Project #3: Parsing
CS 7650 Natural Language Processing

Arvind Krishnaa Jagannathan
GT ID: 902891874

March 04, 2013

1 CFG Parsing for Twitter

1.1 Getting Started

Deliverable 1

The CFG I created has 9 non-terminals. They are:

1. **S**: Corresponds to a sentence (or a tweet in this case). A sentence can be constructed recursively. In the case of the simple CFG which gives only one parse per sentence, a sentence can be constructed by concatenating two sentences with a conjunction, or a conjunction followed by a sentence (in regular English this would be incorrect), or a re-tweet marker followed by a sentence. In addition, in order to remove ambiguities, I do not have the

   ```
   S -> NP VP
   ```

   production. Instead I have a production

   ```
   S -> PHR
   PHR -> NP | VP
   ```

   and there are productions leading from a NP (noun phrase) to VP (verb phrase).

2. **RT**: Corresponds to the re-tweet marker - a sequence of tags corresponding to the words RT @user :. Usually this marker can be found only at the start of a sentence.

3. **PHR**: Corresponds to a phrase. A phrase can be either a noun phrase (NP) or verb phrase (VP).

4. **NP**: A noun phrase. This could either begin with a proper noun ("^"), a common noun ("N"), a pronoun ("O"), a nominal ("S"), proper noun + verbal ("M"), a nominal + verbal ("L"). It could be constructed recursively either by combining a NP with a prepositional phrase (PP) or a verb phrase (VP), adjectival phrase (ADJP), adverbial phrase (ADVP) or a special character (Z).

5. **VP**: A verb phrase. This could be constructed by a verb ("V") followed by a particle ("T") or an existential there ("X"). The verb phrase can be recursively constructed by combining one of these terminals with itself, or with a noun phrase, a prepositional phrase, an adverbial phrase, adjectival phrase or special character.

6. **ADJP**: An adjectival phrase. It is a non-terminal preceeded by one or more adjectives ("A") and can also be constructed recursively from another ADJP.

7. **ADVP**: An adverbial phrase. It is a non-terminal preceeded by one or more adverbs ("R"). This can also be recursively constructed from another ADVp.

8. **PP**: A prepositional phrase. It is a non-terminal preceeded by one or more prepositions ("P") or followed by a noun phrase or a verb phrase.
9. **Z**: This non-terminal collects all the special characters/punctuations and numerals. One or more of these could compose Z.

The CFG is called *"jagannathan-task1.cfg"*. It is reproduced here,

```
S -> S "&" S | "&" S | RT S | PHR

RT -> "~" "@" "~"

PHR -> NP | VP

NP -> "^" | "N" | "O" | "S" | "M" | "Y" | "L" | Z
NP -> "^" VP | "N" VP | "O" VP | "S" VP | "M" VP | "Y" VP | "L" VP
NP -> "D" "N" "P"
NP -> "D" NP | "^" NP | "N" NP | "O" NP | "O" ADJP | Z NP | "S" NP | "L" NP | "M" NP
NP -> "Y" NP | "L" NP
NP -> "^" ADJP | "N" ADJP | "D" ADJP | "S" ADJP
NP -> "R" ADJP | "A" ADVP | "A" NP | "R" NP | "R" ADVP | "P" NP | "P" "V" NP

VP -> "V" | "V" "T"
VP -> "V" NP | "V" "T" NP
VP -> "V" ADVP | "V" ADJP
VP -> "V" NP | "V" VP | Z VP | "V" "T" VP
VP -> "A" VP | "R" VP | "X" VP

ADJP -> "A" | "A" ADJP

PP -> "P"
PP -> "P" NP | "P" VP

ADVP -> "R" | "R" ADVP

Z -> "," | "!" | "#" | "$" | "E" | "U" | "G" | "@"
```

I got the following measures for this grammar. Notably. I get a F-measure of 35.97%.

- **Recall**: 1.0
- **Parses-per-sentence**: 1.0
- **F-Measure**: 0.3597359735973597
- **Precision**: 0.2193158953722334

1.2 Grammar design

Deliverable 2

I have two grammars, *"jagannathan-task2.cfg"* and *"jagannathan-task2_1.cfg"* which give perfect recall for the development data. Their F-Measures are 40.74% and 41.52% respectively, while the number of parses per sentence are 2.137 and 1.348 respectively. I have summarized the two grammars in Table 1.

Deliverable 3

Even after I change the **max_len** parameter in the *evalParser()* method, my two grammars do not produce that many more parses per sentence. However, the grammars no longer produce perfect recall. The summary of two grammars is shown in Table 2.

Context Free Grammar	Recall	Parses per sentence	F-Measure	Precision
jagannathan-task2.cfg	100.00%	2.137	40.74%	25.58%
jagannathan-task2_1.cfg	100.00%	1.348	41.52%	26.20%

Table 1: Two grammars for Deliverable 2

Context Free Grammar	Recall	Parses per sentence	F-Measure	Precision
jagannathan-task2.cfg	81.70%	4.71875	42.80%	29.00%
jagannathan-task2_1.cfg	82.12%	2.507	45.35%	31.33%

Table 2: Comparison of two grammars when **max_len** = 20

Deliverable 4

In my grammar, I added the verbal tags as terminals which are at the head of a noun phrase. However, since these words (such as she'll) will usually be followed by a verb, so I have a production like

```
NP -> "L"
NP -> "L" VP
```

and similarly for other verbal tags.

1.3 Terminal Refinement

Deliverable 5

I made a few tag improvements based on the ideas in the paper on "Accurate Unlexicalized Parsing" by Klein and Manning [1]. However, the F-Measure increased from 40.74% (I used the first grammar *"jagannathan-task2.cfg"*) to 42.91% only. I present the summary of the new grammar in Table 3.

Context Free Grammar	Recall	Parses per sentence	F-Measure	Precision
jagannathan-task3.cfg	94.49%	2.067	42.91%	27.54%

Table 3: Summary of new Grammar (compare with Table 1)

These are the changes which I made using the **preprocess()** function (in addition to assigning a special tag '2' for the word 'to')

1. If the special character is "." and is at the end of a sentence, then assign it a special tag. I did this so that I could have a production of the form,

   ```
   S -> S "." S
   ```

2. Whenever the word "just" is tagged as an adverb, I have changed the tag to 'J'. This is as per [1]. Then I have created a rule,

   ```
   ADVP -> "J" ADVP
   NP -> "R" ADVP
   ```

   since the word "just" is mostly associated with noun phrases.

3. The prepositions "of" and "as" are assigned their own tags. This is so that I could create new rules for a prepositional phrase, with "of" being associated with noun phrases and "as" with verb phrases.

   ```
   PP -> "OF" NP | "AS" VP
   ```

I was unable to think of many new modifications, because most of my original grammar was designed with the tagset from the Twitter POS tagging paper [2] and my own knowledge of common tweet structures.

Deliverable 6

My best grammars are *"jagannathan1.cfg"* and *"jagannathan2.cfg"*, which are the **same as that for deliverable 2**. Please note that these grammars gave me perfect recall for the development data and F-measure of 40.74% and 41.52% respectively. It would be great if both of these grammars (the second one has subtle development data specific production rules) could be evaluated on the test data.

2 Unlabeled Dependency Parsing

Deliverable 7

When I run the *read_ data()* the number of features I get are **801**.

2.1 Training

After 10 iterations of training, the dependency parser gives an accuracy of 50.97%.

2.2 More features

Deliverable 8

I obtain the distance from the head and the modifier and store it as,

```
head_modifier_distance = m - h
```

As per the advice suggested, I set an upper threshold of 8 and a lower threshold of -5. That is,

```
if head_modifier_distance > 8:
   head_modifier_distance = 8
if head_modifier_distance < -5:
   head_modifier_distance = -5
```

After adding the distance between the head and modifier as a feature, I get an accuracy of **64.07%** and the number of features increased from 801 to **814**. The variation of accuracy with number of iterations on the development data, with no features, and after implementing the features of deliverable 8 are shown in Figure 1.

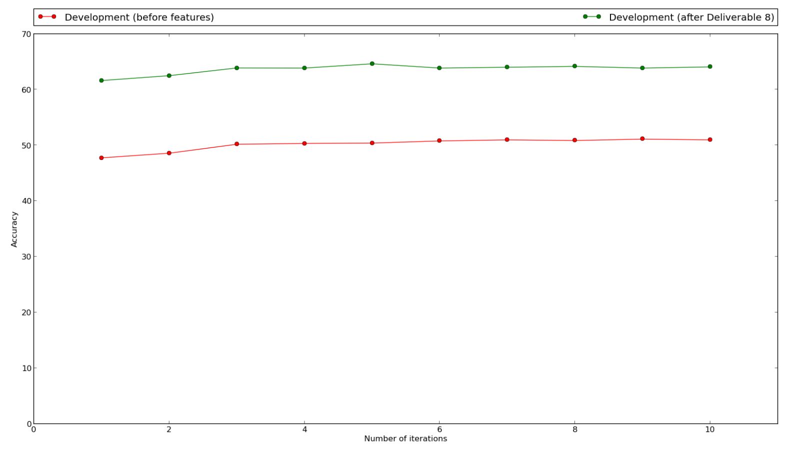

Figure 1: Accuracy vs. Number of iterations for development with and without features

Deliverable 9

I added the two features - (a) lexical features between the word of the head and tag of the modifier (b) between the word of the modifier and tag of the head. This significantly pushed up the accuracy on the development data, to **75.23%**. The accuracy on training data was **85.98%** (both depicted in Figure 2). Also the number of features significantly went up to **44539**.

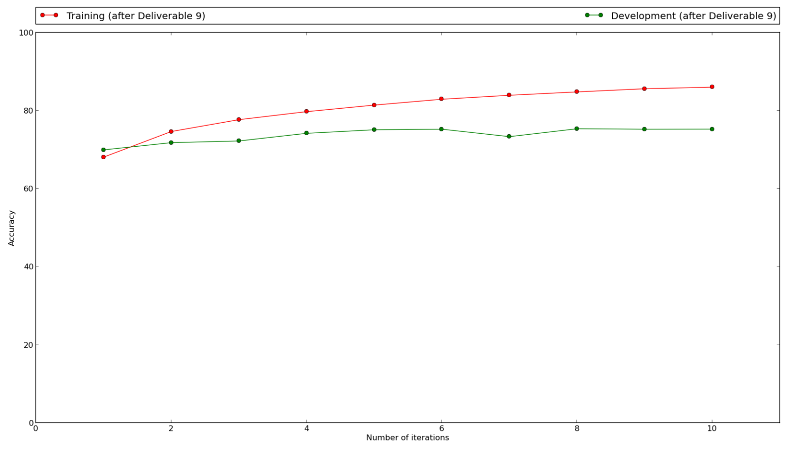

Figure 2: Accuracy vs. Number of iterations for training and development data after Deliverable 9

Deliverable 10

After adding the bi-lexical features between the head and the modifier (i.e the word of the head and the word of the modifier), the number of features almost doubles to **88473**. Both the training and development data accuracy increased slightly to **92.29%** and **76.78%**. Figure 3 represents this data.

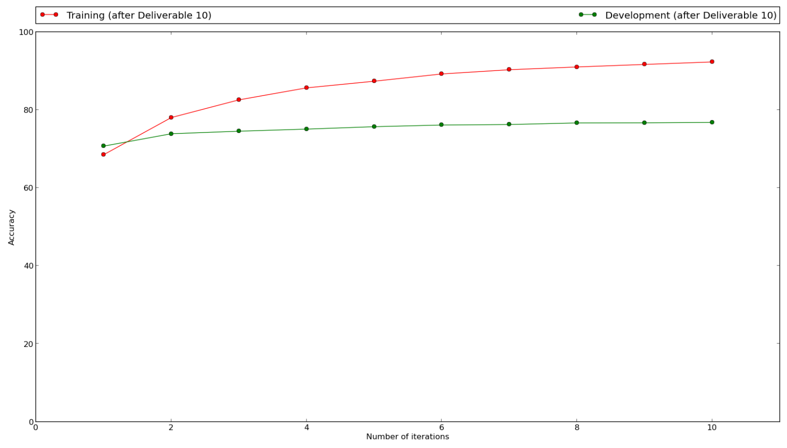

Figure 3: Accuracy vs. Number of iterations for training and development data after Deliverable 10

Deliverable 11

I added the following context features (the ones mentioned in the assignment + a few more combinations):

- $< t[h]; t[h-1]; t[m] >$: head, head-left, modifier

- $< t[h]; t[m]; t[m+1] >$: head, modifier, modifier-right
- $< t[h]; t[m-1]; t[m] >$: head, modifier-left, modifier
- $< t[h]; t[h-1]; t[m]; t[m+1] >$: head, head-left, modifier, modifier-right
- $< t[h]; t[h-1]; t[m]; t[m-1] >$: head, head-left, modifier, modifier-left
- $< t[h]; t[h+1]; t[m]; t[m-1] >$: head, head-right, modifier, modifier-left
- $< t[h]; t[h+1]; t[m]; t[m+1] >$: head, head-right, modifier, modifier-right
- $< t[h]; t[h-1]; t[h+1]; t[m]; t[m+1] >$: head, head-left, head-right, modifier, modifier-right
- $< t[h]; t[h-1]; t[h+1]; t[m]; t[m-1] >$: head, head-left, head-right, modifier, modifier-left
- $< t[h]; t[h-1]; t[h+1]; t[m]; t[m-1]; t[m+1] >$: head, head-left, head-right, modifier-left, modifier, modifier-right
- $< t[h]; t[h+k]; t[m] >$: for k in (1, head_modifier_distance). head, all head-rights, modifier
- $< t[h]; t[m-k]; t[m] >$ for k in (1, head_modifier_distance). head, all modifier-lefts, modifier

Note: I got the idea for the last two features from the paper for literature review.

I tried a few more combinations too, but beyond a point, the number of features went too high, making the program slow, but did not improve the accuracy rate at all. With the above mentioned context features, the total number of features come up to **273415**. The accuracy of training data and development data go upto **98.53%** and **89.39%** respectively. Figure 4 shows the how adding these features affects training and development accuracy.

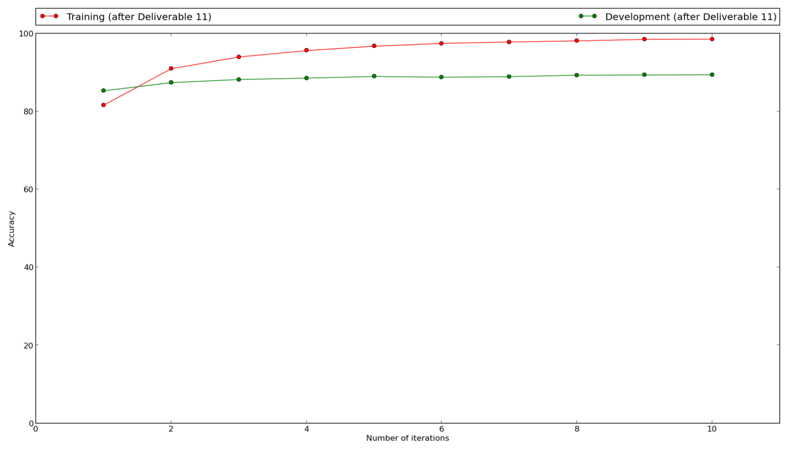

Figure 4: Accuracy vs. Number of iterations for training and development data after Deliverable 11

The increase in the number of features, from deliverable 7 (pair of tags in dependency arc) to deliverable 11 (context features) is depicted in Figure 5

2.3 Bakeoff

Deliverable 12

My best feature set consists of all the tag combinations that I have used from deliverable 8 to deliverable 11. After running the **dp.test()** function, I get the file *"english test.conll.pred"*. I have renamed it as *"jagannathan.conll.pred"* which is my submission for the bake off.

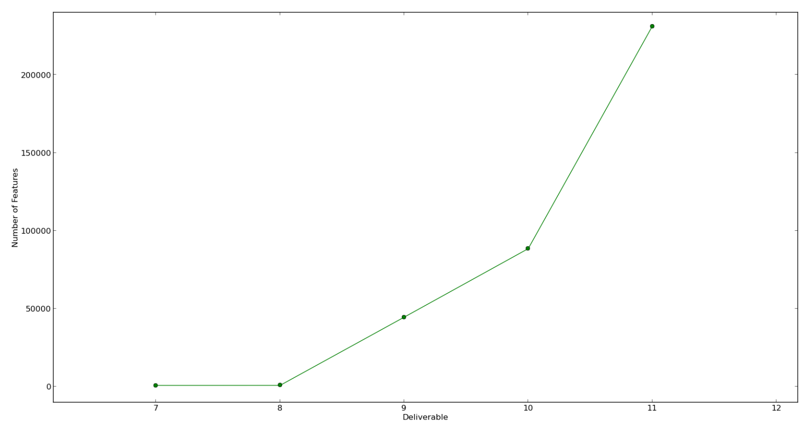

Figure 5: Increase in Feature Count (first two data points correspond to 801 and 814 respectively)

3 7650: Applications of Dependency Parsing

Deliverable 13

The paper I have chosen for Literature Review is **"Quadratic-time dependency parsing for machine translation"** [3] which utilizes dependency parsing to perform a phrase-based decoding/translation from Chinese to English and vice-versa.

Problem Addressed

The primary problem which the authors focus on is an **efficient** machine translation model, with respect to both speed (which is their top priority) and accuracy, of translating text in English to Chinese and vice-versa. The authors describe a technique to utilize maximum spanning tree (MST) parsing to machine translation, and how it can be integrated into a phrase-based decoder to compute dependency language model scores for a pair of source and target language.

Rationale behind using dependency parsing

There are two approaches that could be employed for machine translation - hierarchical and phrase-based. Hierarchical approaches [4], construct a tree-structure for each sentence, consisting of phrases which are recursively constructed using sub-phrases. A context free grammar could be constructed for such a hierarchical model, and sentences could be parsed using the CKY algorithm, and a translation could be performed for each parse. When this approach is applied, using a m-gram language model, the time complexity would be $\mathcal{O}(n^{3m})$ where n is the length of the sentence.

However the authors point out the results of the 2008 Open MT evaluation reveal that although many of the best systems performing Chinese-English translation incorporate such CFG models, they only perform slightly better than the best phrase-based parsers which utilize dependency parsing. The authors build on the dependency parsing algorithm of McDonald et al [5] which has a time complexity of $\mathcal{O}(n^2)$ rather than "full-fledged" chart parsing techniques, which take $\mathcal{O}(n^3)$ time, because (a) the difference in parsing accuracy is lower than 1% and (b) The authors favor a parser which has a lower time complexity since the translation process does not require state-of-the-art parsing techniques to be employed.

Thus, the authors use an *unlabeled* and non-projective dependency parsing technique. The parse tree is non-projective - meaning it may have crossing edges and the dependencies between the head and the modifier are not required to be nested. This relaxation is what allows the time complexity to be $\mathcal{O}(n^2)$. An interesting point mentioned in the paper is that, the time complexity still remains to be quadratic even if the order of the language model increases.

Methodology

- The dependency parsing is viewed as a simple sequence labeling task in which each word will be labeled with a "head". A pseudo "root" is assigned which will be the only word not to have a head.
- Their dependency parser uses a standard linear model to score each dependency relation. This is of the form,
  $s(i, j) = \lambda.\mathbf{f}(i, j)$
- λ corresponds to a weight vector which is obtained after training a structured perceptron augmented by MIRA.
- The score of the full tree is computed as a score of all the edges,
  $s(x, y) = \sum_{(i,j)\in\mathbf{y}} \lambda.\mathbf{f}(i, j)$
- Since this forms a directed graph, the Chu-Liu-Edmonds (CLE) algorithm is applied to find a maximum spanning tree out of it.
- During the application of CLE algorithm, while choosing the highest scoring edge for each modifier, the following translation information is generated by the translation decoder for the expansion,
  1. A partial translation $\mathbf{x}$
  2. A coverage set for the input words $\mathbf{c}$
  3. A translation score σ
  4. A predicted POS tag t_j
  5. A dependency score s_j
- A modification which the authors make to the CLE algorithm is that they do not identify loops every time there is an expansion of nodes. Instead they just reflect that in the dependency score.

Features used

The authors use features for the dependency parsing as well as for the phrase-based decoder. The features used for dependency parsing are,

1. **Unigram features**: h-word, h-pos, h-word & h-pos, m-word, m-pos, m-word & m-pos, where "h-" and "m-" correspond to the head word and modifier word respectively.
2. **Bigram features**: h-word & m-word, h-pos & m-pos, h-word & h-pos & m-word, h-word & h-pos & m-pos, m-word & m-pos & h-word, m-word & m-pos & h-pos, h-word & h-pos & m-word & m-pos
3. **Adjacent POS features**: h-pos & h-pos+1 & m-pos-1 & m-pos, h-pos & h-pos+1 & m-pos & m-pos+1, h-pos-1 & h-pos & m-pos-1 & m-pos, h-pos-1 & h-pos & m-pos & m-pos+1
4. **In-between features**: h-pos & h-pos + k & m-pos , where k∈ [i, min(i+5, j)]
   h-pos & m-pos - k & m-pos, where k∈ [max(i, j-5), j] (5 is the upper bound and -5 is the lower bound for the distance between head and the modifier as considered by the authors).

Here h-pos, m-pos refer to the POS tag of the head and modifier and h-word and m-word refer to the head word and modifier word. An offset of +k or -k indicate k elements to the right or left respectively.

The features used to perform decoding are,

1. Phrase-based translational probabilities
2. Lexically-weighted probabilities
3. Word penalties, phrase penalties and linear distortions
4. Language model score

Results

In this paper, the authors present the results of experiments run on Chinese-to-English language pairs. The training data consists of about 28 million English words and 23.3 million Chinese words drawn from various news parallel corpora distributed by the Linguistic Data Consortium (LDC).

Chinese words were segmented with a conditional random field classifier [6]. The authors built a large 5-gram language model for both English and Chinese, and performed Kneser-Ney smoothing (removing 4-grams and 5-grams occurring less than 3 times in training data). The authors used the NIST MT evaluation data for Chinese from 2002 to 2008 for performing the training and testing. Each Chinese sentence in this corpus has four corresponding English reference sentences. The authors present their results based on evaluation of their experiments on two metrics (a) BLEU and (b) TER

1. **BLEU**: Across all the data sets the authors were able to get a BLEU score of 32.29% on average without using a dependency language model and 32.74% after using a dependency language model.

2. **TER**: Across all the data sets the authors got a TER score of 58.02% on average without using a dependency language model and 57.10% on average after using a dependency language model.

The authors also present the results on the data broken down by genre. These are presented in Table 4

BLEU [%]				
Dependency Language Model	newswire	web	speech	all
no	32.86	21.75	36.88	32.29
yes	33.19	22.64	37.51	32.74
TRE [%]				
Dependency Language Model	newswire	web	speech	all
no	57.73	62.64	55.16	58.02
yes	56.73	61.97	54.26	57.1

Table 4: Results of Machine Translation Broken Down by Genre

With respect to timing measures, in the case of English translations of 40 words and shorter, the baseline system took 6.5 seconds per sentence, whereas the dependency LM system spent 15.6 seconds per sentence, i.e., 2.4 times the baseline running time. In the case of translations longer than 40 words, average speeds were respectively 17.5 and 59.5 seconds per sentence, i.e., the dependency was only 3.4 times slower.

Differences from (and Improvements to) existing methods

The non-projective dependency parser presented by the authors improves upon the cubic time implementations already in existence, with very little loss of accuracy. Also their translation results provide a 0.92% TER and 0.45% BLEU absolute improvements. The authors claim that at the scale of data that their algorithm operates on, even such minor improvements are considerable achievements.

References

[1] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[2] K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N.A. Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. Technical report, DTIC Document, 2010.

[3] Michel Galley and Christopher D Manning. Quadratic-time dependency parsing for machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International*

Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2, pages 773–781. Association for Computational Linguistics, 2009.

[4] David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics, 2005.

[5] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.

[6] Pi-Chuan Chang, Michel Galley, and Christopher D Manning. Optimizing chinese word segmentation for machine translation performance. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pages 224–232, 2008.