A Joint Model for Pause Prediction and Dependency Parsing using Latent Variables

The Tung Nguyen, Graham Neubig, Hiroyuki Shindo,
Sakriani Sakti, Tomoki Toda[1]*, Satoshi Nakamura*

Graduate School of Information Science, Nara Institute of Science and Technology

1. Introduction

Prediction of prosodic information from text is a basic technology used in a number of speech-related applications. In particular, pauses prediction is used in speech synthesis to allow for more natural prosodic boundaries [1]. While early studies in pause prediction only relied on lexical information such as POS tags or punctuation [1, 2], recent works have shown that using syntactic structure helps achieve better accuracy [3].

This work focuses on the use of dependency structure for pause prediction. Our method is inspired by recent work by Honnibal and Johnson [4] on the related, but quite different task, of disfluency detection. In this work, they propose a model that jointly performs dependency parsing and disfluency detection, and demonstrate that a joint model that considers these two tasks improves over solving these two tasks individually. Thus, in this paper, we propose a method for joint *dependency parsing and pause prediction*.

One of the most widely used methods for dependency parsing is the transition-based method based on the *shift-reduce* algorithm [5]. As shown in the black text of Figure 1, and explained in detail in Section 2, the shift-reduce method builds a dependency tree expressing the syntactic structure of the sentence by performing a series of "shift" and "reduce" actions, and if the correct action sequence is chosen, the correct dependency tree will be created. A classifier to choose the correct answer is trained from syntactically annotated data.

In our proposed model, we further expand the action set of the shift-reduce algorithm by adding actions that predict pauses, as shown in the bold, red text in Figure 1, and is described in detail in Section 3. This presents more difficulties in training, however, as it is necessary to have data that is annotated with both pauses and syntactic information, which cannot be obtained in large quantities. To solve this problem, we treat the pauses as latent variables, allowing our parsing model to be trained on data fully annotated with syntax and pauses as well as data only annotated with syntactic trees.

In the experiments described in Section 4, we find that the proposed model exceeds all baselines, and that the proposed latent variable allows for effective use of data not explicitly annotated with pauses, resulting in an 11.6% absolute improvement in pause F-measure.[2]

2. Shift-reduce dependency parsing

2.1. Shift-reduce algorithm

Shift-reduce is a dependency parsing algorithm that parses sentences one word at a time from left to right [5]. Shift-reduce parsers use a queue to store the input words and a stack to store the dependency structure that has been built so far. The shift-reduce process converts the input sentence into dependency structure through a sequence of actions, an example of which is shown in Figure 1. Our work adopts a modified version of the arc-standard shift-reduce algorithm proposed by [7], which has three different actions:

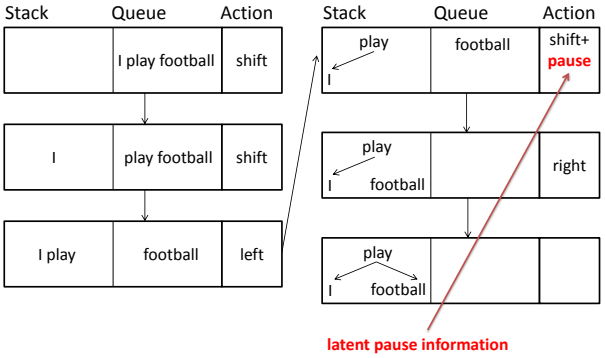

Figure 1: *An example of shift-reduce dependency parsing, and the proposed pause prediction action.*

SHIFT: remove the first word in the queue and put it into the stack.

LEFT: create an edge from the first word in the stack to the second, then remove the second word from the stack.

RIGHT: create an edge from the second word in the stack to the first, then remove the first word from the stack.

For each edge in the dependency tree, the parent node is called the head and the child node is called a dependent. Each word have exactly one head and can have multiple dependents. To create the dependency structure of a sentence, our task is to choose the sequence of shift-reduce actions $\boldsymbol{a}$ that achieve the correct tree $\boldsymbol{y}$ from input sentence $\boldsymbol{x}$. Note that in this modified arc-standard shift-reduce framework, there is a one-to-one correspondence between action sequences $\boldsymbol{a}$ and trees [8], so for convenience in the rest of this paper, we will assume that we are attempting to find the correct action sequence $\boldsymbol{a}$.

At every step of the shift-reduce process, the score of each action a is calculated by the following formula:

$$Score(S, Q, a) = \boldsymbol{w}^T \boldsymbol{f}(S, Q, a). \quad (1)$$

S is the stack and Q is the queue of the current step. $\boldsymbol{f}(S, Q, a)$ creates a feature vector given S, Q, and a, and $\boldsymbol{w}$ is the corresponding feature weight vector. The score of an action is the weighted sum of each feature of the current stack, queue and the corresponding action.

[1] Now with Nagoya University.
[2] This paper is a shortened version of [6].

There are two ways to build dependency trees with the shift-reduce algorithm: greedy search and beam search. With greedy search, at each step the action with highest score is chosen. We repeat this process until both the stack and the queue are empty. With beam search, we choose a beam size k. At each step we define the score of the sequence of actions $\boldsymbol{a} = a_1, \ldots, a_M$ by the formula:

$$Score(\boldsymbol{a}) = \sum_{i=1}^{M} Score(S_{i-1}, Q_{i-1}, a_i), \qquad (2)$$

S_i and Q_i are the stack and queue resulting from action a_i in sequence A. At each step, a set of new sequences of actions is generated by applying each of the 3 actions to the k sequences in the beam from the previous time step and choose the best k from the resulting sequences. Previous research has found that beam search achieves better accuracy [9, 10], and thus, we use beam search in the rest of this paper. Our parser is also capable of labeled parsing, in which the parser predicts not only the dependency arc, but also the grammatical relation represented by the dependency [11]. This can be done by simply augmenting LEFT and RIGHT with dependency labels.

2.2. Structured perceptron with latent variables

The structured perceptron is a structured classifier introduced by Collins [12]. We assume a set of I training examples $\{\langle \boldsymbol{x}_1, \boldsymbol{a}_1 \rangle, \ldots, \langle \boldsymbol{x}_I, \boldsymbol{a}_I \rangle\}$ where $\boldsymbol{x}_i$ is a sequence of observations and $\boldsymbol{a}_i$ is the corresponding action sequence for $\boldsymbol{x}_i$. Let $\boldsymbol{w}$ be the weight vector and function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{a})$ be the feature vector function calculated from observations $\boldsymbol{x}$ and action sequence $\boldsymbol{a}$. The algorithm processes every sample in the training data one-at-a-time, first performing the decoding step to obtain the highest scoring action sequence $\boldsymbol{a}'_i$. Next, if $\boldsymbol{a}'_i$ is different from the gold sequence $\boldsymbol{a}_i$ then we update the weights by increasing their value by the feature vector of the correct sequence $\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{a}_i)$ and decreasing its value by the features of the 1-best parse $\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{a}'_i)$.

Recently, the perceptron has also been extended to handle additional latent variables [13, 11]. Given the sequence of labels $\boldsymbol{a} = a_1, \ldots, a_M$, in the latent structured perceptron, assume that we have some latent information expressed as a sequence of hidden variables $\boldsymbol{h} = h_1, \ldots, h_M$. We assume that each label a corresponds to a set of latent labels H_a, and that the sets of latent labels for each a are disjoint from each other. For example, Bohnet and Nivre [11] considered the case of dependency parsing without POS tag information. In this case, $\boldsymbol{x}$ is the sequence of words of input sentence. $\boldsymbol{a}$ is sequence of standard shift-reduce actions, while $\boldsymbol{h}$ further appends a POS tag to each of the actions. The training algorithm for the structured perceptron with latent variables is described in Algorithm 1.

The training algorithm is very similar to the normal structured perceptron except for a few minor modifications. In the decoding step, the result is the sequence of latent variables $\boldsymbol{h}'_i$ that has the highest score among all sequences of latent variables. Then we make a projection from the sequence of latent variables $\boldsymbol{h}'_i$ to a sequence of labels $\boldsymbol{a}'_i$ in which $\boldsymbol{a}'_{i,j}$ is chosen such that $\boldsymbol{h}'_{i,j} \in H_{\boldsymbol{a}'_{i,j}}$. We then compare this projected sequence with the true sequence $\boldsymbol{a}_i$ and if they are different, perform a perceptron update. When we perform an update, we would like to update towards the "correct" sequence of latent variables. However, since the correct sequence of latent variables is unobservable, we choose the sequence $\boldsymbol{h}_i$ that has highest score but satisfies the constraint $\boldsymbol{a}_i$ = project($\boldsymbol{h}_i$). This step is called forced decoding, which we express using the forced_decode($\cdot$) function. For updating, the features are created from sequences of observations and latent variables.

Algorithm 1 Structured perceptron with latent variables

Input: Set of training samples D.
Init: Feature vector $\boldsymbol{w}$ = 0.
for $iter \leftarrow 1, max$ **do**
  for ($\boldsymbol{x}_i$,$\boldsymbol{a}_i$) $\in D$ **do**
    $\boldsymbol{h}'_i$ = decode($\boldsymbol{w}, \boldsymbol{x}_i$)
    $\boldsymbol{a}'_i$ = project($\boldsymbol{h}'_i$)
    if $\boldsymbol{a}'_i \neq \boldsymbol{a}_i$ **then**
      $\boldsymbol{h}_i$ = forced_decode ($\boldsymbol{w}, \boldsymbol{x}_i, \boldsymbol{a}_i$)
      $\boldsymbol{w}$ = $\boldsymbol{w}$ + $\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{h}_i)$ - $\boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{h}'_i)$

3. Dependency parsing with latent pause information

3.1. Using pause information as latent variables

Previous work [3] has shown that syntactic structure has an important role in pause prediction. However, to our knowledge there are no models that tackle dependency parsing and pause prediction jointly. Thus in this paper, we build on the recent success of latent variable models for joint dependency parsing and POS tagging [11], proposing a latent variable joint model for pause prediction and dependency parsing. To do that, we modify the shift-reduce algorithm to be able to perform the two tasks jointly. Specifically, we divided pauses into two classes:

NoPause: no pause is perceivable to the human ear.
Pause: a pause perceivable to the human ear exists.
Next, we create a new set of actions:
RIGHT: same as standard shift-reduce.
LEFT: same as standard shift-reduce.
SNONE: same as SHIFT, for words preceded by NoPause.
SPAUSE: same as SHIFT, for words preceded by Pause.

Each SHIFT action in a sentence corresponds to a single word, and thus the set of actions above allow us to perform pause prediction for each word in the sentence. By using these new actions, our parser can both predict the syntactic structure and the pauses between words in parallel. The bold, red line and words in Figure 1 show this additional latent information. In this case, we predict that there is a pause between two words "play" and "football" and thus the action shift-pause (SPAUSE) is chosen.

With the modified set of shift-reduce actions, if we have data annotated with both pauses and dependency syntax, the model can be trained using the normal structured perceptron. However, because creating data with both manually checked pauses and manually checked parse trees is laborious, we cannot obtain this data in large quantities. Since the quantity of data is insufficient, the parsing accuracy of a model trained on such data will not be very high. However, if we were able to use syntactic data without pauses to increase dependency parsing accuracy, this could potentially increase the accuracy of the model as a whole.

Thus we envision a training setting in which we have a large set of text data D_t annotated with only syntax trees, and a smaller set of speech data D_s annotated with syntax trees and pauses. In order to train on the data that is not annotated with pauses, the proposed method treats pauses as latent variables. Thus, we use the standard structured perceptron on our small set of data D_s annotated with both types of information, and

Algorithm 2 Training algorithm for the proposed model.

Input: Set of training samples $D = D_t \cup D_s$.
D_t is the set of text data with only syntax.
D_s is the set of speech data with syntax and pauses.
Init: Feature vector $\boldsymbol{w} = 0$.
for $iter \leftarrow 1, max$ **do**
  for $(\boldsymbol{x}_i, \boldsymbol{a}_i) \in D$ **do**
    if $(\boldsymbol{x}_i, \boldsymbol{a}_i) \in D_t$ **then** ▷ latent perceptron
      $\boldsymbol{h}'_i$ = decode($\boldsymbol{w}$,$\boldsymbol{x}_i$)
      $\boldsymbol{a}'_i$ = project($\boldsymbol{h}'_i$)
      if $\boldsymbol{a}'_i \neq \boldsymbol{a}_i$ **then**
        $\boldsymbol{h}_i$ = forced_decode($\boldsymbol{w}$,$\boldsymbol{x}_i$,$\boldsymbol{a}_i$)
        $\boldsymbol{w} = \boldsymbol{w} + f(\boldsymbol{x}_i, \boldsymbol{h}_i)$ - $f(\boldsymbol{x}_i, \boldsymbol{h}'_i)$
    else ▷ standard perceptron
      $\boldsymbol{h}'_i$ = decode($\boldsymbol{w}, \boldsymbol{x}_i$)
      if $\boldsymbol{h}'_i \neq \boldsymbol{h}_i$ **then**
        $\boldsymbol{w} = \boldsymbol{w} + f(\boldsymbol{x}_i, \boldsymbol{h}_i)$ - f($\boldsymbol{x}_i$,$\boldsymbol{h}'_i$)

the latent perceptron on D_t annotated with only syntax information.

By modifying the standard algorithm, we can train the model on the combined data as shown in Algorithm 2.

3.2. Features

Most of the features used in our model were adopted from previous work [5, 7, 10]. We define S as the stack to store the

Features
Single word: $S_{-1}w$, $S_{-1}p$, $S_{-1}wp$, $S_{-1}wi$, $S_{-2}w$, $S_{-2}p$, $S_{-2}wp$, $S_{-2}wi$, $S_{-3}w$, $S_{-3}p$, $S_{-3}wp$, $S_{-3}wi$, Q_0w, S_0p, Q_0wp, Q_0wi.
Two words: $S_{-3-2}w$, $S_{-3-2}p$, $S_{-3-2}wp$, $S_{-2-1}w$, $S_{-2-1}p$, $S_{-2-1}wp$, $S_{-1}Q_0w$, $S_{-1}Q_0p$, $S_{-1}Q_0wp$, $S_{-2}wpS_{-1}wp$, $S_{-2}pS_{-1}wp$, $S_{-2}wS_{-1}wp$, $S_{-2}wpS_{-1}w$, $S_{-2}wpS_{-1}p$, $S_{-1}wpQ_0wp$, $S_{-1}wpQ_0p$, $S_{-1}wpQ_0w$, $S_{-1}wQ_0wp$, $S_{-1}pQ_0wp$.
Three words: $S_{-3-2-1}w$, $S_{-3-2-1}p$, $S_{-2-1}Q_0w$, $S_{-2-1}Q_0p$.
Index+Distance: $S_{-1}wi$, $S_{-1}pi$, Q_0wi, Q_0pi, $S_{-1}Q_0wd$, $S_{-1}Q_0pd$.
Modifiers: $S_{-3}lw$, $S_{-3}lp$, $S_{-3}rw$, $S_{-3}rp$, $S_{-2}lw$, $S_{-2}lp$, $S_{-2}rw$, $S_{-2}rp$, $S_{-1}lw$, $S_{-1}lp$, $S_{-1}rw$, $S_{-1}rp$.
Previous word: $Q_{-1}w$, $Q_{-1}p$,$Q_{-10}w$, $Q_{-10}p$.

Table 1: List of features

dependency structure that has been built so far and Q to be the queue storing the input sentence. S_{-3}, S_{-2} and S_{-1} are the top three elements of the stack, Q_0 is the first element in the queue and Q_{-1} is the one before it in the sentence. In Table 1, w indicates the surface form of the word, and p indicates its part-of-speech (POS) tag. For example, $S_{-1}Q_0wp$ is the combination of text and POS tag of the word on the top of the stack and the first word in the queue. We also define features regarding modifiers of stack items, where l indicates the leftmost modifier and r indicates the rightmost modifier of the word. The position of the word in the sentence is also used in our model where i indicates the index of the word (starting at 1 for the first word in the sentence). Finally, we define features over distance between two words denoted as d.

4. Experiments

4.1. Experimental setup

We conducted experiments to test the effectiveness of the proposed joint dependency parsing and pause prediction model. To do so, we first prepare both text and speech data.

Text: As text data, we used the Wall Street Journal Section of the Penn Treebank [14], the standard data set for English parsers' experiments. Following convention, sections 2-21 were used for training, section 23 was used for testing and sections 22&24 were used for development.

Speech: As speech data, we used the NAIST-NTT Ted Talk Treebank [15], which is a corpus of TED talks annotated with syntactic structure. We manually annotated all talks in the treebank with audibly perceptible pauses by first performing forced alignment using an ASR system [16] to automatically detect sections of silence, then having a human annotator manually correct these pauses.[1]

Details of the size of each data set are shown in Table 2. We created models on two different sets of data: TED and TED+WSJ. Test sets from WSJ and TED were used to measure the accuracy

Data Set	Train	Test
WSJ	39,832	2,416
TED	822	395

Table 2: Data size in sentences

of dependency parsing. To measure the performance of pause prediction, only test data from TED was used. The beam size used for beam search is fixed at 12 in all experiments.

We compare with two baseline models: "All Pause" and "Dec. Tree." "All Pause" is a trivial baseline that always selects pauses for every word. "Dec. Tree" is a model based on a research by Hirschberg and Prieto [2] that used decision trees for learning intonational phrasing rules. We adopted their work for a baseline model for pause prediction task. Due to the differences between the two tasks, we only use a subset of their features relevant to our task, specifically POS tags of the current word, previous word and the combination of them. Finally, to examine the effect of predicting both syntax and pauses jointly, we test a "Pipeline" built by using two separate models, the first one is a normal parser model trained on WSJ data and the second is a joint model trained on TED data. Then, we use the first model to predict parse trees on TED test data and used these trees for forced decoding using the second model. As an evaluation measure for pause prediction, we use pause prediction F-measure:

$$F = \frac{2 \times precision \times recall}{precision + recall}. \quad (3)$$

Statistical significance of accuracy differences between methods was measured using pairwise bootstrap resampling with 95% confidence [17].

[1] Note that the written text in WSJ is likely slightly different in syntax than TED talks, but previous work [15] has shown that WSJ is data is still useful in parsing of TED speech.

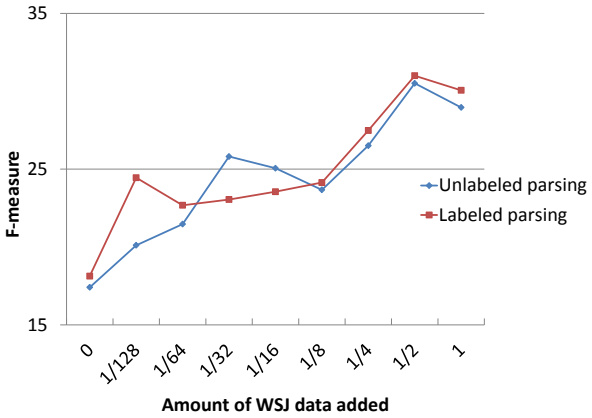

Figure 2: *Influence of WSJ data on pause prediction F-measure.*

4.2. Results

First, we show precision, recall, and F-measure for the pause prediction task in Table 3. All the models in this table are trained using the unlabeled parsing method. If we first compare the first three rows, it is clear that our joint dependency parsing and pause prediction model performs better than the baseline models. Next, and more interestingly, we can see that adding WSJ data greatly improves recall and F-measure. This is notable, because the WSJ data contains no annotation of pauses, demonstrating that by improving the accuracy of parsing itself, we were able to improve pause prediction as well.

To examine this relationship in more detail, in Figure 2 we show results of pause prediction F-measure for various sizes of WSJ data. Although the increase is not entirely consistent, we can see that in general by adding more WSJ data to TED data, we improve F-measure for pause prediction significantly. We also performed experiments on both unlabeled parsing and labeled parsing. From the chart, the difference is not consistent, although labeled parsing tends to achieve slightly higher scores.

Finally, we examine the effect of latent pauses on dependency parsing accuracy. In this test, accuracy was measured using UAS - Unlabeled Attachment Score. Compared to a standard model that used no latent pause information, the proposed model achieved essentially the same accuracy, with UAS scores for the standard and proposed models being 91.32% and 91.25% on WSJ, and 85.26% and 85.54% on TED respectively.

Model	Data	P	R	F
All Pause	—	6.99%	**100.00%**	13.07%
Dec. Tree	TED	**37.77%**	3.42%	6.28%
Parser	TED	26.78%	12.90%	17.41%
Pipeline	TED+WSJ	27.87%	17.74%	22.19%
Parser	TED+WSJ	26.41%	32.06%	**28.96%**

Table 3: Pause prediction precision, recall and F-measure. Bold indicates a statistically significant gain over other methods.

5. Conclusions

In this paper, we introduced a method that is able to jointly perform dependency parsing and pause prediction. Our model achieved an F-measure of 28.96% on our pause prediction task, surpassing several baselines. Our most notable result is that a model trained on a combination of data annotated with pauses and syntax, as well as data annotated with only syntax achieved better results on pause prediction than when using only the data annotated with pauses. In the future, we hope to adapt this latent variable model to other tasks such as joint dependency parsing and disfluency detection.

Acknowledgements: Part of this work was supported by JSPS KAKENHI Grant Number 24240032, and grants from Nippon Telegraph and Telephone Corp. and Honda Research Institute Japan Co., Ltd.

6. References

[1] A. W. Black and P. A. Taylor, "Assigning phrase breaks from part-of-speech sequences." *Computer speech and language*, vol. 12, pp. 99–117, 1998.

[2] J. Hirschberg and P. Prieto, "Training intonational phrasing rules automatically for english and spanish text-to-speech," *Speech Communication*, vol. 18, no. 3, pp. 281–290, 1996.

[3] J. Tauberer, "Predicting intrasentential pauses: Is syntactic structure useful?" in *Proc. Speech Prosody*. Citeseer, 2008, pp. 405–408.

[4] M. Honnibal and M. Johnson, "Joint incremental disfluency detection and dependency parsing," *Transactions of the Association of Computational Linguistics*, 2014.

[5] H. Yamada and Y. Matsumoto, "Statistical dependency analysis with support vector machines," in *Proc. IWPT*, vol. 3, 2003, pp. 195–206.

[6] T. T. Nguyen, G. Neubig, H. Shindo, S. Sakti, T. Toda, and S. Nakamura, "A latent variable model for joint pause prediction and dependency parsing," in *Proc. InterSpeech*, Dresden, Germany, September 2015.

[7] L. Huang, W. Jiang, and Q. Liu, "Bilingually-constrained (monolingual) shift-reduce parsing," in *Proc. EMNLP*, 2009, pp. 1222–1231.

[8] Y. Goldberg and J. Nivre, "A dynamic oracle for arc-eager dependency parsing." in *Proc. COLING*, 2012, pp. 959–976.

[9] Y. Zhang and S. Clark, "Joint word segmentation and pos tagging using a single perceptron." in *Proc. ACL*, 2008, pp. 888–896.

[10] Y. Zhang and J. Nivre, "Transition-based dependency parsing with rich non-local features," in *Proc. ACL-HLT*, 2011, pp. 188–193.

[11] B. Bohnet and J. Nivre, "A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing," in *Proc. EMNLP*, 2012, pp. 1455–1465.

[12] M. Collins, "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms," in *Proc. EMNLP*, 2002, pp. 1–8.

[13] X. Sun, T. Matsuzaki, D. Okanohara, and J. Tsujii, "Latent variable perceptron algorithm for structured classification." in *Proc. IJCAI*, vol. 9, 2009, pp. 1236–1242.

[14] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[15] G. Neubig, K. Sudoh, Y. Oda, K. Duh, H. Tsukada, and M. Nagata, "The NAIST-NTT TED talk treebank," in *Proc. IWSLT*, Lake Tahoe, USA, December 2014.

[16] S. Sakti, K. Kubo, G. Neubig, T. Toda, and S. Nakamura, "The NAIST english speech recognition system for IWSLT 2013," in *Proc. IWSLT*, Heidelberg, Germany, December 2013.

[17] P. Koehn, "Statistical significance tests for machine translation evaluation," in *Proc. EMNLP*, 2004.

[18] M. Collins and B. Roark, "Incremental parsing with the perceptron algorithm," in *Proc. ACL*, 2004, p. 111.