate that we can solve the model exactly and to help us understand the model. Even though the accuracy and translation quality are not necessarily improved, the studies of the algorithm are itself interesting, and solving a model exactly let us separate model errors from optimization errors.

One direction of future work is to apply Lagrangian relaxation or dual decomposition to other decoding problems, such as coreference resolution or decipherment. Dual decomposition can also be useful when combing different steps of an NLP system. For example, it could be used to do joint tagging, parsing, and translation.

Shifting our focus on improving the translation quality, we turn to explore another side of the trade-off between a rich model and an exact decoding algorithm. We explore incorporating dependency language model into the phrase-based translation model, and use local search as our

tool to decode. Including dependency information into our model would yield more grammatically correct translation, but would at the same time increase the model complexity. Though inexact, local search allows us decode the model efficiently.

Employing local search gives us more freedom in designing the scoring function. We could further consider other information that might improve the translation quality. In addition to different level of syntactic information, we could include semantic information as well.

Another natural future direction would be designing a richer model and an exact algorithm at the same time. In terms of designing a richer model, one direction that has been explored in the machine translation area is to incorporate syntactic information into translation. We have demonstrated that adding a dependency language model could be helpful in this thesis, but there are other ways to develop a syntactic translation model. For example, an idea is to combine parsing and translation as one decoding problem.

Bibliography

Almeida, M. B. and Martins, A. F. T. (2013). Fast and robust compressive summarization with dual decomposition and multi-task learning. In *The 51st Annual Meeting of the Association for Computational Linguistics*, pages 196–206.

Ayan, N. F. and Dorr, B. J. (2006). Going beyond aer: An extensive analysis of word alignments and their impact on mt. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 9–16. Association for Computational Linguistics.

Belanger, D., Passos, A., Riedel, S., and McCallum, A. (2012). Map inference in chains using column generation. In *NIPS*, pages 1853–1861.

Blackwood, G., de Gispert, A., Brunning, J., and Byrne, W. (2009). Large-scale statistical machine translation with weighted finite state transducers. In *Proceeding of the 2009 conference on Finite-State Methods and Natural Language Processing: Post-proceedings of the 7th International Workshop FSMNLP 2008*, pages 39–49, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Boyd, S. and Mutapcic, A. (2007). Subgradient methods.

Brown, P. F., Pietra, V. J. D., Pietra, S. A. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19:263–311.

Chang, Y.-W. and Collins, M. (2011). Exact decoding of phrase-based translation models through lagrangian relaxation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 26–37. Association for Computational Linguistics.

Chang, Y.-W., Rush, A. M., DeNero, J., and Collins, M. (2014). A constrained viterbi relaxation for bidirectional word alignment. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 1481–1490. Association for Computational Linguistics.

Chiang, D. (2007). Hierarchical phrase-based translation. *computational linguistics*, 33(2):201–228.

Chiang, D. (2010). Learning to translate with source and target syntax. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1443–1452. Association for Computational Linguistics.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics.

Collins, M., Koehn, P., and Kučerová, I. (2005). Clause restructuring for statistical machine translation. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 531–540. Association for Computational Linguistics.

Cowan, B., Kučerová, I., and Collins, M. (2006). A discriminative model for tree-to-tree translation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 232–241. Association for Computational Linguistics.

Cromieres, F. and Kurohashi, S. (2009). An alignment algorithm using belief propagation and a structure-based distortion model. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 166–174. Association for Computational Linguistics.

de Gispert, A., Iglesias, G., Blackwood, G., Banga, E. R., and Byrne, W. (2010). Hierarchical Phrase-Based Translation with Weighted Finite-State Transducers and Shallow-n Grammars. *Computational linguistics*, 36(3):505–533.

DeNeefe, S. and Knight, K. (2009). Synchronous tree adjoining machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 727–736. Association for Computational Linguistics.

DeNero, J. and Klein, D. (2010). Discriminative modeling of extraction sets for machine translation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1453–1463. Association for Computational Linguistics.

DeNero, J. and Macherey, K. (2011). Model-based aligner combination using dual decomposition. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 420–429. Association for Computational Linguistics.

Dyer, C., Lopez, A., Ganitkevitch, J., Weese, J., Ture, F., Blunsom, P., Setiawan, H., Eidelman, V., and Resnik, P. (2010). cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *In Proceedings of ACL System Demonstrations*.

Eisner, J. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhagen.

Ganchev, K., Graça, J., Gillenwater, J., and Taskar, B. (2010). Posterior Regularization for Structured Latent Variable Models. *Journal of Machine Learning Research*, 11:2001–2049.

Ganchev, K., Graça, J. a. V., and Taskar, B. (2008). Better alignments = better translations? In *Proceedings of ACL-08: HLT*, pages 986–993, Columbus, Ohio. Association for Computational Linguistics.

Germann, U. (2003). Greedy decoding for statistical machine translation in almost linear time. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 1–8. Association for Computational Linguistics.

Germann, U., Jahr, M., Knight, K., Marcu, D., and Yamada, K. (2001). Fast decoding and optimal decoding for machine translation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 228–235.

Graca, J., Ganchev, K., and Taskar, B. (2008). Expectation maximization and posterior constraints. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*, pages 569–576. MIT Press, Cambridge, MA.

Gurobi Optimization, I. (2013). Gurobi optimizer reference manual.

Haghighi, A., Blitzer, J., DeNero, J., and Klein, D. (2009). Better word alignments with supervised itg models. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 923–931. Association for Computational Linguistics.

Hardmeier, C., Nivre, J., and Tiedemann, J. (2012). Document-wide decoding for phrase-based statistical machine translation. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1179–1190. Association for Computational Linguistics.

Held, M. and Karp, R. M. (1971). The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical programming*, 1(1):6–25.

Huang, L. and Chiang, D. (2007). Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151, Prague, Czech Republic. Association for Computational Linguistics.

Huang, L., Knight, K., and Joshi, A. (2006). Statistical syntax-directed translation with extended domain of locality. In *Proceedings of AMTA*, pages 66–73.

Huang, L. and Mi, H. (2010). Efficient incremental decoding for tree-to-string translation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 273–283, Cambridge, MA. Association for Computational Linguistics.

Iglesias, G., de Gispert, A., Banga, E. R., and Byrne, W. (2009). Rule filtering by pattern for efficient hierarchical translation. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 380–388, Athens, Greece. Association for Computational Linguistics.

Knight, K. (1999). Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615.

Koehn, P. (2004). Pharaoh: a beam search decoder for phrase-based statistical machine translation models. *Machine translation: From real users to research*, pages 115–124.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *Proceedings of the MT Summit*.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, NAACL '03, pages 48–54.

Komodakis, N., Paragios, N., and Tziritas, G. (2007). MRF optimization via dual decomposition: Message-passing revisited. In *Proceedings of the 11th International Conference on Computer Vision*.

Koo, T., Rush, A. M., Collins, M., Jaakkola, T., and Sontag, D. (2010). Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA. Association for Computational Linguistics.

Korte, B. and Vygen, J. (2008). *Combinatorial Optimization: Theory and Application*. Springer Verlag.

Kumar, S. and Byrne, W. (2005a). Local phrase reordering models for statistical machine translation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 161–168.

Kumar, S. and Byrne, W. (2005b). Local phrase reordering models for statistical machine translation. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 161–168, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Lacoste-Julien, S., Taskar, B., Klein, D., and Jordan, M. I. (2006). Word alignment via quadratic assignment. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 112–119. Association for Computational Linguistics.

Langlais, P., Patry, A., and Gotti, F. (2007). A greedy decoder for phrase-based statistical machine translation. *Proc. of TMI*.

Lemaréchal, C. (2001). Lagrangian Relaxation. In *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [based on a Spring School]*, pages 112–156, London, UK. Springer-Verlag.

Liang, P., Taskar, B., and Klein, D. (2006). Alignment by agreement. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 104–111. Association for Computational Linguistics.

Liu, D. and Gildea, D. (2005). Syntactic features for evaluation of machine translation. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 25–32.

Liu, Y., Liu, Q., and Lin, S. (2006). Tree-to-string alignment template for statistical machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 609–616. Association for Computational Linguistics.

Marcu, D. and Wong, W. (2002). A phrase-based, joint probability model for statistical machine translation. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 133–139. Association for Computational Linguistics.

Martin, R. K., Rardin, R. L., and Campbell, B. A. (1990). Polyhedral characterization of discrete dynamic programming. *Operations research*, 38(1):127–138.

Martins, A. F. T., Smith, N. A., Aguiar, P. M. Q., and Figueiredo, M. A. T. (2011). Dual decomposition with many overlapping components. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 238–249, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mehay, D. N. and Brew, C. (2006). Bleu atre: Flattening syntactic dependencies for mt evaluation. *TMI 2007*, page 122.

Moore, R. C. (2005). A discriminative framework for bilingual word alignment. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 81–88. Association for Computational Linguistics.

Nedić, A. and Ozdaglar, A. (2009). Approximate primal solutions and rate analysis for dual subgradient methods. *SIAM Journal on Optimization*, 19(4):1757–1780.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 7(4):308–313.

Nivre, J. (2004). Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics.

Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, ACL '03, pages 160–167.

Och, F. J., Tillmann, C., Ney, H., et al. (1999a). Improved alignment models for statistical machine translation. In *Proc. of the Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28.

Och, F. J., Tillmann, C., Ney, H., and Informatik, L. F. (1999b). Improved alignment models for statistical machine translation. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 20–28.

Och, F. J., Ueffing, N., and Ney, H. (2001). An efficient A* search algorithm for statistical machine translation. In *Proceedings of the workshop on Data-driven methods in machine translation - Volume 14*, DMMT '01, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Polyak, B. (1987). *Introduction to Optimization*. Optimization Software, Inc.

Popović, M. and Ney, H. (2009). Syntax-oriented evaluation measures for machine translation output. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 29–32. Association for Computational Linguistics.

Rasooli, M. S. and Tetreault, J. (2015). Yara parser: A fast and accurate dependency parser. *arXiv preprint arXiv:1503.06733*.

Riedel, S. and Clarke, J. (2006). Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 129–137, Stroudsburg, PA, USA. Association for Computational Linguistics.

Riedel, S. and Clarke, J. (2009). Revisiting optimal decoding for machine translation IBM model 4. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, NAACL-Short '09, pages 5–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Riedel, S. and McCallum, A. (2011). Fast and robust joint models for biomedical event extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1–12, Stroudsburg, PA, USA. Association for Computational Linguistics.

Riedel, S., Smith, D., and McCallum, A. (2012). Parse, price and cut: delayed column and row generation for graph based parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 732–743. Association for Computational Linguistics.

Roux, J. L., Rozenknop, A., and Foster, J. (2013). Combining PCFG-LA models with dual decomposition: A case study with function labels and binarization. In *Proceedings of the Empirical Methods in Natural Language Processing*, pages 1158–1169.

Rush, A. M., Chang, Y.-W., and Collins, M. (2013). Optimal beam search for machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 210–221. Association for Computational Linguistics.

Rush, A. M. and Collins, M. (2011). Exact decoding of syntactic translation models through lagrangian relaxation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 72–82.

Rush, A. M. and Collins, M. (2012). A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*, 45:305–362.

Rush, A. M., Sontag, D., Collins, M., and Jaakkola, T. (2010). On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Cambridge, MA. Association for Computational Linguistics.

Schwartz, L., Callison-Burch, C., Schuler, W., and Wu, S. (2011). Incremental syntactic language models for phrase-based translation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 620–631. Association for Computational Linguistics.

Shen, L., Xu, J., and Weischedel, R. M. (2008). A new string-to-dependency machine translation algorithm with a target dependency language model. In *ACL*, pages 577–585.

Smith, D. A. and Eisner, J. (2008). Dependency parsing by belief propagation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 145–156.

Sontag, D., Meltzer, T., Globerson, A., Jaakkola, T., and Weiss, Y. (2008). Tightening LP relaxations for MAP using message passing. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 503–510.

Tillmann, C. (2006). Efficient dynamic programming search algorithms for phrase-based SMT. In *Proceedings of the Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, CHSLP '06, pages 9–16.

Tillmann, C. and Ney, H. (2003). Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational Linguistics*, 29:97–133.

Tromble, R. W. and Eisner, J. (2006). A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 423–430, Stroudsburg, PA, USA. Association for Computational Linguistics.

Vogel, S., Ney, H., and Tillmann, C. (1996). Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841. Association for Computational Linguistics.

Wainwright, M., Jaakkola, T., and Willsky, A. (2005). MAP estimation via agreement on trees: Message-passing and linear programming. 51(11):3697–3717.

Wang, M., Che, W., and Manning, C. D. (2013). Joint word alignment and bilingual named entity recognition using dual decomposition. In *The 51st Annual Meeting of the Association for Computational Linguistics*, pages 1073–1082.

Watanabe, T., Tsukada, H., and Isozaki, H. (2006). Left-to-right target generation for hierarchical phrase-based translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 777–784, Morristown, NJ, USA. Association for Computational Linguistics.

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 523–530. Association for Computational Linguistics.

Zaslavskiy, M., Dymetman, M., and Cancedda, N. (2009). Phrase-based statistical machine translation as a traveling salesman problem. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 333–341, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zhang, Y., Lei, T., Barzilay, R., and Jaakkola, T. (2014). Greed is good if randomized: New inference for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1013–1024.

Zhang, Y., Li, C., Barzilay, R., and Darwish, K. (2015). Randomized greedy inference for joint segmentation, pos tagging and dependency parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Appendices

Appendix A

Phrase-Based Decoding

A.1 Step Size

In the subgradient method that we used to optimize the dual objective, the choice of step size affects the speed of convergence. Note that ideally the step size sequence should satisfy the assumption in Section 2.4.3.

There are several choices. In the experiments in Chapter 2, we set the step size at the tth iteration to be $\alpha^t = 1/(1 + \lambda^t)$, where λ^t is the number of times that $L(u^{(t')}) > L(u^{(t'-1)})$ for all $t' \leq t$. We define λ^t this way because we only want to decrease the step size when the dual is not improving, which prevents us from decreasing the step size too fast. This rate is also used in Koo et al. (2010).

In Chapter 5, we consider a rate that decays more slowly in the beginning: $\alpha^t = c^{\lambda^t}$ where c is a constant smaller than 1, and λ^t has the same definition as above. The choices of c decides how fast the rate is decaying.

When an lower bound is available, we can consider using Polyak's rule (Polyak, 1987; Boyd and Mutapcic, 2007) to compute step size. We substitute the unknown optimal score for the last computed lower bound: $\alpha^t \leftarrow \frac{\theta^\top y^{(t)} - \mathrm{lb}^{(t)}}{||Ay^{(t)} - b||_2^2}$. The denominator is the square of the subgradient $Ay^t - b$. In our experiment in Chapter 3, this rate outperforms those that solely depend on the number of

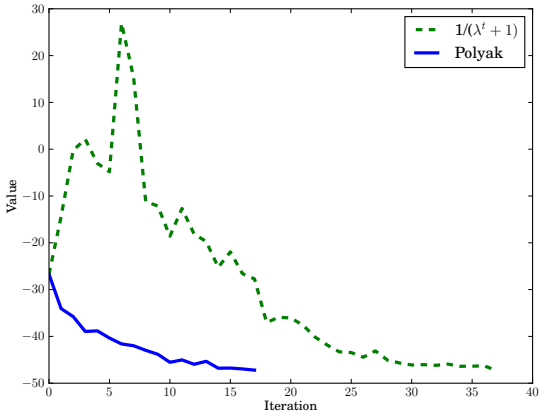


Figure A-1: The dual values $L(u^{(t)})$ at each iteration for two different choices of step size: $1/(\lambda^t + 1)$ and the rate based on Polyak's rule. The curves end when the algorithm converges. This is based on the results of one example sentence, using the algorithm in Chapter 3.

iteration. Figure A-1 compares two different choices of step size. It shows that the step size based on Polyak's rule is more stable and converges faster.

A.2 Bigram Trick to Speed up the DP Computation

This section involves some implementation details of the Lagrangian relaxation algorithm for phrase-based decoding. The dynamic program in Section 2.4.1 will be solved several times in the Lagrangian relaxation algorithm. Each time the dynamic program has the same structure but with different weights. To reuse the same structure, we construct a lattice to represent the dynamic program. Each time we solve the dynamic program, we only need to update the weights associated with each edge.

The states in the dynamic program are represented by the nodes in the lattice, and the phrases that leads from one state to another are represented by edges. Each edge is associated with a weight that includes the translation score, language model score, and the distortion scores. Solving the dynamic program is the same as finding a highest scoring path in the lattice, which can be converted into a shortest path problem.

The size of the lattice determine the computation complexity. We can reduce the size of the

lattice by a bigram trick that utilizes the sparsity of the trigram language model. This section introduces the bigram trick in details.

The key observation is that there are many trigrams that do not appear in the training data. When we are computing the trigram probability of a trigram that has never been seen in the training data, we use back-off models to compute the trigram probability. In this case, the trigram probability can be decomposed into two terms. One depends only on the first two words and one depends only on the latter two words. Thus, if there is a bigram that does not appear in any trigram seen in the training data, we can ignore the second word in the bigram by separating the two terms into the incoming and the outgoing edge of the node with the bigram. Therefore, we can collapse all such bigrams that begin with the same word into a single node in the lattice. This section details the construction.

First, we identify all bigrams (w_1, w_2) such that the trigram (w_1, w_2, w') does not exist in the language model for all w' in the vocabulary. Let $\mathcal{B}$ denote the set of such bigrams:

$$\mathcal{B} = \{(w_1, w_2) \text{ s.t. } c(w_1, w_2, w') = 0 \ \forall \ w' \in \mathcal{V}\}$$

where $c(w_1, w_2, w')$ is the number of times the trigram (w_1, w_2, w') has been seen. In the back-off language model, if the trigram does not exist, the back-off language model score can be expressed by two components:

$$q_{BO}(w'|w_1, w_2) = \alpha(w_1, w_2) q_{BO}(w'|w_2) \text{ if } c(w_1, w_2, w') = 0,$$

The back-off weights $\alpha(w_1, w_2)$ depends on the prefix (w_1, w_2) and the lower order back-off language model score $q_{BO}(w'|w_2)$ depends on the suffix (w_2, w'). For each hypothesis (w_1, w_2, b, l, m, r), where $(w_1, w_2) \in \mathcal{B}$, we no longer need to keep w_1, the first word of the ending bigram. Originally w_1 is kept for the calculation of $q_{BO}(w'|w_1, w_2)$ when we extend from the hypothesis. Now we know that $q_{BO}(w'|w_1, w_2) = \alpha(w_1, w_2) q_{BO}(w'|w_2)$. We can factored the language model score into the incoming edge and the outgoing edge. Therefore, when we reach the hypothesis

(w_1, w_2, b, l, m, r) where $(w_1, w_2) \in \mathcal{B}$, we include the back-off weight $\alpha(w_1, w_2)$ to the weights on the incoming edge, and only keep (w_2, b, l, m, r) in the hypothesis. When we extend from the hypothesis (w_2, b, l, m, r), we add the score of $q_{BO}(w'|w_2)$ to the outgoing edge.

The modification for creating a transition would be as follows. For any such phrase, we create a transition

$$(w_1, w_2, b, l, m, r) \xrightarrow{p=(s,t,e)} (w'_1, w'_2, b', l', m', r')$$

where

- $(w'_1, w'_2) = \begin{cases} (e_{M-1}, e_M) & \text{if } M \geq 2 \text{ and } (e_{M-1}, e_M) \notin \mathcal{B} \\ (NULL, e_M) & \text{if } M \geq 2 \text{ and } (e_{M-1}, e_M) \in \mathcal{B} \\ (w_2, e_1) & \text{if } M = 1 \text{ and } (w_2, e_1) \notin \mathcal{B} \\ (NULL, e_1) & \text{if } M = 1 \text{ and } (w_2, e_1) \in \mathcal{B} \end{cases}$

The modification to the language model score associated with the transition is described below. Note that the language model score is in $\log$ space.

$$h'(e_1|w_1, w_2) + h(e_2|w_2, e_1) + \sum_{i=1}^{M-2} h(e_{i+2}|e_i, e_{i+1}) + h''(e_{M-1}, e_M)$$

where

- $h'(e_1|w_1, w_2) = \begin{cases} h(e_1|w_1, w_2) & \text{if } w_1 \neq NULL \\ \log q_{BO}(e_1|w_2) & \text{if } w_1 = NULL \end{cases}$
- $h''(e_{M-1}, e_M) = \begin{cases} 0 \text{ if } (e_{M-1}, e_M) \notin \mathcal{B} \\ \log \alpha(e_{M-1}, e_M) \text{ if } (e_{M-1}, e_M) \in \mathcal{B} \end{cases}$

This modification reduces the number of nodes to 65% of its original number on average, and reduces the number of edges to 63%.

We use SriLM package to handle the language model. The interface of the language model can be found in Ngram.h. We use the function findBOW() to find the back-off weight $\alpha(w_1, w_2)$, and the function findProb() to find the lower level back-off language model score $q_{BO}(w'|w_2)$. Note that if the bigram (w_2, w') has not been seen in the language model, $q_{BO}(w'|w_2) = \alpha(w_2)q_{BO}(w')$.

Appendix B

Bidirectional Word Alignment Problem

B.1 Proof of NP-Hardness

We can show that the bidirectional alignment problem is NP-hard by reduction from the traveling salesman problem (TSP). A TSP instance with N cities has distance $c(i', i)$ for each $(i', i) \in [N]^2$. We can construct a sentence pair in which $I = J = N$ and ϵ-alignments have infinite cost.

$$\begin{aligned}
\omega(i', i, j) &= -c(i', i) \quad \forall i' \in [N]_0, i \in [N], j \in [N] \\
\theta(j', i, j) &= 0 \quad \forall j' \in [N]_0, i \in [N], j \in [N] \\
\omega(i', 0, j) &= -\infty \qquad \forall i' \in [N]_0, j \in [N] \\
\theta(j', i, 0) &= -\infty \qquad \forall j' \in [N]_0, i \in [N]
\end{aligned}$$

Every bidirectional alignment with finite objective score must align exactly one word in e to each word in f, encoding a permutation a. Moreover, each possible permutation has a finite score: the negation of the total distance to traverse the N cities in order a under distance c. Therefore, solving such a bidirectional alignment problem would find a minimal Hamiltonian path of the TSP encoded in this way, concluding the reduction.