# Exact Decoding of Phrase-Based Translation Models through Lagrangian Relaxation

by

Yin-Wen Chang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2012





Author .....
Department of Electrical Engineering and Computer Science
December 9, 2011

Certified by .....
Michael Collins
Associate Professor
Thesis Supervisor

Accepted by .....
Leslie Kolodziejski
Chairman, Department Committee on Graduate Theses

# Exact Decoding of Phrase-Based Translation Models through Lagrangian Relaxation

by

Yin-Wen Chang

Submitted to the Department of Electrical Engineering and Computer Science
on December 9, 2011, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

## Abstract

This thesis describes two algorithms for exact decoding of phrase-based translation models, based on Lagrangian relaxation. Both methods recovers exact solutions, with certificates of optimality, on over 99% of test examples. The first method is much more efficient than approaches based on linear programming (LP) or integer linear programming (ILP) solvers: these methods are not feasible for anything other than short sentences. We compare our methods to MOSES [6], and give precise estimates of the number and magnitude of search errors that MOSES makes.

Thesis Supervisor: Michael Collins
Title: Associate Professor

# Acknowledgments

First of all, I would like to thank my advisor Michael Collins. Working with him lets me experience the excitement of doing research. My group mate Sasha also provides invaluable resources on this project. I am looking forward to continueing working with them.

I also want to thank my former advisor Patrick Jaillet and Cynthia Rudin. Working with them enlightens me on the usefulness of machine learning.

My friends Ann, Dawsen, Owen, Sarah, Yu are great companies in my daily life at MIT. With Sarah, I saw the beauty of algorithms, and Yu often gives great suggestions in both research ideas and course projects. I would like to thank my Friends and former labmates in Taiwan, Peng-Jen, Cho-Jui, Kai-Wei, Hsiang-Fu, Li-Jen, Rong-En, Wen-Hsin and Ming-Hen who have given me great support while I tried to adjust to the new life in Boston.

Finally, I would like to thank my parents and sister for their support on many aspects of life, throughout the years of my pursuit of education.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Phrase-based models [15, 7, 6] 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 thesis describes two algorithm for exact decoding of phrase-based models, based on Lagrangian relaxation [12]. The core of the first 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 thesis 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 sec-

[1] We refer here to the phrase-based models of [7, 6], considered in this thesis. Other variants of phrase-based models, which allow polynomial time decoding, have been proposed, see the related work section.

ond technical contribution of this thesis 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 [6]. 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 used in MOSES.

The second decoding algorithm, also based on Lagrangian relaxation, presents an alternative way to decompose the problem. In this algorithm, we make the dynamic programming more efficient by avoiding keeping track of the language model score. Then, we incorporate the language model score by using Lagrange multipliers to achieve agreement between the results of two subproblems. The two subproblems are a Lagrangian relaxation algorithm very similar to our first method, and a method to find the highest scoring trigram for each word assuming that the word is at the ending position.

The reminder of the thesis is structured as follows. In Section 1.1, we discuss related work. Chapter 2 will introduce the phrase-based translation models and describe a Lagrangian relaxation algorithm for decoding the phrase-based translation models exactly. Chapter 3 presents an alternative Lagrangian relaxation algorithm that exploits a dynamic program that is more efficient. Chapter 4 gives the discussion and conclusion.

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

## 1.1 Related Work

Lagrangian relaxation is a classical technique for solving combinatorial optimization problems [10, 12]. Dual decomposition, a special case of Lagrangian relaxation, has been applied to inference problems in NLP [9, 21], and also to Markov random fields [27, 8, 23]. Earlier work on belief propagation [22] is closely related to dual decomposition. Recently, [20] describe a Lagrangian relaxation algorithm for decoding for syntactic translation; the algorithmic construction described in the first algorithm of the current thesis is, however, very different in nature to this work.

Beam search stack decoders [7] 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 [25] and [24]. Greedy decoding [4] is an alternative approximate search method, which is again efficient, but has no guarantee of returning optimal translations.

Several works attempt exact decoding, but efficiency remains an issue. Exact decoding via integer linear programming (ILP) for IBM model 4 [2] has been studied by [4], with experiments using a bigram language model for sentences up to eight words in length. [19] 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). [28] 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. [16] propose an A* search algorithm for IBM model 4, and test on sentence lengths up to 14 words. Other work [11, 1] 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 [23]; work that encodes constraints using finite-state machines [26]; and work on non-projective dependency parsing [18].

# Chapter 2

# A Decoding Algorithm based on Lagrangian Relaxation

In this chapter, we will describe the phrase-based translation models and the decoding problem. Then we will introduce a decoding algorithm based on Lagrangian relaxation. The core of the algorithm is a dynamic program, which is efficient, but which allows ill-formed derivations. The constraints specifying a valid derivation will be introduced by Lagrangian relaxation method. Formal properties of the algorithm and the relationship to linear programming relaxations are included. We also introduce a method to incrementally tighten the relaxation until convergence. Experiments on translations from German to English have shown that the method is efficient in practice. The major part of this chapter was originally published as [3].

## 2.1 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 [7], as implemented in MOSES [6].

The input to a phrase-based translation system is a source-language sentence with $N$ words, $x_1 x_2 \ldots x_N$. 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 $x_2 \ldots x_5$ 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 [7]).

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 $y = \langle p_1 p_2 \ldots p_L \rangle$. 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 $e(p_1), e(p_2), \ldots, e(p_L)$. 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

$$f(y) = h(e(y)) + \sum_{k=1}^{L} g(p_k) + \sum_{k=1}^{L-1} \eta \times \delta(t(p_k), s(p_{k+1})) \tag{2.1}$$

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.[1]

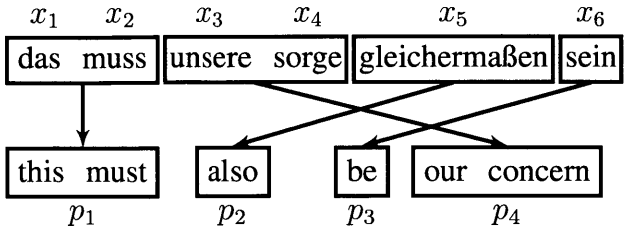
The decoding problem is to find

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

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 $\pi$ 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*.

[1] 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 $\eta$, are typically chosen using MERT training [14].




- phrase $p = (s, t, e)$

  (1, 2, this must), (5, 5, also), (6, 6, be), (3, 4, our concern)

- derivation

  $y = p_1, p_2, \ldots, p_L$

Figure 2-1: An example illustrating the notion of phrases and derivations used in this thesis.

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 [7]). 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.

## 2.2 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.

### 2.2.1 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 $e = (e_0 \ldots e_{M-1} e_M)$ 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 $(s, t)$.

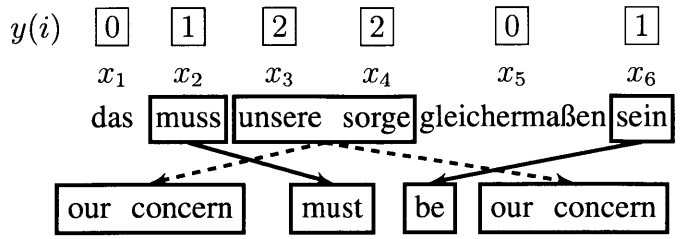
The score of the transition is given by a sum of the phrase translation score $g(p)$, the language model score, and the distortion cost $\eta \times \delta(r, s)$. The trigram language model score is $h(e_1|w_1, w_2) + h(e_2|w_2, e_1) + \sum_{i=1}^{M-2} h(e_{i+2}|e_i, e_{i+1})$, where $h(w_3|w_1, w_2)$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




$y$ = (3, 4, our concern), (2, 2, must), (6, 6, be), (3, 4, our concern)

Figure 2-2: An example of an ill-formed derivation in the set $\mathcal{Y}'$. Here we have $y(1) = y(5) = 0$, $y(2) = y(6) = 1$, and $y(3) = y(4) = 2$. Some words are translated more than once and some words are not translated at all. However, the sum of the number of source-language words translated is equal to 6, which is the length ($N$) of the sentence.

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 2-4 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 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 2-3: The decoding algorithm. $\alpha^t > 0$ is the step size at the $t$'th iteration.

### 2.2.2 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 $\mathcal{Y} = \{y : y \in \mathcal{Y}', \text{ and } \forall i,\ y(i) = 1\}$. The original decoding problem can therefore be stated as:

$$\arg\max_{y \in \mathcal{Y}'} f(y) \text{ such that } \forall i,\ y(i) = 1 \tag{2.2}$$

We use Lagrangian relaxation [10] 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

$$L(u) = \max_{y \in \mathcal{Y}'} L(u, y). \tag{2.3}$$

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 [10], $L(u)$ 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 $\gamma_u$ 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 experiments, the step size decreases each time the dual value increases from one iteration to the next. Similar to [9], we set the step size at the $t$'th iteration to be $\alpha^t = 1/(1 + \lambda^t)$, where $\lambda^t$ is the number of times that $L(u^{(t')}) > L(u^{(t'-1)})$ for all $t' \leq t$.

Figure 2-3 depicts the resulting algorithm. At each iteration, we solve

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

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 $y^t(i) = 1$.

### 2.2.3 Properties

We now give some theorems stating formal properties of the Lagrangian relaxation algorithm. The proofs are simple, and are well known results for Lagrangian relaxation—for

**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 2-4: An example run of the algorithm in Figure 2-3. 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.

completeness, we state them here. 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.*

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

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 $\alpha^t$, 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 [10]. □

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

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

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 some cases, however, the algorithm in Figure 2-3 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 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 2.4 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.

### 2.2.4 An Example Run of the Algorithm

Figure 2-4 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 Lagrange 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.

## 2.3 Relationship to Linear Programming Relaxations

This section explains the relationship between Lagrangian relaxation and linear programming relaxations. The algorithm we described is minimizing the dual of a particular linear programming relaxation problem given by the set $\mathcal{Y}'$ and the constraints that $y(i) = 1$ for all $i$. The algorithm converges if the solution to the relaxed problem is integral.

### 2.3.1 The Linear Programming Relaxation

We first describe the optimization over a simplex. We define $\Delta_{\mathcal{Y}'}$ to be the simplex over elements in $\mathcal{Y}'$:

$$\Delta_{\mathcal{Y}'} = \{\alpha : \alpha \in \mathbb{R}^{|\mathcal{Y}'|}, \sum_y \alpha_y = 1, \ 0 \leq \alpha_y \leq 1 \ \forall y\}$$

Each $\alpha \in \Delta_{\mathcal{Y}'}$ is a distribution over $\mathcal{Y}'$, and the simplex corresponds to the set of all distributions over elements in $\mathcal{Y}'$. Each dimension of $\alpha$ represents a derivation in the set $\mathcal{Y}'$. Suppose that a binary vector $\alpha$ has 1 for only one dimension, and 0 for all other

dimensions, every such $\alpha$ specifies a derivation. Also notice that those $\alpha$'s that represent derivations in $\mathcal{Y}'$ are the *vertices* of the set $\Delta_{\mathcal{Y}'}$.

We define a new optimization program over the simplex $\Delta_{\mathcal{Y}'}$:

$$\begin{aligned} \arg\max_{\alpha \in \Delta_{\mathcal{Y}'}} & \sum_y \alpha_y f(y) \\ \text{s.t.} & \sum_y \alpha_y y(i) = 1 \text{ for } i = 1 \ldots n \end{aligned} \tag{2.4}$$

The constraint states that, in expectation, the number of times that word $i$ is translated should be exactly one. The highest scoring distribution no longer specifies a single derivation. Instead, it can be the combination of several derivations.

This problem is a linear program, since both the objective and the constraints are linear with respect to the $\alpha$ variables.

This optimization problem is very similar to our original problem described in equation (2.2). To illustrate the connection, we define $\Delta'_{\mathcal{Y}'}$ as follows:

$$\Delta'_{\mathcal{Y}'} = \{\alpha : \alpha \in \mathbb{R}^{|\mathcal{Y}'|}, \sum_y \alpha_y = 1, \ \alpha_y \in \{0, 1\} \ \forall y\}$$

$\Delta'_{\mathcal{Y}'}$ is a subset of $\Delta_{\mathcal{Y}'}$, where the constraints $0 \leq \alpha_y \leq 1$ have been replaced by $\alpha_y \in \{0, 1\}$.

Each element in the set $\Delta'_{\mathcal{Y}'}$ corresponds to a derivation in the set $\mathcal{Y}'$. More formally, let $\delta : \mathcal{Y}' \rightarrow \mathbb{R}^{|\mathcal{Y}'|}$ denote the function that maps a derivation to a vector in a $|\mathcal{Y}'|$ dimensional space. Then $\Delta'_{\mathcal{Y}'} = \{\delta(y) : y \in \mathcal{Y}'\}$.

Consider the following optimization problem, where we replace $\Delta_{\mathcal{Y}'}$ in equation (2.4) by $\Delta'_{\mathcal{Y}'}$:

$$\begin{aligned} \arg\max_{\alpha \in \Delta'_{\mathcal{Y}'}} & \sum_y \alpha_y f(y) \\ \text{s.t.} & \sum_y \alpha_y y(i) = 1 \text{ for } i = 1 \ldots n \end{aligned} \tag{2.5}$$

This optimization problem is an integer linear program, since both the objective and the

constraints are linear with respect to $\alpha$, and $\alpha$ are constrained to be either 0 or 1. Also, $\Delta_{\mathcal{Y}'}$ is the *convex hull* of the set $\Delta'_{\mathcal{Y}'}$. The elements in $\Delta'_{\mathcal{Y}'}$ form the vertices of the polytope $\Delta_{\mathcal{Y}'}$. Thus, the optimization problem in equation (2.4) is a relaxation of this problem. The relaxed problem replace the constraints $\alpha_y \in \{0, 1\}$ by the constraints $0 \leq \alpha_y \leq 1$.

Since a vector $\alpha \in \Delta'_{\mathcal{Y}'}$ represents a derivation in the set $\mathcal{Y}'$, this new problem (2.5) is equivalent to our original problem in equation (2.2). Thus, we can view the optimization in equation (2.4) as a relaxation of our original problem.

The following theorem states that optimizing over a discrete set $\mathcal{Y}'$ can be replaced by optimizing over the simplex $\Delta_{\mathcal{Y}'}$. This theorem will be useful later on.

**Theorem 4.** *For any finite set $\mathcal{Y}'$, and any function $f$: $\mathcal{Y}' \to \mathbb{R}$*

$$\max_{y \in \mathcal{Y}'} f(y) = \max_{\alpha \in \Delta_{\mathcal{Y}'}} \sum_y \alpha_y f(y)$$

This is true since the optimal value of linear program is always at the vertices of the polytope, and points in $\mathcal{Y}'$ correspond to vertices of the simplex $\Delta'_{\mathcal{Y}'}$. More specifically, The maximum of linear program over a polytope $\Delta_{\mathcal{Y}'}$ can always be achieved at a vertex of the polytope:

$$\max_{\alpha \in \Delta_{\mathcal{Y}'}} \sum_y \alpha_y f(y) = \max_{\alpha \in \Delta'_{\mathcal{Y}'}} \sum_y \alpha_y f(y),$$

Since a derivation in $\mathcal{Y}'$ corresponds to a vector in $\Delta'_{\mathcal{Y}'}$, we have

$$\max_{y \in \mathcal{Y}'} f(y) = \max_{\alpha \in \Delta'_{\mathcal{Y}'}} \sum_y \alpha_y f(y).$$

[10] provides a full proof.

### 2.3.2 The Dual of the New Optimization Problem

We now describe the dual problem of the optimization problem in equation (2.4). This will be a function $M(u)$ of dual variables $u = \{u(i) : i \in \{1 \ldots n\}\}$. We will show that the dual problem $M(u)$ is identical to $L(u)$ in equation (2.3), the dual problem of the original problem.

The Lagrangian of the problem in equation (2.4) is

$$M(u, \alpha) = \sum_y \alpha_y f(y) + \sum_i u(i) \left( \sum_y \alpha_y y(i) - 1 \right)$$

The Lagrangian dual is

$$M(u) = \max_{\alpha \in \Delta_{y'}} M(u, \alpha)$$

and the dual problem is to solve

$$\min_u M(u)$$

In the following, we will describe two theorems regarding the dual problem. We first define $\alpha^*$ to be the optimal solution for the linear program.

**Definition 3.**

$$\alpha^* = \arg\max_{\alpha \in \Delta_{y'}} \sum_y \alpha_y f(y)$$
$$s.t. \quad \sum_y \alpha_y y(i) = 1 \ \textit{for} \ \ i = 1 \ldots n$$

By strong duality, we have the following theorem, stating that the solution of the dual problem is the maximum of the primal problem.

**Theorem 5.**

$$\min_u M(u) = \sum_y \alpha_y^* f(y)$$

Note that in our previous result (Theorem 1), the dual solution only gives an upper bound on the primal solution:

$$\min_u L(u) \geq f(y^*)$$

Now we have equality in the above theorem, which means that the dual solution will be equal to the primal solution.

The second theorem states that solving the original Lagrangian dual also solves the dual of the linear program.

**Theorem 6.** *For any value of $u$,*

$$M(u) = L(u).$$

*Proof.* This theorem follows from Theorem 4, since $M(u, \alpha) = \sum_y \alpha_y L(u, y)$:

$$\begin{aligned}
M(u,\alpha) &= \sum_y \alpha_y f(y) + \sum_i u(i) \left( \sum_y \alpha_y y(i) - 1 \right) \\
&= \sum_y \alpha_y f(y) + \sum_i \sum_y \alpha_y u(i) y(i) - \sum_i u(i) \\
&= \sum_y \alpha_y f(y) + \sum_y \alpha_y \sum_i u(i) y(i) - \sum_y \alpha_y \sum_i u(i) \qquad (2.6) \\
&= \sum_y \alpha_y \left( f(y) + \sum_i u(i) y(i) - \sum_i u(i) \right) \\
&= \sum_y \alpha_y \left( f(y) + \sum_i u(i) \left( y(i) - 1 \right) \right) \\
&= \sum_y \alpha_y L(u, y)
\end{aligned}$$

(2.7)

The last term of equation (2.6) follows by the fact that $\sum_y \alpha_y = 1$.

Then,

$$L(u) = \max_{y \in \mathcal{Y}'} L(u, y) = \max_{\alpha \in \Delta_{\mathcal{Y}'}} \sum_y \alpha_y L(u, \alpha) = \max_{\alpha \in \Delta_{\mathcal{Y}'}} M(u, \alpha) = M(u)$$

The second equality follows by Theorem 4, □

This theorem says that the two dual functions are identical. Thus, the algorithm described in Figure 2-3, which minimizes $L(u)$, also minimizes $M(u)$.

### 2.3.3 The Relationship between the Two Primal Problems

To explain the relationship between the original primal problem and the primal problem of the linear program, we introduce the following notations. Let $\mathcal{Q} \subseteq \Delta_{\mathcal{Y}'}$ be the set corresponding to the feasible solutions of the original problem (2.2), which are also the

valid derivations.

$$\mathcal{Q} = \{\delta(y) : y \in \mathcal{Y}\}$$

Note that $\mathcal{Y} = \{y : y \in \mathcal{Y}',\ y(i) = 1\ \forall i = 1 \ldots N\}$

Let $\mathcal{Q}' \subseteq \Delta_{\mathcal{Y}'}$ be the set of feasible solutions to the linear program.

$$\mathcal{Q}' = \{\alpha : \alpha \in \Delta_{\mathcal{Y}'}, \sum_y \alpha_y y(i) = 1\ \forall i = 1 \ldots N\}$$

Note that the set $\mathcal{Q}$ is a subset of the set $\mathcal{Q}'$ since $\mathcal{Q}$ contains only vertices that represent valid derivations, while $\mathcal{Q}'$ allows fractional solution that is a combination of more than one derivation. This happens since the "exactly once" constraints are enforced in expectation. Also, the convex hull of $\mathcal{Q}$ conv($\mathcal{Q}$) is a subset of the set $\mathcal{Q}'$. This is because conv($\mathcal{Q}$) contains only combinations of valid derivations, while $\mathcal{Q}$ allowed combinations of ill-formed derivations.

- $\mathcal{Q} \subseteq \mathcal{Q}'$
- conv($\mathcal{Q}$) $\subseteq \mathcal{Q}'$

By the definition of the set $\mathcal{Q}$, each element of $\mathcal{Q}$ corresponds to a valid derivation, and, therefore, is a vector of only integral values. Thus,

$$\max_{y \in \mathcal{Y}} f(y) = \max_{\alpha \in Q} \sum_y \alpha_y f(y).$$

Since $\mathcal{Q} \subseteq \mathcal{Q}'$, we have

$$\max_{q \in \mathcal{Q}} \sum_y \alpha_y f(y) \leq \max_{q \in \mathcal{Q}'} \sum_y \alpha_y f(y)$$

.

Combining the above results, we have

$$\max_{y \in \mathcal{Y}} f(y) = \max_{\alpha \in Q} \sum_y \alpha_y f(y) \leq \max_{q \in \mathcal{Q}'} \sum_y \alpha_y f(y)$$

If the linear programming relaxation is tight, the equality in the above equation will hold, which implies that the solution is integral. In this case, solving the linear programming relaxation equals to solving the original problem. However, in the case that the relaxation is not tight, the optimal solution to the linear program (2.4) will be a fractional solution which has a higher score than the original primal optimal solution. This also means that there is a gap between the dual solution and the primal solution for the original problem. Thus, the algorithm in Figure 2-3 will not converge. Instead, it will alternate between two or more derivations. These derivations are those that could form the optimal solution for the linear program by the distribuion specified by the fractional solution. In the next section, we give an example to illustrate this case. In Section 2.4, we describe a tightening technique that tightens the relaxation by incrementally adding more constraints to further restrict the set.

### 2.3.4 An Example

In this section, we give an example to illustrate the relationship between the Lagrangian relaxation and the linear programming relaxation. The example also illustrates the case when the algorithm alternates between two derivations and cannot converge to a single valid derivation. The two derivations correspond to a fractional solution of the corresponding linear programming relaxation. We draw the example from the full example in Figure 2-6.

In this example, we assume there are three possible derivations within the set $\mathcal{Y}'$. Suppose that $\mathcal{Y}' = \{y_1, y_2, y_3\}$, and the derivations are described as follows:

$$y_1 = \left|\begin{array}{c|c|c|c|c|c|c|c|c} 1,5 & 6,6 & 8,9 & 6,6 & 7,7 & 11,12 & 16,16 & 13,15 & 17,17 \\ \text{nonetheless ,} & \text{that} & \text{a country} & \text{that} & \text{colombia} & \text{, which} & \text{must} & \text{be closely monitored} & \text{.} \end{array}\right|$$

$$y_2 = \left|\begin{array}{c|c|c|c|c|c|c|c} 1,5 & 7,7 & 10,10 & 8,8 & 9,12 & 16,16 & 13,15 & 17,17 \\ \text{nonetheless ,} & \text{colombia} & \text{is} & \text{a} & \text{country that} & \text{must} & \text{be closely monitored} & \text{.} \end{array}\right|$$

$$y_3 = \left|\begin{array}{c|c|c|c|c|c|c} 1,5 & 7,7 & 6,6 & 8,12 & 16,16 & 13,15 & 17,17 \\ \text{nonetheless ,} & \text{colombia} & \text{that} & \text{a country that} & \text{must} & \text{be closely monitored} & \text{.} \end{array}\right|$$

The scores of the derivations, together with the number of times each word has been

translated in the derivations, are as follows:

$$f(y_1) = -18.3299 \quad y_1(i) = 11111211101111111$$
$$f(y_2) = -16.0169 \quad y_2(i) = 11111011121111111$$
$$f(y_3) = -17.2290 \quad y_3(i) = 11111111111111111$$

The derivations can be represented as vectors in the set $\Delta_{\mathcal{Y}'}$:

$$\delta(y_1) = (1, 0, 0)$$
$$\delta(y_2) = (0, 1, 0)$$
$$\delta(y_3) = (0, 0, 1)$$

We first consider the primal problem of the original problem (2.2). In this example, there is only one valid derivation: $y_3$. Thus, the set of feasible solutions of the original problem is $\mathcal{Y} = \{y_3\}$. The highest scoring derivation is therefore $y_3$ and the highest score is $f(y_3)$. This will be the primal solution to our original problem.

$$y_3 = \arg\max_{y \in \mathcal{Y}} f(y)$$

and,

$$\max_{y \in \mathcal{Y}} f(y) = f(y_3) = -17.2290$$

Next, we will look at the optimization over the simplex (2.4). We consider two vectors $\alpha^1 = [0, 0, 1]$, and $\alpha^2 = [0.5, 0.5, 0]$, which represent two distributions over the set $\mathcal{Y}'$.

The first vector $\alpha^1 = [0, 0, 1]$ corresponds to the highest scoring derivation. It satisfies the constraints that $\sum_y \alpha_y y(i) = 1$ for all $i = 1 \ldots N$, since $\sum_y \alpha_y y(i) = y_3(i) = 1$ for all

$i$. Thus, we have $\alpha^1 \in \mathcal{Q}$. This is an integral solution to the linear program (2.4), which gives score:

$$\sum_y \alpha_y f(y) = \alpha_{y_3} \times f(y_3) = -17.2290$$

Then we consider the second vector $\alpha^2 = [0.5, 0.5, 0]$, which represents a combination of two derivations. We can see that

$$\sum_y \alpha_y y(i) = 0.5 \times y_1(i) + 0.5 \times y_2(i) = 1$$

for all $i = 1 \ldots N$. Thus, $\alpha^2 \in \mathcal{Q}'$. Then we consider the score of $\sum_y \alpha_y^2 f(y)$:

$$\begin{aligned}\sum_y \alpha_y^2 f(y) =& 0.5 \times f(y_1) + 0.5 \times f(y_2) + 0 \times f(y_3) \\ =& 0.5 \times -18.3299 + 0.5 \times -16.0169 \\ =& -17.1734\end{aligned}$$

Thus, $\alpha^2$ can achieve a higher score than $\alpha^1$.

When we consider optimizing over the simplex $\Delta_{\mathcal{Y}'}$.

$$\alpha^* = \arg\max_{\alpha \in \mathcal{Q}'} \sum_{y \in \mathcal{Y}} \alpha_y f(y)$$

We will have $\alpha^2$ as the optimal solution to the linear program. Thus, combining derivations $y_1$ and $y_2$ will give a higher score than the valid derivation $y_3$ alone, when we are optimizing over the simplex.

$$\max_{\alpha \in \mathcal{Q}'} \sum_y \alpha_y f(y) > \max_{\alpha \in \mathcal{Q}} \sum_y \alpha_y f(y) = \max_{y \in \mathcal{Y}} f(y).$$

This is the case when solving the linear programming relaxation does not equal to solving the original problem. The primal solution of the linear programming is larger than the primal solution of the original problem.

According to Theorem 5, for the linear program, the solution of the dual problem is the maximum of the primal problem. Thus, $\min_u M(u) = -17.1734$. Then we have $\min_u L(u) = -17.1734$ by Theorem 6.

On the other hand, the solution to the primal problem of the original problem is $y^* = y_3$. We have

$$\min_u L(u) = -17.1734 > f(y^*) = -17.2290.$$

Thus, there is a gap between the dual optimal solution and the primal optimal solution for the original problem.

## 2.4 Tightening the Relaxation

In some cases the algorithm in Figure 2-3 will not converge to $y(i) = 1$ for $i = 1 \ldots N$ 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.

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 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.[2]

Figure 2-5 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 $\mathcal{C} \subseteq \{1 \ldots N\}$ we have $\mathcal{Y} \subseteq \mathcal{Y}'_{\mathcal{C}}$; hence the theorems in section 2.2.3 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 $\mathcal{C} = \{1 \ldots N\}$, 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

[2]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 [13]. Our method eliminates the fractional solution through the introduction of hard constraints.

*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 $count(i)$, that are not in $\mathcal{C}$, and that are not adjacent to each other
**return** *Optimize*$(\mathcal{C} \cup \mathcal{C}', u)$

Figure 2-5: 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 each time the dual value increases from one iteration to the next.

first and second best dual values $L(u')$ and $L(u'')$ in the sequence of Lagrange multipliers $u^1, u^2, \ldots$ 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 $\epsilon$ is a parameter of the approach: in experiments we used $\epsilon = 0.002$.

### 2.4.1 An Example Run of the Algorithm with Tightening Method

Figure 2-6 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.

**Input German:** es bleibt jedoch dabei , dass kolumbien ein land ist , das aufmerksam beobachtet werden muss .

| $t$ | $L(u^{t-1})$ | $y^t(i)$ | derivation $y^t$ |
|---|---|---|---|
| 1 | -11.8658 | 0 0 0 0 1 3 0 3 3 4 1 1 0 0 0 0 1 | 5, 6 that \| 10, 10 is \| 8, 9 a country \| 6, 6 that \| 10, 10 is \| 8, 9 a country \| 6, 6 that \| 10, 10 is \| 8, 8 a \| 9, 12 country that \| 17, 17 . |
| 2 | -5.46647 | 2 2 4 0 2 0 1 0 0 0 1 0 1 1 1 1 | 3, 3 however , \| 1, 1 it \| 2, 3 is , however \| 5, 5 , \| 3, 3 however , \| 1, 1 it \| 2, 3 is , however \| 5, 5 , \| 7, 7 colombia \| 11, 11 , \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| ⋮ | | | |
| 32 | -17.0203 | 1 1 1 1 1 0 1 1 1 2 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 7, 7 colombia \| 10, 10 is \| 8, 8 a \| 9, 12 country that \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| 33 | -17.1727 | 1 1 1 1 1 2 1 1 1 0 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 6, 6 that \| 8, 9 a country \| 6, 6 that \| 7, 7 colombia \| 11, 12 , which \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| 34 | -17.0203 | 1 1 1 1 1 0 1 1 1 2 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 7, 7 colombia \| 10, 10 is \| 8, 8 a \| 9, 12 country that \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| 35 | -17.1631 | 1 1 1 1 1 0 1 1 1 2 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 7, 7 colombia \| 10, 10 is \| 8, 8 a \| 9, 12 country that \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| 36 | -17.0408 | 1 1 1 1 1 2 1 1 1 0 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 6, 6 that \| 8, 9 a country \| 6, 6 that \| 7, 7 colombia \| 11, 12 , which \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| 37 | -17.1727 | 1 1 1 1 1 0 1 1 1 2 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 7, 7 colombia \| 10, 10 is \| 8, 8 a \| 9, 12 country that \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| 38 | -17.0408 | 1 1 1 1 1 2 1 1 1 0 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 6, 6 that \| 8, 9 a country \| 6, 6 that \| 7, 7 colombia \| 11, 12 , which \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| 39 | -17.1658 | 1 1 1 1 1 2 1 1 1 0 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 6, 6 that \| 8, 9 a country \| 6, 6 that \| 7, 7 colombia \| 11, 12 , which \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| 40 | -17.056 | 1 1 1 1 1 0 1 1 1 2 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 7, 7 colombia \| 10, 10 is \| 8, 8 a \| 9, 12 country that \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| 41 | -17.1732 | 1 1 1 1 1 2 1 1 1 0 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 6, 6 that \| 8, 9 a country \| 6, 6 that \| 7, 7 colombia \| 11, 12 , which \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |
| | | 0 0 0 0 0 • 0 0 0 • 0 0 0 0 0 0 0 | $count(6) = 10$; $count(10) = 10$; $count(i) = 0$ for all other $i$<br>**adding constraints: 6 10** |
| 42 | -17.229 | 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 | 1, 5 nonetheless , \| 7, 7 colombia \| 6, 6 that \| 8, 12 a country that \| 16, 16 must \| 13, 15 be closely monitored \| 17, 17 . |

Figure 2-6: An example run of the algorithm in Figure 2-5. 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 $i = 1 \ldots N$, and the translation is returned, with a guarantee that it is optimal.

## 2.5 Speeding up the DP: A* Search

In the algorithm depicted in Figure 2-5, 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 ($\mathcal{C} = \emptyset$).

| # 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.

| # 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 2-5, 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.

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 2.6 presents comparison of the running time with and without A* algorithm.

| # 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 2-5, 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*.

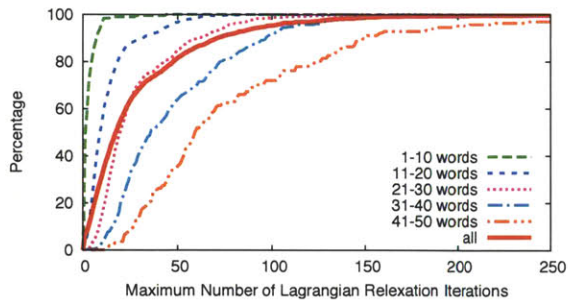
## 2.6 Experiments

In this section, we present experimental results to demonstrate the efficiency of the decoding algorithm. We compare to MOSES [6], a phrase-based decoder using beam search, and to a general purpose integer linear programming (ILP) solver, which solves the problem exactly.

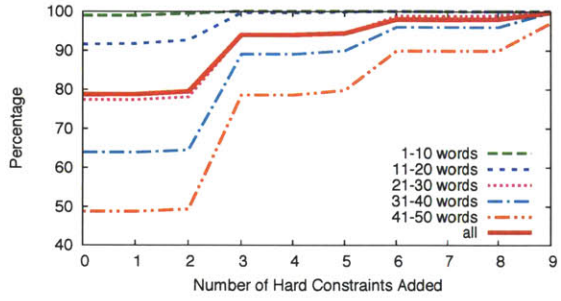
The experiments focus on translation from German to English, using the Europarl data [5]. We tested on 1,824 sentences of length at most 50 words. The experiments use the algorithm shown in Figure 2-5. We limit the algorithm to a maximum of 250 iterations and a maximum of 9 hard constraints. The distortion 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. Figure 2-7(a) shows the percentage of sentences that converged before certain number of iterations, while Figure 2-7(b) shows the percentage of sentences that converged with less than certain number of constraints. 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.

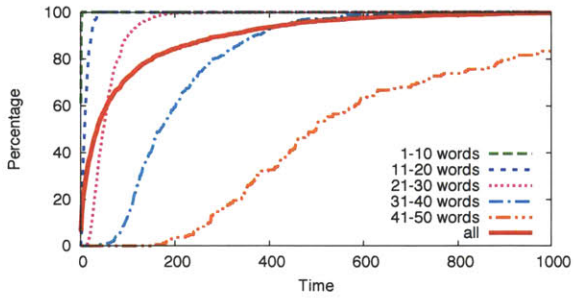
Table 2.3 shows the average times for decoding, broken down by sentence length, and by the number of constraints that are added. As expected, decoding times increase as the




(a) Number of iterations




(b) Number of constraints




(c) Time required

Figure 2-7: Percentage of sentences that converged with less than certain number of iterations/constraints.

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.

### 2.6.1 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 very long running times on these shorter

| 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 2.2.1, 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.

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).

### 2.6.2 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 2.1 is the same as that in [7], and is the same as that described in the user manual for MOSES [6]. However, a complicating factor for our comparisons is that MOSES uses an additional distortion constraint, not documented in the manual, which we describe here.[3] 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.

[3] 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.

**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.

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

**BLEU Scores** Finally, table 2.7 gives BLEU scores [17] for decoding using MOSES and our method. The BLEU scores under the two decoders are almost identical; hence while MOSES makes a significant proportion of search errors, these search errors appear to be benign in terms of their impact on BLEU scores, at least for this particular translation model. Future work should investigate why this is the case, and whether this applies to other models and language pairs.

In table 2.8, we also measure the BLEU scores for Chinese to English translations. We tested on sentences with length 1-30 words. Same as the results for German-English translations, the BLEU scores are similar under the two decoders.

| 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.

| type of Moses | beam size | # sents | Moses | our method |
|---|---|---|---|---|
| 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: BLEU score comparisons. We consider only those sentences where both decoders produce a translation.

| type of Moses | beam size | # sents | Moses | our method |
|---|---|---|---|---|
| MOSES-gc | 100 | 1,135 | 25.56 | 25.66 |
| | 200 | 1,135 | 25.67 | 25.66 |
| | 1,000 | 1,135 | 25.68 | 25.66 |
| MOSES-nogc | 100 | 937 | 24.97 | 25.77 |
| | 200 | 1,008 | 24.81 | 25.61 |
| | 1,000 | 1,105 | 25.13 | 25.60 |

Table 2.8: BLEU score comparisons for translation from Chinese to English. We consider only those sentences where both decoders produce a translation.

# Chapter 3

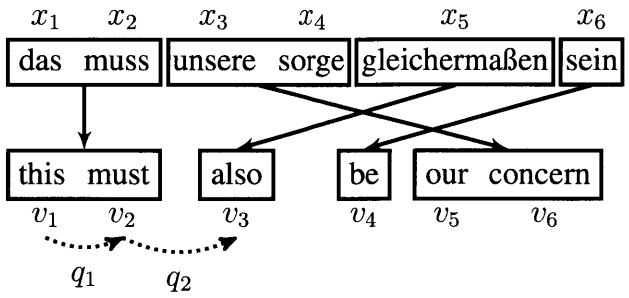
# An Alternative Decoding Algorithm based on Lagrangian Relaxation

In the approach described in Chapter 2, the dynamic program states are tuples $(w_1, w_2, n, l, m, r)$. The number of states is large: in particular, the number of bigrams $(w_1, w_2)$ is multiplied by the number of settings for $(n, l, m, r)$. In this chapter, we describe an alternative Lagrangian relaxation method that is inspired by the method proposed by [20], which intersects a hypergraph and a language model.

## 3.1 An Alternative Decoding Algorithm based on Lagrangian Relaxation

We now describe an alternative decoding algorithm based on Lagrangian relaxation. The algorithm decomposes the problem in a different way than the one described in Section 2.2. In this algorithm, we use Lagrangian relaxation to decompose the problem into two subproblems such that each subproblem can be solved efficiently. This closely follows the algorithm proposed by Rush and Collins in [20]. The two subproblems are:

1. Dynamic programming that decodes the phrase-based translation models without considering the language model score. This reduces the number of states of the dynamic program tremendously and therefore decoding this problem is more efficient




- leaves: $v_1, v_2, \ldots, v_6$
- trigram path: $(v_1, q_1, v_2, q_2, v_3)$

  $q_1 = \mathsf{NULL}$

  $q_2 = (2, 5)$

Figure 3-1: An example illustrating the notion of leaves and trigram paths used in this thesis. The path $q_1$ is NULL since $v_1$ and $v_2$ are in the same phrase $(1, 2, \text{this must})$. The path $q_2$ specifies a transition $(2, 5)$ since $v_1$ is the ending word of phrase $(1, 2, \text{this must})$, which ends at position 2, and $v_2$ is the starting word of the phrase $(5, 5, \text{also})$, which starts at position 5. This is the same derivation as in Figure 2-1.

than the dynamic program described in 2.2.1.

2. Calculating the highest scoring incoming trigram path for each *leaf*. This part is used to incorporate language model scores into the model.

Lagrange multipliers are used to encourage the agreement between the decoding result and the best incoming trigram path for each leaf.

We begin by some definitions. In addition to the notations we introduced in Section 2.1, we will introduce the ideas of *leaves* and *trigram paths*.

A leaf is an index of a particular target-language word in a particular phrase. Each phrase $(s, t, e)$ implies $M$ leaves, where $M$ is the number of words in the target-language string $e$. $V_L = \{1, 2, \ldots, |V_L|\}$ is the set of all leaves. We use $\mathcal{P}$ to denote the set of all phrases.

Now, we define *trigram path*, which will be useful in incorporating the language model score. A trigram path $q$ is a tuple $(v_1, q_1, v_2, q_2, v_3)$ where

1. $v_1, v_2, v_3 \in V_L$

2. $q_1$ is the *path* between leaves $v_1$ and $v_2$.

3. $q_2$ is the *path* between leaves $v_2$ and $v_3$.

4. Each path can take the value NULL, or can specify a transition $(j, k)$. NULL is used if the two words being linked are in the same phrase. $(j, k)$ is used if the first leaf is at the end of a phrase ending in $j$, and the second leaf is at the start of a phrase starting at position $k$. Note that the value for $q_1$ is a deterministic function of $(v_1, v_2)$, and the value for $q_2$ is a deterministic function of $(v_2, v_3)$.

We use $v1(q)$, $q1(q)$, $v2(q)$, $q2(q)$, and $v3(q)$ to refer to the components $(v1, q1, v2, q2, v3)$ of a trigram path $q$. Figure 3-1 illustrates the idea of leaves, trigram paths, and paths between leaves.

We introduce the following variables:

- $y_v$ for all leaves $v \in V_L$. $y_v = 1$ if and only if the leaf $v$ is used in the derivation, $y_v = 0$ otherwise.

- $y_p$ for all phrases $p \in \mathcal{P}$. $y_p = 1$ if and only if the phrase $p$ is used in the derivation, $y_p = 0$ otherwise.

- $y_{j,k}$ for all $1 \leq j < k \leq N$. $y_{j,k} = 1$ if and only if there is a *transition* from $j$ to $k$: that is, a phrase ending at word $j$ in the source-language sentence is immediately followed by a phrase starting at word $k$.

- $y_q$ for each possible trigram path. $y_q = 1$ if and only if the trigram path $q$ is used to score the derivation.

Now the scoring function of a derivation (2.1) can be rewritten as

$$f(y) = \theta \cdot y = \sum_v \theta_v y_v + \sum_p \theta_p y_p + \sum_{j,k} \theta_{j,k} y_{j,k} + \sum_q \theta_q y_q.$$

The weight $\theta_v$ is set to 0; the weight $\theta_p$ specifies the phrase translation score $g(p)$; the weight $\theta_{j,k}$ specifies the distortion cost $\eta \times \delta(j, k)$; the weight $\theta_q$ is the language model score $h(v_3(q)|v_1(q)v_2(q))$.

The decoding problem is to find the highest scoring derivation within the set of valid derivations $\mathcal{Y}$:

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

The set $\mathcal{Y}$ will be defined later.

The constraints we would like to have are:

- **C0**: The $y_v$ and $y_p$ variables form a derivation that satisfies the distortion limit for all pairs of consecutive phrases.
- **C1**: for all $i = 1 \ldots N$, $y(i) = 1$
- **C2**: for all $v \in V_L$, $y_v = \sum_{p:v \in p} y_p$
- **C3**: for all $v \in V_L$, $y_v = \sum_{q:v_3(q)=v} y_q$
- **C4**: for all $v \in V_L$, $y_v = \sum_{q:v_2(q)=v} y_q$
- **C5**: for all $v \in V_L$, $y_v = \sum_{q:v_1(q)=v} y_q$
- **C6**: for all $(j,k)$, $y_{(j,k)} = \sum_{q:q_1(q)=(j,k)} y_q$
- **C7**: for all $(j,k)$, $y_{(j,k)} = \sum_{q:q_2(q)=(j,k)} y_q$

**C1** says that each word should be translated exactly once. **C0** and **C1** together require that $y_v$ and $y_p$ variables specify a *valid derivation* as defined in Section 2.1. **C2** states that the $y_v$ and $y_p$ variables are consistent. The number of times that a leaf is used is equal to the number of times that the phrase it belongs to is used. **C3**-**C5** indicates the consistency between the leaf and the trigram path. **C3** states that each leaf has exactly one incoming trigram path. **C4** states that each leaf is the middle of exactly one trigram path. **C5** states that each leaf is the beginning of exactly one trigram path. **C6** and **C7** enforce the consistency between the transition and the trigram path.

Define $\mathcal{Y}$ to be the set of all valid derivations, i.e., valid settings for the $y_v$, $y_p$ and $y_{j,k}$ variables. For a derivation to be valid, the $y_v$ and $y_p$ variables must be consistent; and the $y_{j,k}$ variables have to specify a valid ordering of the phrases such that $y_p = 1$.

$$\mathcal{Y} = \{y : y \text{ satisfies constraints } \mathbf{C0} - \mathbf{C7}\}$$

We define a new set:

$$\hat{\mathcal{Y}} = \{y : y \text{ satisfies constraints } \mathbf{C0} - \mathbf{C3}\}$$

In this set, we have omitted constraints **C4–C7**. These constraints will be introduced again using Lagrange multipliers. The problem of finding the highest scoring derivation within the set $\hat{\mathcal{Y}}$ can be solved efficiently by a decoding algorithm based on Lagrangian relaxation and dynamic programming, similar to the one described in Chapter 2.2.

The problem can be rewritten as:

$$\begin{aligned} \arg\max_{y \in \hat{\mathcal{Y}}} \quad & f(y) \\ \text{such that} \quad & \text{constraints } \mathbf{C4\text{-}C7} \text{ are satisfied} \end{aligned}$$

We introduce Lagrange multipliers $\lambda_v, \gamma_v, u_{(j,k)}, v_{(j,k)}$ for the constraints.

The Lagrangian is

$$\begin{aligned} L(y, \lambda, \gamma, u, v) = \theta \cdot y \\ & + \sum_v \lambda_v \left( y_v - \sum_{q:v_1(q)=v} y_q \right) \\ & + \sum_v \gamma_v \left( y_v - \sum_{q:v_2(q)=v} y_q \right) \\ & + \sum_{(j,k)} u_{(j,k)} \left( y_{(j,k)} - \sum_{q:q_1(q)=(j,k)} y_q \right) \\ & + \sum_{(j,k)} v_{(j,k)} \left( y_{(j,k)} - \sum_{q:q_2(q)=(j,k)} y_q \right) \end{aligned}$$

**Initialization**: set $\lambda^0 = 0$, $\gamma^0 = 0$, $u^0 = 0$, $v^0 = 0$.

**Algorithm**: For $t = 1 \ldots T$:

$y^t = \arg\max_{y \in \hat{\mathcal{Y}}} L(y, \lambda^{t-1}, \gamma^{t-1}, u^{t-1}, v^{t-1})$

**If** $y^t$ satisfies constraints **C4–C7**, return $y^t$

**Else**

$\forall v \in V_L, \lambda_v^t = \lambda_v^{t-1} - \alpha^t \left( y_v^t - \sum_{q:v1(q)=v} y_q^t \right)$

$\forall v \in V_L, \gamma_v^t = \gamma_v^{t-1} - \alpha^t \left( y_v^t - \sum_{q:v2(q)=v} y_q^t \right)$

$\forall (j,k), u_{(j,k)}^t = u_{(j,k)}^{t-1} - \alpha^t \left( y_{(j,k)}^t - \sum_{q:q1(q)=(j,k)} y_q^t \right)$

$\forall (j,k), v_{(j,k)}^t = v_{(j,k)}^{t-1} - \alpha^t \left( y_{(j,k)}^t - \sum_{q:q2(q)=(j,k)} y_q^t \right)$

Figure 3-2: The decoding algorithm. $\alpha^t > 0$ is the step size at the $t$'th iteration.

It can be rewritten as

$$\begin{aligned} L(y, \lambda, \gamma, u, v) &= \beta \cdot y \\ &= \sum\nolimits_v (\theta_v + \lambda_v + \gamma_v)\, y_v \\ &\quad + \sum\nolimits_p \theta_p y_p \\ &\quad + \sum\nolimits_{(j,k)} \left(\theta_{(j,k)} + u_{(j,k)} + v_{(j,k)}\right) y_{(j,k)} \\ &\quad + \sum\nolimits_q \left(\theta_q - \lambda_{v_1(q)} - \gamma_{v_2(q)} - u_{q_1(q)} - v_{q_2(q)}\right) y_q \end{aligned}$$

where

$$\begin{aligned} \beta_v &= \theta_v + \lambda_v + \gamma_v \\ \beta_p &= \theta_p \\ \beta_{(j,k)} &= \theta_{(j,k)} + u_{(j,k)} + v_{(j,k)} \\ \beta_q &= \theta_q - \lambda_{v_1(q)} - \gamma_{v_2(q)} - u_{q_1(q)} - v_{q_2(q)} \end{aligned}$$

Here we use $\beta_v$, $\beta_p$, $\beta_{(j,k)}$, and $\beta_q$ to denote the weights that incorporate the Lagrange multipliers.

1. For each $v \in V_L$, find $\rho_v^* = \arg\max_{q:v3(q)=v} \beta_q$, and $\delta_v^* = \beta_{\rho_v^*}$
2. Find $y_v$, and $y_{(j,k)}$ that forms a valid derivation, and that maximize

   $f'(y) = \sum_v (\beta_v + \delta_v^*) y_v + \sum_p \beta_p y_p + \sum_{(j,k)} \beta_{(j,k)} y_{(j,k)}$,

   which can be done using an algorithm very similar to the decoding algorithm described in Figure 2-3, based on a slightly different dynamic program.
3. Set $y_q = 1$ if and only if $y_{v3(q)} = 1$ and $q = \rho_v^*$

Figure 3-3: The procedure used to compute $\arg\max_{y \in \hat{\mathcal{Y}}} L(y, \lambda, \gamma, u, v) = \arg\max_{y \in \hat{\mathcal{Y}}} \beta \cdot y$ in the algorithm in Figure 3-2.

The dual objective is

$$L(\lambda, \gamma, u, v) = \max_{y \in \hat{\mathcal{Y}}} L(y, \lambda, \gamma, u, v)$$

and the dual problem is to solve

$$\min_{\lambda, \gamma, u, v} L(\lambda, \gamma, u, v).$$

Figure 3-2 shows a subgradient algorithm that solves the dual problem. At each iteration, we need to compute $\arg\max_{y \in \hat{\mathcal{Y}}} L(y, \lambda, \gamma, u, v) = \arg\max_{y \in \hat{\mathcal{Y}}} \beta \cdot y$. This can be done efficiently by the steps described in Figure 3-3.

The first step is to find the highest scoring incoming trigram path for each leaf $v$. The score consists of the language model score and the Lagrangian multipliers associated with each leaf and path of the trigram path. The second step can be viewed as to compute the highest scoring derivation within the set $\hat{\mathcal{Y}}$ without considering the language model score. We will describe the method in detail in Section 3.1.1. We will use "inner subgradient" to refer the method. The third step is to set $y_q = 1$ for those best incoming trigram paths for the leaves $v$ used in the derivation $y^t$.

Thus, in the algorithm, the second step will return a derivation $y^t$, which gives us the value of the variables $y_v$ and $y_{(j,k)}$. Then we will set $y_q$ to be 1 if $y_{v3(q)} = 1$ and $q = \rho_v^*$. Note that the language model score is calculated according to $y_q$. Also notice that, for each

leaf $v$ in the derivaiton, the previous word given by the derivation $y^t$ does not necessarily match the previous leaf given by the best incoming trigram path $\delta_v^*$.

The language model score is incorporated into the second steps through $\delta_v^*$. It is calculated according to the best incoming trigram path for each leaf. The Lagrange multipliers $\lambda, \gamma, u$ and $v$ are used to encourage the agreement between the two steps. Thus, in the algorithm in Figure 3-2, they are updated to encourage agreement. If the incoming trigram path for each leaf $v$ agrees with the what precedes each leaf $v$ in the derivations found in the second step, the language model score carried from the first step is exactly the language model score of the derivation. Thus, we have found a derivation that maximizes $\beta \cdot y$: $\arg\max_{y \in \hat{\mathcal{Y}}} \beta \cdot y$.

### 3.1.1 The Inner Subgradient Algorithm for $\arg\max_{y \in \hat{\mathcal{Y}}} f'(y)$

We use a subgradient algorithm that is very close to the decoding algorithm in Figure 2-3. In the step computing $y^t = \arg\max_{y \in \mathcal{Y}'} L(u^{t-1}, y)$, we replace $\mathcal{Y}'$ by $\hat{\mathcal{Y}}$ defined in this section. It becomes:

$$y^t = \arg\max_{y \in \hat{\mathcal{Y}}} L(u^{t-1}, y)$$

Then a slightly different dynamic program is used to find the derivation within the set $\hat{\mathcal{Y}}$. We replace the original dynamic program states $(w_1, w_2, n, l, m, r)$ by $(n, l, m, r)$. The bigram $(w_1, w_2)$ is omitted since we do not need to keep track of the trigram language model score in the dynamic program. Instead, for each edge between two nodes, we pick the highest scoring phrase for that edge at the beginning of the algorithm. Let $\mathcal{P}_{(s,t)}$ be the set of all phrases that start at $s$ and end at $t$. The phrase $\hat{p}$ we pick will be

$$\hat{p} = \arg\max_{p \in \mathcal{P}_{(s,t)}} \quad g(p) + \sum_{v \in p} (\beta_v + \delta_v^*).$$

The number of states becomes much less and the dynamic programming can be performed more efficiently. We use the Lagrangian relaxation to encourage a valid derivation, where each word is translated exactly once. We will call this step the inner subgradient method.

Similar to the dynamic program described in Section 2.2.1, the dynamic program can

be viewed as a shortest-path problem in a directed graph, with nodes in the graph corresponding to states $(n, l, m, r)$. For each state, we consider phrases that satisfy the distortion limit and do not overlap with the span $(l, m)$. For any such phrase, we create transition of the form

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

where

1. $n' = n + t - s + 1$
2. $(l', m') = \begin{cases} (l,\ t\ ) & \text{if } s = m + 1 \\ (s, m) & \text{if } t = l - 1 \\ (s,\ t\ ) & \text{otherwise} \end{cases}$
3. $r' = t$

The score of the transition is given by a sum of a updated translation score and the distortion cost $\eta \times \delta(r, s)$.

$$\hat{g}(\hat{p}) + \eta \times \delta(r, s)$$

The updated translation score $\hat{g}(\hat{p})$ includes the translation score $g(\hat{p})$, and the language model score and the language multiplier weights, both carried over by $(\beta_v + \delta_v^*)$ for each leaf $v \in p$, and the Lagrange multipliers $u(i)$ associated with the phrase $\hat{g}(p)$.

$$\hat{g}(\hat{p}) = g(\hat{p}) + \sum_{v \in \hat{p}} (\beta_v + \delta_v^*) + \sum_{i=s}^{t} u(i).$$

### 3.1.2 A Different View

In addition to the algorithm in Figure 3-2, we present a slightly different algorithm in Figure 3-4. First, we introduce a new constraint:

- **C1(a)**: for all $\sum_i^N y(i) = N$.

Then, we define another set

$$\tilde{\mathcal{Y}} = \{y : y \text{ satisfies constraints } \textbf{C0}, \textbf{C2}, \textbf{C3} \text{ and } \textbf{C1(a)}\}$$

Compared to the set $\hat{\mathcal{Y}}$, the set $\tilde{\mathcal{Y}}$ dropped constraint **C1**, which requires that each word to be translated exactly once. Instead, it enforces a constraint **C1(a)** that only requires the sum of the total number of words translated to be $N$, the sentence length. Note that $\mathcal{Y} \subset \hat{\mathcal{Y}} \subset \tilde{\mathcal{Y}}$. Also, note that the constraint **C1(a)** is enforced by the dynamic program described in Section 3.1.1

In the algorithm depicted in Figure 3-4, we would like to find a derivation within the set $\tilde{\mathcal{Y}}$ that maximizes the Lagrangian $L(y, \lambda, \gamma, u, v)$. This step can be done using the dynamic programming method directly, without using the inner subgradient method. The dynamic program is exactly the same as the one described above. The other difference in this algorithm is that we use Lagrangian multipliers, $\zeta_i$ for each word $i$, which are used to encourage all words to be translated exactly once.

The algorithm in Figure 3-2 can be viewed as a variant of the algorithm in Figure 3-4. In Figure 3-4, we updated all Lagrangian multipliers at once, while in Figure 3-2, we updated two sets of Lagrange multipliers alternatively. The two sets are $\{\lambda, \gamma, u, v\}$ and $\{\zeta_i : i = \ldots N\}$. The Lagrange multipliers $\zeta_i$ have the same function as the Lagrange multipliers $u(i)$ in Figure 2-3. The Lagrangian

$$L(y, \lambda^{t-1}, \gamma^{t-1}, u^{t-1}, v^{t-1}, \zeta^{t-1}) = \beta \cdot y + \sum_{i}^{N} \zeta_i (y(i) - 1)$$

In Section 3.3, we will present results on a method that is very similar to the one in Figure 3-2, but we set a hard limit on the number of iterations for the inner subgradient method computing $\arg\max_{y \in \hat{\mathcal{Y}}} \beta \cdot y$. This method can be viewed as a variant of Figure 3-4.

## 3.2 Tightening the Relaxation

Sometimes the underlying relaxation is not tight enough and the algorithm will not converge to an integral solution of the LP relaxation defined by the set $\hat{\mathcal{Y}}$. In this section, we will describe a method that incrementally adds hard constraints to the set $\hat{\mathcal{Y}}$ to tighten the relaxation, until the algorithm converges and returns the optimal solution. The algorithm is very similar to the one described in [20].

**Initialization**: set $\lambda^0 = 0$, $\gamma^0 = 0$, $u^0 = 0$, $v^0 = 0$.

**Algorithm**: For $t = 1 \ldots T$:

$y^t = \arg\max_{y \in \tilde{\mathcal{Y}}} L(y, \lambda^{t-1}, \gamma^{t-1}, u^{t-1}, v^{t-1}, \zeta^{t-1})$

**If** $y^t$ satisfies constraints **C1** and **C4–C7**, return $y^t$

**Else**

$\forall v \in V_L, \lambda_v^t = \lambda_v^{t-1} - \alpha^t \left(y_v^t - \sum_{q:v1(q)=v} y_q^t\right)$

$\forall v \in V_L, \gamma_v^t = \gamma_v^{t-1} - \alpha^t \left(y_v^t - \sum_{q:v2(q)=v} y_q^t\right)$

$\forall (j,k), u_{(j,k)}^t = u_{(j,k)}^{t-1} - \alpha^t \left(y_{(j,k)}^t - \sum_{q:q1(q)=(j,k)} y_q^t\right)$

$\forall (j,k), v_{(j,k)}^t = v_{(j,k)}^{t-1} - \alpha^t \left(y_{(j,k)}^t - \sum_{q:q2(q)=(j,k)} y_q^t\right)$

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

Figure 3-4: The decoding algorithm. $\alpha^t > 0$ is the step size at the $t$'th iteration.

Note that in the Lagrangian relaxation method described in the previous section, we would like to enforce constraints **C4-C5**, which requires that each leaf is the beginning of exactly one trigram path. At each iteration, given a leaf, we are encouraging the agreement of first and second leaf of the best incoming trigram path, and the second previous and first previous leaves given by the derivation output by the dynamic programming algorithm. To state formally, let $v_{-1}(v, y)$ be the leaf preceding $v$ in the trigram path $q$ with $y_q = 1$ and $v_3(q) = v$, and $v_{-2}(v, y)$ be the leaf preceding $v_{-1}(v, y)$, which is the beginning of the trigram path $q$ that ends in $v$. Then define $v'_{-1}$ and $v'_{-2}$ to be the previous two leaves preceding $v$ given by the derivation $y$ output by the dynamic programming algorithm. A consistent solution will have

- $v_{-1}(v, y) = v'_{-1}(v, y)$
- $v_{-2}(v, y) = v'_{-2}(v, y)$

for all leaves $v$ in the translation $y$, with $y_v = 1$. Enforcing all these constraints will result in the dynamic programming algorithm described in Section 2.2.1.

Here we enforce a weaker set of constraints. We assign each leaf to a partition and require that $v_{-1}(v, y)$ and $v'_{-1}(v, y)$ should be in the same partition, and so are $v_{-2}(v, y)$ and $v'_{-2}(v, y)$. Let $\pi$ be a function that partitions all the leaves into $r$ partitions. $\pi : V_L \rightarrow$

$\{1, 2, \ldots, r\}$. Then we will enforce the constraints that

- $\pi(v_{-1}(v, y)) = \pi(v'_{-1}(v, y))$
- $\pi(v_{-2}(v, y)) = \pi(v'_{-2}(v, y))$

for all leaves $v$ with $y_v = 1$. Let $\hat{\mathcal{Y}}'$ be the new set with these constraints added. Now we would like to find

$$\arg\max_{y \in \hat{\mathcal{Y}}'} \beta \cdot y.$$

We need to modify the steps described in Section 3.1.

1. For each $v \in V_L$, find $\rho^*_{v|\pi_1\pi_2} = \arg\max_{q:v_3(q)=v, v_2(q)\in\pi_2, v_1(q)\in\pi_1} \beta_v$ , and $\delta^*_{v|\pi_1\pi_2} = \beta_{\rho^*_{v|\pi_1\pi_2}}$.

2. Use the dynamic program with states $(\pi_1, \pi_2, n, l, m, r)$ to find the highest scoring derivation that satisfies the hard constraints.

The procedure used to decide a partition $\pi$ has two steps. First, when we observe that the dual value $L$ is not decreasing fast enough, we will run for 15 more iterations and add hard constraints between pairs of leaves that are violating the consistency constraints above. They are pairs $a = v_{-1}(v, y)/b = v'_{-1}(v, y)$ or $a = v_{-2}(v, y)/b = v'_{-2}(v, y)$ such that $a \neq b$. The hard constraints require that $a$ and $b$ are not in the same partition. That is, $\pi(a) \neq \pi(b)$. Thus, in the next iteration, they will not be selected as the previous word and the second leaf on the best incoming trigram path for a certain word at the same time. The second part is a graph coloring algorithm to find a partition in which $a$ and $b$ are in different partitions. In the graph, each node represents a leaf, and an edge is created between node $a$ and $b$ for all pairs of leaves $a$ and $b$ that violates the constraints. A graph coloring algorithm ensures that adjacent nodes will not have the same color, which makes sure that $a$ and $b$ will be in different partitions. With the new projection function, we continue the Lagrangian relaxation algorithm with the new constraints added.

## 3.3 Experiments

We report experimental results for the Lagrangian relaxation method described in this chapter. The same as the experiments in Section 2.6, we test on translations from German to English in the Europarl dataset. We will focus on the comparison between the two Lagrangian relaxation method described in this chapter and Chapter 2.2.

### 3.3.1 Complexity of the Dynamic Program

The motivation of this method is that the dynamic program would be much more efficient without keeping track of the language model score. In this section, we report the complexity of the new dynamic program (DP) compared with the dynamic program described in Section 2.2.1 (DPLM).

The run time of DP is in average $3\%$ of the run time of DPLM. As for the number of states, on average, the number of states of DP is $2.5\%$ of that of DPLM. We can see that the English bigram adds a lot of complexity to the dynamic program. The dynamic programming method becomes much more efficient when removing the English bigram from the state.

### 3.3.2 Time and Number of Iterations

In this section, we present four different sets of results on the algorithm in Figure 3-2. There are two features that we would like to vary.

First is the limit of the number of iterations of the inner subgradient method. We will use HARD to refer to a limit of 300 iterations, which is considered to solve the inner subgradient till convergence in most cases. Then we use LOOSE to refer to a limit of 25 iterations, which is usually less than the number of iterations required to achieve convergence. The idea is based on the observation that the inner subgradient often takes a huge amount of iterations, which prolong the run time. For the LOOSE case, we carry over the Language multipliers $\zeta_i$ for $i = 1 \ldots N$ from iteration to iteration, while for the HARD case, the Language multipliers will be reinitialized.

The second feature is regarding how to design the projection function $\pi$ that maps leaves to partitions when tightening the relaxation. One idea, which we will use SUB to refer to, is that each time the projection should make sure that the new set $\hat{\mathcal{Y}}'$ is a proper subset of the previous set. On the other hand, NON, will be used to refer to a method that the new set is not necessary a proper subset of the previous set. The idea is that if the tightening shrinks the set of derivation each time, the dual objective will be ensured to decrease after the tightening. However, the first projection function, which is obtained by a graph coloring algorithm, might add constraints on pairs of leaves that we do not require a constraint. A subsequent graph coloring procedure will only make sure that those pairs that we require constraints to be in different partitions. Requiring a proper subset will therefore adding constraints between the pairs that are in different partitions from the previous projections to make sure that they are still in different partitions by the following projection functions. This will explode the number of partitions we need to enforce all the hard constraints, which might cause a memory problem when we store the best incoming trigram path for the possible bigram combinations.

In summary, we have the following

- HARD: a limit of 300 iterations
- LOOSE: a limit of 25 iterations
- SUB: proper subset when tightening
- NON: not requiring a proper subset

The first set of experiments are HARD and SUB. The results will be presented in Table 3.1, Table 3.2, and Table 3.3.

The second set of experiments are HARD and NON. The results will be presented in Table 3.4, Table 3.5, and Table 3.6.

The third set of experiments are LOOSE and SUB. The results will be presented in Table 3.7, Table 3.8, and Table 3.9.

The fourth set of experiments are LOOSE and NON. The results will be presented in Table 3.10, Table 3.11, and Table 3.12.

| # iter. | 1-10 words | 11-20 words | All sentences | |
|---|---|---|---|---|
| 0-30 | 44 (23.8 %) | 9 ( 1.6 %) | 53 ( 7.1 %) | 7.1 % |
| 31-60 | 101 (54.6 %) | 155 (27.8 %) | 256 (34.5 %) | 41.6 % |
| 61-120 | 40 (21.6 %) | 367 (65.8 %) | 407 (54.8 %) | 96.4 % |
| 121-250 | 0 ( 0.0 %) | 26 ( 4.7 %) | 26 ( 3.5 %) | 99.9 % |
| x | 0 ( 0.0 %) | 1 ( 0.2 %) | 1 ( 0.1 %) | 100.0 % |

Table 3.1: Table showing the number of iterations taken for the algorithm to converge for the method HARD-SUB. We use a limit of 300 iterations and we ensure that with the new projection, the new set is a proper subset of the set in the previous iteration. x indicates sentences that fail to converge due to memory problem. All sentences refer to all sentences with less than 20 words.

| # expansions | 1-10 words | 11-20 words | All sentences | |
|---|---|---|---|---|
| 0 | 95 (51.4 %) | 99 (17.7 %) | 194 (26.1 %) | 26.1 % |
| 1 | 85 (45.9 %) | 368 (65.9 %) | 453 (61.0 %) | 87.1 % |
| 2 | 5 ( 2.7 %) | 85 (15.2 %) | 90 (12.1 %) | 99.2 % |
| 3 | 0 ( 0.0 %) | 4 ( 0.7 %) | 4 ( 0.5 %) | 99.7 % |
| 4 | 0 ( 0.0 %) | 1 ( 0.2 %) | 1 ( 0.1 %) | 99.9 % |
| x | 0 ( 0.0 %) | 1 ( 0.2 %) | 1 ( 0.1 %) | 100.0 % |

Table 3.2: Table showing the number of times that we expand the number of partitions that the leaves are assigned to during the tightening method. This is for the HARD-SUB method. x indicates the sentences that fail to due to memory problem. All sentences refer to all sentences with less than 20 words.

The LOOSE-SUB setting has the best performance among the above four settings of experiments. The average time for decoding sentences with 1 to 20 words is 41 seconds. However, it is less stable and less efficient on average compared with the method in Chapter 2. Using the HARD setting, the inner subgradient method might take many iterations, which increase the time required at each iteration. The LOOSE setting, although not solving the inner problem to convergence, does not affect the total number of iterations to convergence much, while saves time at each iteration. The SUB setting requires much less iterations than the NON setting, but encounters the memory problem that we described.

## 3.4 Conclusion

We consider this alternative Lagrangian method for decoding phrase-based translation models due to the observation that the dynamic programming algorithm would be much more efficient without considering the language model. However, in our experiments, we find that the major bottle neck is the large number of iterations of the inner subgradient method.

| # cons. | 1-10 words | 11-20 words | All sentences |
|---|---|---|---|
| 0 | 0.7 ( 95) | 13.6 ( 99) | 7.3 ( 194) |
| 1 | 2.7 ( 85) | 87.4 ( 368) | 71.5 ( 453) |
| 2 | 25.9 ( 5) | 901.7 ( 85) | 853.0 ( 90) |
| 3 | 0.0 ( 0) | 358.6 ( 4) | 358.6 ( 4) |
| 4 | 0.0 ( 0) | 624.1 ( 1) | 624.1 ( 1) |
| mean | 2.3 ( 185) | 201.5 ( 557) | 151.8 ( 742) |
| median | 0.9 | 32.6 | 16.5 |

Table 3.3: The average time (in seconds) for decoding with the HARD-SUB method. All sentences refer to all sentences with less than 20 words.

| # iter. | 1-10 words | 11-20 words | All sentences | |
|---|---|---|---|---|
| 0-30 | 44 (23.8 %) | 9 ( 1.6 %) | 53 ( 7.1 %) | 7.1 % |
| 31-60 | 100 (54.1 %) | 152 (27.2 %) | 252 (33.9 %) | 41.0 % |
| 61-120 | 39 (21.1 %) | 332 (59.5 %) | 371 (49.9 %) | 91.0 % |
| 121-250 | 2 ( 1.1 %) | 59 (10.6 %) | 61 ( 8.2 %) | 99.2 % |
| 251-999 | 0 ( 0.0 %) | 6 ( 1.1 %) | 6 ( 0.8 %) | 100.0 % |
| x | 0 ( 0.0 %) | 0 ( 0.0 %) | 0 ( 0.0 %) | 100.0 % |

Table 3.4: Table showing the number of iterations taken for the algorithm to converge for the method HARD-NON. x indicates sentences that fail to converge due to memory problem. All sentences refer to all sentences with less than 20 words.

Without the language model, the number of iteration required to converge to a valid derivation increases a lot. The reason might be that the bigram used to calculate the trigram language model in Section 2.2.1 might help to eliminate some ill-formed derivation. Looking at the derivation at each iteration more closely, we find many sentences repeat several phrases. For example, at one iteration, the derivation is

$$(11, 11, \text{task}), (13, 13, .), (11, 11, \text{task}), (13, 13, .), (11, 11, \text{task}) \ldots,$$

and at the next iteration, it becomes

$$(4, 4, \text{that}), (7, 7, \text{presidency}), (4, 4, \text{that}), (7, 7, \text{presidency}), (4, 4, \text{that}) \ldots.$$

This is less likely to happen if we are looking at the trigram language model score at the same time, since the English sentence "task . task . task" would likely to receive a lower language model score than a sentence that take different part from the source language sentence.

| # expansions | 1-10 words | 11-20 words | All sentences | |
|---|---|---|---|---|
| 0 | 95 (51.4 %) | 99 (17.7 %) | 194 (26.1 %) | 26.1 % |
| 1 | 85 (45.9 %) | 368 (65.9 %) | 453 (61.0 %) | 87.1 % |
| 2 | 3 ( 1.6 %) | 26 ( 4.7 %) | 29 ( 3.9 %) | 91.0 % |
| 3 | 1 ( 0.5 %) | 22 ( 3.9 %) | 23 ( 3.1 %) | 94.1 % |
| 4 | 1 ( 0.5 %) | 15 ( 2.7 %) | 16 ( 2.2 %) | 96.2 % |
| $> 5$ | 0 ( 0.0 %) | 28 ( 5.0 %) | 28 ( 3.8 %) | 100.0 % |
| x | 0 ( 0.0 %) | 0 ( 0.0 %) | 0 ( 0.0 %) | 100.1 % |

Table 3.5: Table showing the number of times that we expand the number of partitions that the leaves are assigned to during the tightening method. This is for the HARD-NON method. x indicates the sentences that fail to due to memory problem. All sentences refer to all sentences with less than 20 words.

| # cons. | 1-10 words | 11-20 words | All sentences |
|---|---|---|---|
| 0 | 0.7 ( 95) | 13.5 ( 99) | 7.2 ( 194) |
| 1 | 2.6 ( 85) | 86.1 ( 368) | 70.5 ( 453) |
| 2 | 15.5 ( 3) | 594.0 ( 26) | 534.1 ( 29) |
| 3 | 18.6 ( 1) | 1,994.0 ( 22) | 1,908.2 ( 23) |
| 4 | 131.0 ( 1) | 903.6 ( 15) | 855.3 ( 16) |
| $> 5$ | 0.0 ( 0) | 4,919.2 ( 28) | 4,919.2 ( 28) |
| mean | 2.6 ( 185) | 436.6 ( 558) | 328.6 ( 743) |
| median | 0.8 | 32.2 | 16.6 |

Table 3.6: The average time (in seconds) for decoding with the HARD-NON method. All sentences refer to all sentences with less than 20 words.

| # iter. | 1-10 words | 11-20 words | All sentences | |
|---|---|---|---|---|
| 0-30 | 43 (23.2 %) | 6 ( 1.1 %) | 49 ( 6.6 %) | 6.6 % |
| 31-60 | 104 (56.2 %) | 117 (21.0 %) | 221 (29.7 %) | 36.3 % |
| 61-120 | 38 (20.5 %) | 343 (61.5 %) | 381 (51.3 %) | 87.6 % |
| 121-250 | 0 ( 0.0 %) | 80 (14.3 %) | 80 (10.8 %) | 98.4 % |
| x | 0 ( 0.0 %) | 12 ( 2.2 %) | 12 ( 1.6 %) | 100.0 % |

Table 3.7: Table showing the number of iterations taken for the algorithm to converge for the method LOOSE-SUB. x indicates sentences that fail to converge due to memory problem. All sentences refer to all sentences with less than 20 words.

| # expansions | 1-10 words | 11-20 words | All sentences | |
|---|---|---|---|---|
| 0 | 96 (51.9 %) | 93 (16.7 %) | 189 (25.4 %) | 25.4 % |
| 1 | 83 (44.9 %) | 325 (58.2 %) | 408 (54.9 %) | 80.3 % |
| 2 | 6 ( 3.2 %) | 112 (20.1 %) | 118 (15.9 %) | 96.2 % |
| 3 | 0 ( 0.0 %) | 14 ( 2.5 %) | 14 ( 1.9 %) | 98.1 % |
| 4 | 0 ( 0.0 %) | 2 ( 0.4 %) | 2 ( 0.3 %) | 98.4 % |
| $> 5$ | 0 ( 0.0 %) | 0 ( 0.0 %) | 0 ( 0.0 %) | 98.4 % |
| x | 0 ( 0.0 %) | 12 ( 2.2 %) | 12 ( 1.6 %) | 100.0 % |

Table 3.8: Table showing the number of times that we expand the number of partitions that the leaves are assigned to during the tightening method. This is for the LOOSE-SUB method. x indicates the sentences that fail to due to memory problem. All sentences refer to all sentences with less than 20 words.

| # cons. | 1-10 words | 11-20 words | All sentences |
|---|---|---|---|
| 0 | 0.6 ( 96) | 6.8 ( 93) | 3.6 ( 189) |
| 1 | 1.9 ( 83) | 23.6 ( 325) | 19.2 ( 408) |
| 2 | 11.4 ( 6) | 145.5 ( 112) | 138.7 ( 118) |
| 3 | 0.0 ( 0) | 326.6 ( 14) | 326.6 ( 14) |
| 4 | 0.0 ( 0) | 198.1 ( 2) | 198.1 ( 2) |
| mean | 1.5 ( 185) | 54.1 ( 546) | 40.8 ( 731) |
| median | 0.7 | 17.7 | 9.7 |

Table 3.9: The average time (in seconds) for decoding with the LOOSE-SUB method. All sentences refer to all sentences with less than 20 words.

| # iter. | 1-10 words | 11-20 words | All sentences | |
|---|---|---|---|---|
| 0-30 | 43 (23.2 %) | 6 ( 1.1 %) | 49 ( 6.6 %) | 6.6 % |
| 31-60 | 103 (55.7 %) | 116 (20.8 %) | 219 (29.5 %) | 36.1 % |
| 61-120 | 38 (20.5 %) | 304 (54.6 %) | 342 (46.1 %) | 82.2 % |
| 121-250 | 1 ( 0.5 %) | 108 (19.4 %) | 109 (14.7 %) | 96.9 % |
| 251-999 | 0 ( 0.0 %) | 23 ( 4.1 %) | 23 ( 3.1 %) | 100.0 % |
| x | 0 ( 0.0 %) | 0 ( 0.0 %) | 0 ( 0.0 %) | 100.0 % |

Table 3.10: Table showing the number of iterations taken for the algorithm to converge for the method LOOSE-NON. x indicates sentences that fail to converge due to memory problem. All sentences refer to all sentences with less than 20 words.

| # expansions | 1-10 words | 11-20 words | All sentences | |
|---|---|---|---|---|
| 0-0 | 96 (51.9 %) | 93 (16.7 %) | 189 (25.5 %) | 25.5 % |
| 1 | 83 (44.9 %) | 325 (58.3 %) | 408 (55.0 %) | 80.5 % |
| 2 | 3 ( 1.6 %) | 39 ( 7.0 %) | 42 ( 5.7 %) | 86.1 % |
| 3 | 1 ( 0.5 %) | 24 ( 4.3 %) | 25 ( 3.4 %) | 89.5 % |
| 4 | 1 ( 0.5 %) | 19 ( 3.4 %) | 20 ( 2.7 %) | 92.2 % |
| $> 5$ | 1 ( 0.5 %) | 58 (10.4 %) | 59 ( 8.0 %) | 100.1 % |
| x | 0 ( 0.0 %) | 0 ( 0.0 %) | 0 ( 0.0 %) | 100.1 % |

Table 3.11: Table showing the number of times that we expand the number of partitions that the leaves are assigned to during the tightening method. This is for the LOOSE-NON method. x indicates the sentences that fail to due to memory problem. All sentences refer to all sentences with less than 20 words.

| # cons. | 1-10 words | 11-20 words | All sentences |
|---|---|---|---|
| 0 | 0.5 ( 96) | 6.7 ( 93) | 3.5 ( 189) |
| 1 | 1.8 ( 83) | 23.4 ( 325) | 19.0 ( 408) |
| 2 | 4.4 ( 3) | 136.1 ( 39) | 126.7 ( 42) |
| 3 | 11.0 ( 1) | 201.7 ( 24) | 194.0 ( 25) |
| 4 | 32.0 ( 1) | 352.7 ( 19) | 336.6 ( 20) |
| $> 5$ | 76.8 ( 1) | 1,183.9 ( 58) | 1,165.1 ( 59) |
| mean | 1.8 ( 185) | 168.0 ( 558) | 126.6 ( 743) |
| median | 0.7 | 18.1 | 9.8 |

Table 3.12: The average time (in seconds) for decoding with the LOOSE-NON method. All sentences refer to all sentences with less than 20 words.

# Chapter 4

# Conclusion

In this thesis, we present two Lagrangian relaxation algorithms for exact decoding of phrase-based models. The first algorithm uses the idea that when relaxing the "exactly-once" constraints, the problem can be solved efficiently by dynamic programming. Then the constraints are introduced by Lagrangian relaxation. This method is efficient in recovering exact solutions under the phrase-based translation model. The second algorithm is based on the observation that the language model adds a lot of complexity to the dynamic program. We borrowed the idea in [20] to factor out the language model and use Lagrangian relaxation to encourage agreement between the subproblem that includes the language model and the subproblem that focuses on using dynamic programming to find the best derivation. Although the resulting dynamic program is much more efficient, the second method does not outperform the first method due to a larger number of iterations.

# Bibliography

[1] Graeme Blackwood, Adrià de Gispert, Jamie Brunning, and William Byrne. 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, 2009.

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

[3] Yin-Wen Chang and Michael Collins. Exact decoding of phrase-baed translation models through Lagrangian relaxation. In *Proceedings of EMNLP*, 2011.

[4] Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 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, 2001.

[5] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of the MT Summit*, 2005.

[6] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 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, 2007.

[7] Philipp Koehn, Franz Josef Och, and Daniel Marcu. 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, 2003.

[8] Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. MRF optimization via dual decomposition: Message-passing revisited. In *Proceedings of the 11th International Conference on Computer Vision*, 2007.

[9] Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 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, October 2010.

[10] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Application*. Springer Verlag, 2008.

[11] Shankar Kumar and William Byrne. 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, 2005.

[12] Claude Lemaréchal. Lagrangian Relaxation. In *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [based on a Spring School]*, pages 112–156, 2001.

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

[14] Franz Josef Och. 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, 2003.

[15] Franz Josef Och, Christoph Tillmann, Hermann Ney, and Lehrstuhl Fiir Informatik. 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, 1999.

[16] Franz Josef Och, Nicola Ueffing, and Hermann Ney. 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, 2001.

[17] Kishore Papineni, Salim Roukos, Todd Ward, and Wei Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of ACL 2002*, 2002.

[18] Sebastian Riedel and James Clarke. 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, 2006.

[19] Sebastian Riedel and James Clarke. 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, 2009.

[20] Alexander M. Rush and Michael Collins. Exact decoding of syntactic translation models through Lagrangian relaxation. In *Proceedings of ACL*, 2011.

[21] Alexander M Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 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, October 2010.

[22] David A. Smith and Jason Eisner. Dependency parsing by belief propagation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 145–156, 2008.

[23] David Sontag, Talya Meltzer, Amir Globerson, Tommi Jaakkola, and Yair Weiss. Tightening LP relaxations for MAP using message passing. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 503–510, 2008.

[24] Christoph Tillmann. 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, 2006.

[25] Christoph Tillmann and Hermann Ney. Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational Linguistics*, 29:97–133, March 2003.

[26] Roy W. Tromble and Jason Eisner. 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, 2006.

[27] Martin Wainwright, Tommi Jaakkola, and Alan Willsky. MAP estimation via agreement on trees: Message-passing and linear programming. In *IEEE Transactions on Information Theory*, volume 51, pages 3697–3717, 2005.

[28] Mikhail Zaslavskiy, Marc Dymetman, and Nicola Cancedda. 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, 2009.