Exact Decoding of Phrase-Based Translation Models through Lagrangian Relaxation

Yin-Wen Chang
MIT CSAIL
Cambridge, MA 02139, USA
yinwen@csail.mit.edu

Michael Collins
Department of Computer Science,
Columbia University,
New York, NY 10027, USA
mcollins@cs.columbia.edu

Abstract

This paper describes an algorithm for exact decoding of phrase-based translation models, based on Lagrangian relaxation. The method recovers exact solutions, with certificates of optimality, on over 99% of test examples. The 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 method to MOSES (Koehn et al., 2007), and give precise estimates of the number and magnitude of search errors that MOSES makes.

1 Introduction

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

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

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

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

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

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

covered exact solutions for the type of phrase-based models used in MOSES.

2 Related Work

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

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

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

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

3 The Phrase-based Translation Model

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

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

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

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

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

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

a language model.[3]

The decoding problem is to find

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

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

1. Each word in the input is translated exactly once: that is, $y(i) = 1$ for $i = 1 \ldots N$.
2. For each pair of consecutive phrases p_k, p_{k+1} for $k = 1 \ldots L-1$, we have $\delta(t(p_k), s(p_{k+1})) \leq d$, where d is the *distortion limit*.

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

4 A Decoding Algorithm based on Lagrangian Relaxation

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

4.1 An Efficient Dynamic Program

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

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

can be found efficiently using dynamic programming. The set $\mathcal{Y}'$ omits some constraints—specifically, the constraints that each source-language word is translated once, i.e., that $y(i) = 1$ for $i = 1 \ldots N$—that are enforced for members of $\mathcal{Y}$. In the next section we describe how to reintroduce 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

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

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

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

where

- $(w'_1, w'_2) = \begin{cases} (e_{M-1}, e_M) & \text{if } M \geq 2 \\ (w_2, e_1) & \text{if } M = 1 \end{cases}$
- $n' = n + t - s + 1$
- $(l', m') = \begin{cases} (l,\ t\ ) & \text{if } s = m + 1 \\ (s, m) & \text{if } t = l - 1 \\ (s,\ t\ ) & \text{otherwise} \end{cases}$
- $r' = t$

The new target-language bigram (w'_1, w'_2) is the last two words of the partial translation after including phrase p. It comes from either the last two words of e, or, if e consists of a single word, the last word of the previous bigram, w_2, and the first and only word, e_1, in e. (l', m') is expanded from (l, m) if the spans (l, m) and (s, t) are adjacent. Otherwise, (l', m') will be the same as (s, t).

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

We also include start and end states in the directed graph. The start state is (<s>, <s>, 0, 0, 0, 0) where <s> is the start symbol in the language model. For each state (w_1, w_2, n, l, m, r), such that $n = N$, we create a transition to the end state. This transition takes the form

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

For this transition, we define the score as $score = h(\text{</s>}|w_1, w_2)$; 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 $p_1 p_2 \dots p_L$. We define $\mathcal{Y}'$ to be the full set of such sequences. We can use the Viterbi algorithm to solve $\arg\max_{y \in \mathcal{Y}'} f(y)$ by simply searching for the highest scoring path from the start state to the end state.

The set $\mathcal{Y}'$ clearly includes derivations that are ill-formed, in that they may include words that have been translated 0 times, or more than 1 time. The first line of Figure 2 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 $\mathcal{Y}$.

4.2 The Lagrangian Relaxation Algorithm

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

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

We use Lagrangian relaxation (Korte and Vygen, 2008) to deal with the $y(i) = 1$ constraints. We introduce Lagrange multipliers $u(i)$ for each such constraint. The Lagrange multipliers $u(i)$ can take any positive or negative value. The Lagrangian is

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

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

The dual objective is then

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

and the dual problem is to solve

$$\min_u L(u).$$

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

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

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

and $\gamma_u(i) = y_u(i) - 1$ for $i = 1 \ldots N$, then γ_u is a subgradient of $L(u)$ at u. A subgradient method is an iterative method for minimizing $L(u)$, which perfoms updates $u^t \leftarrow u^{t-1} - \alpha^t \gamma_{u^{t-1}}$ where $\alpha^t > 0$ is the step size for the t'th subgradient step.

Figure 1 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 $y^t(i) = 1$.

4.3 Properties

We now give some theorems stating formal properties of the Lagrangian relaxation algorithm. These results for Lagrangian relaxation are well known: for 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 α^t, the method converges to the minimum of $L(u)$. Hence we will successfully find the tightest possible upper bound defined by the dual $L(u)$.

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

Proof. See Korte and Vygen (2008). □

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: An example run of the algorithm in Figure 1. For each value of t we show the dual value $L(u^{t-1})$, the derivation y^t, and the number of times each word is translated, $y^t(i)$ for $i = 1 \dots N$. 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 $i = 1 \dots N$, and the translation is returned, with a guarantee that it is optimal.

Our final theorem states that if at any iteration the algorithm finds a solution y^t such that $y^t(i) = 1$ for $i = 1 \dots N$, 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* $i = 1 \dots N$, *then* $f(y_u) = f(y^*)$, *i.e.* y_u *is optimal.*

Proof. We have

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

The second equality is true because of the definition of y_u. The third equality follows because by assumption $y_u(i) = 1$ for $i = 1 \dots N$. 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 $f(y_u) \leq f(y^*)$. It follows that $f(y_u) = f(y^*)$. □

In some cases, however, the algorithm in Figure 1 may not return a solution y^t such that $y^t(i) = 1$ for all i. There could be two reasons for this. In the first case, we may not have 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 5 describes a method for tightening the underlying relaxation by introducing hard constraints (of the form $y(i) = 1$ for selected values of i). We will see that this method is highly effective in tightening the relaxation until the algorithm converges to an optimal solution.

4.4 An Example of the Algorithm

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

5 Tightening the Relaxation

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

$Optimize(\mathcal{C}, u)$
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 3: A decoding algorithm with incremental addition of constraints. The function $Optimize(\mathcal{C}, u)$ is a recursive function, which takes as input a set of constraints $\mathcal{C}$, and a vector of Lagrange multipliers, u. The initial call to the algorithm is with $\mathcal{C} = \emptyset$, and $u = 0$. $\alpha > 0$ is the step size. In our experiments, the step size decreases each time the dual value increases from one iteration to the next; see Appendix 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.[4]

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

If at any stage the algorithm finds a solution y^* such that $y^*(i) = 1$ for all i, then this is the solution to our original problem, $\arg\max_{y \in \mathcal{Y}} f(y)$. This follows because for any $\mathcal{C} \subseteq \{1 \ldots N\}$ we have $\mathcal{Y} \subseteq \mathcal{Y}'_{\mathcal{C}}$; hence the theorems in section 4.3 go through for $\mathcal{Y}'_{\mathcal{C}}$ in place of $\mathcal{Y}'$, with trivial modifications. Note also that the algorithm is guaranteed to eventually find the optimal solution, because eventually $\mathcal{C} = \{1 \ldots N\}$, and $\mathcal{Y} = \mathcal{Y}'_{\mathcal{C}}$.

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

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

The remaining question concerns the "dual still improving" condition; i.e., how to determine that the first phase of the algorithm should terminate. We do this by recording the first and second best dual values $L(u')$ and $L(u'')$ in the sequence of Lagrange multipliers $u^1, u^2, \ldots$ 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 ϵ is a parameter of the approach: in experiments we used $\epsilon = 0.002$.

See the supplementary material for this submission for an example run of the algorithm.

When $\mathcal{C} \neq \emptyset$, A* search can be used for decoding, with the dynamic program for $\mathcal{Y}'$ providing admissible estimates for the dynamic program for $\mathcal{Y}'_{\mathcal{C}}$. Experiments show that A* gives significant improvements in efficiency. The supplementary material contains a full description of the A* algorithm.

6 Experiments

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

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

Our method finds exact solutions on 1,818 out of 1,824 sentences (99.67%). (6 examples do not converge within 250 iterations.) Table 1 shows the number of iterations required for convergence, and Table 2 shows the number of constraints required for convergence, broken down by sentence length. In 1,436/1,818 (78.7%) sentences, the method converges without adding hard constraints to tighten the relaxation. For sentences with 1-10 words, the vast majority (183 out of 185 examples) converge with 0 constraints added. As sentences get longer, more constraints are often required. However most examples converge with 9 or fewer constraints.

Table 3 shows the average times for decoding, broken down by sentence length, and by the number of constraints that are added. As expected, decoding times increase as the length of sentences, and the number of constraints required, increase. The average run time across all sentences is 120.9 seconds. Table 3 also shows the run time of the method without the A* algorithm for decoding. The A* algorithm gives significant reductions in runtime.

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

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 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 4.1, and $\mathcal{Y}''$ indicates the dynamic program using states (w_1, w_2, n, r). The statistics for ILP for length 16-20 are based on 50 sentences.

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 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 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).

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

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

Results for MOSES-nogc Table 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.

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

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

Table 5: Table showing the number of examples where MOSES-nogc fails to give a translation, and the number/percentage of search errors for cases where it does give a translation.

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

Table 6: Table showing statistics for the difference between the translation score from MOSES, and from the optimal derivation, for those sentences where a search error is made. For MOSES-gc we include cases where the translation produced by our system is not reachable by MOSES-gc. The average score of the optimal derivations is -23.4.

Results for MOSES-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 6 shows statistics for the magnitude of the search errors that MOSES-gc and MOSES-nogc make.

BLEU Scores Finally, table 7 gives BLEU scores (Papineni et al., 2002) 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.

7 Conclusions

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

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 7: BLEU score comparisons. We consider only those sentences where both decoders produce a translation.

A Step Size

Similar to Koo et al. (2010), we set the step size at the t'th iteration to be $\alpha^t = 1/(1 + \lambda^t)$, where λ^t is the number of times that $L(u^{(t')}) > L(u^{(t'-1)})$ for all $t' \leq t$. Thus the step size decreases each time the dual value increases from one iteration to the next.

Acknowledgments Yin-Wen Chang and Michael Collins were supported under the GALE program of the Defense Advanced Research Projects Agency, Contract No. HR0011-06-C-0022. Michael Collins was also supported by NSF grant IIS-0915176.

References

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

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

Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *Proceed-*

ings of the 39th Annual Meeting on Association for Computational Linguistics, ACL '01, pages 228–235.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, NAACL '03, pages 48–54.

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. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180.

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

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

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA, October. Association for Computational Linguistics.

Bernhard Korte and Jens Vygen. 2008. *Combinatorial Optimization: Theory and Application*. Springer Verlag.

Shankar Kumar and William Byrne. 2005. 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

Christoph Tillmann. 2006. Efficient dynamic programming search algorithms for phrase-based SMT.

In *Proceedings of the Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, CHSLP '06, pages 9–16.

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

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

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