$\mathcal{HC}$-Search for Incremental Parsing

Yijia Liu, Wanxiang Che [*], Bing Qin, Ting Liu
Research Center for Social Computing and Information Retrieval
Harbin Institute of Technology, China
{yjliu,car,qinb,tliu}@ir.hit.edu.cn

Abstract

Standard incremental parsing algorithm employs a single scoring function and beam-search to find the best parse tree from an exponentially large search space. Inspired by recently proposed $\mathcal{HC}$-search framework, we decompose the incremental parsing algorithm into two steps: first searching a set of high-quality outputs with beam-search, and second selecting the best output with a ranking model. We learn our incremental parsing model with a relaxed learning objective. We incorporate arbitrary features in our ranking model and learn the model from fine grain ranking examples. Experimental results on standard English and Chinese datasets show our method significantly outperforms a strong baseline.

1 Introduction

Transition-based dependency parsing models the derivation of a parse tree (as illustrated in Figure 1) as a sequence of parsing actions [Nivre, 2008]. Finding the correct tree is transformed into a search-based structured prediction problem [Collins, 2002; Daumé *et al.*, 2009] of finding the optimal action sequence from an exponentially large search space. The search problem is treated as a left-to-right incremental process, namely, the *incremental parsing* [Collins and Roark, 2004] and inexact search algorithms like greedy search and beam-search are adopted. Compared with greedy search, beam-search reduces search errors and achieves better performance, especially when guided by a discriminative model learned with structured perceptron algorithm.

The most basic approach to learn such model is via imitation of one *oracle action sequence* and making corrective updates to enforce the model to score the oracle sequence high. Although empirically performs well, such learning approach faces several problems. One of the problems can be resulted from the intrinsic *spurious ambiguity* in the transition system [Goldberg and Nivre, 2012], which is, one gold tree can be derived by multiple action sequences. Such ambiguity can confuse the imitation learning process and lead the model to improperly score the "non-oracle but correct"

[*]Email corresponding.

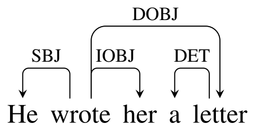

Figure 1: An example dependency parse tree.

action sequence. Another problem is that for each state in the search process, only a partial tree is constructed, while for the final states, information of the tree can be fully recovered. Incremental parsing treats these two different states in the same way and doesn't efficiently use the full tree information, comparing to the global parsing methods like dual-decomposition [Martins *et al.*, 2010] and sampling [Zhang *et al.*, 2014].

Inspired by the recent proposed $\mathcal{HC}$-search [Doppa *et al.*, 2014a], especially its similarity to the incremental parsing in term of loss decomposition, we propose a new method which decomposes the incremental parsing into two steps: $\mathcal{H}$-step and $\mathcal{C}$-step. In the $\mathcal{H}$-step, we perform beam-search to uncover high-quality candidate outputs. In the $\mathcal{C}$-step, we apply a ranking model to select the best loss output. By such decomposition, we are allowed to adopt a relaxed learning objective to learn the incremental parsing model in the $\mathcal{H}$-step and it mitigates improperly scoring problems. We treat search and ranking differently by incorporating global features in the $\mathcal{C}$-step and gain improvement by learning the model with fine grain ranking examples. Contributions of this work fall into the following parts:

- We follow the loss decomposition in the $\mathcal{HC}$-search framework and decompose the incremental parsing into two steps: $\mathcal{H}$-step for uncovering high-quality candidate outputs and $\mathcal{C}$-step for selecting the best output.
- We study the problem of adopting a relaxed learning objective to learn the $\mathcal{H}$-step model and it achieves both stage-wise and overall better performance.
- We also study different ranking examples generation approaches when learning the $\mathcal{C}$-step model and find it necessary to learn from fine grain ranking examples.
- We conduct our experiments on standard datasets and the results show our method outperforms a strong baseline by an error reduction of 3.63% on English and

Algorithm 1: Beam-search for one step, i.e. BEST_k.

```
Input: $x$ = the input, $\mathcal{B}_j$ = list of states in previous step, $\vec{w}$ =
       the weight, $s$ = the scoring function.
1  $\mathcal{B} \leftarrow \emptyset$
2  for each state $c \in \mathcal{B}_j$ do
3      for each transition action $t \in T$ do
4          $c' \leftarrow t(c)$, $c'.score \leftarrow c.score + s(x, c', \vec{w})$
5          $\mathcal{B} \leftarrow \mathcal{B} \cup c'$
6  sort $\mathcal{B}$ according to the score of each state.
7  return $\text{TOP-K}(\mathcal{B})$, k-highest scoring states in $\mathcal{B}$.
```

Algorithm 2: Learning algorithm for standard incremental parsing.

```
Input: $\mathcal{D}$ = the training instances, $R$ = the max iteration.
Output: $\vec{w}$, The weight for incremental parsing model.
1  $\vec{w}_0 \leftarrow \vec{0}, t \leftarrow 0$                    ▷ t, the timestamp
2  for $r \leftarrow 1..R$ do
3      for each training instance $(x, y^*) \in \mathcal{D}$ do
4          $C_{0..m} \leftarrow \text{ORACLEFINDING}(y^*)$
5          $\mathcal{B}_0 \leftarrow \{c_0\}$
6          for $j \leftarrow 1..m$ do
7              $\mathcal{B}_j \leftarrow \text{BEST}_k(x, \mathcal{B}_{j-1}, \vec{w}_t)$
8              if $c_j \notin \mathcal{B}_j$ then
9                  $C'_{0..j} \leftarrow \text{argmax}_{c' \in \mathcal{B}_j} s(x, C_{0..c'}, \vec{w}_t)$
10                 $\vec{w}_{t+1} \leftarrow \text{UPDATE}(C_{0..j}, C'_{0..j}, \vec{w}_t)$
11                 break
12         if $c_m \neq \text{argmax}_{c' \in \mathcal{B}_m} s(x, C_{0..c'}, \vec{w}_t)$ then
13             $C'_{0..m} \leftarrow \text{argmax}_{c' \in \mathcal{B}_m} s(x, C_{0..c'}, \vec{w}_t)$
14             $\vec{w}_{t+1} \leftarrow \text{UPDATE}(C_{0..m}, C'_{0..m}, \vec{w}_t)$
15         $t \leftarrow t + 1$
16 return $\vec{w}$, which is average of $\vec{w}_0, .., \vec{w}_t$
```

5.21% on Chinese. Further improvements are achieved when combining two steps together.

We release our code at https://github.com/ExpResults/hc-incremental-parsing.

2 Transition-based Incremental Parsing

We follow Nivre [2008] and define the transition-based parsing algorithm as a *transition system* $S = (C, T, I, C_t)$, where C is a set of states, T is a set of transition actions, I is the initial states and C_t is a set of terminal states. Given a sentence x, parsing is performed by starting from an initial state $c_0 \in I$, and repeatedly applying transition action $t \in T$ to the current state $c \in C$ until a terminal state $c_m \in C_t$ is reached. The action sequence $C_{0..m} = (c_0, c_1, .., c_m)$ is a sequence of states resulted by actions, where $c_0 \in I$, $c_m \in C_t$, and $t_i(c_i) = c_{i+1}$ given t_i is the ith step transition action. In this paper, we use the *arc-standard* algorithm [Nivre, 2008] as our transition system, which has three transition actions: LEFT-ARC (LA), RIGHT-ARC (RA) and SHIFT (SH). [1]

The beam-search algorithm is used to derive the best parse tree $\hat{y}$, which searches for the highest scoring action sequence step by step. At each step, the beam-search algorithm keeps a list (i.e. beam) of high-quality states with the guidance of a scoring function and applies transition actions only to the states in beam to result in states for next step, as outlined in Algorithm 1. For a state c, standard incremental parsing parameterizes the scoring function as $s(x, c, \vec{w}) = \vec{w}\Phi(x, c)$, where $\vec{w}$ is the parameters and $\Phi(x, c)$ extracts features from the state. For an action sequence $C_{0..m}$, such score is $\sum_{c \in C_{0..m}} s(x, c, \vec{w})$.

[1]refer to Nivre [2008] for details about the *arc-standard* system.

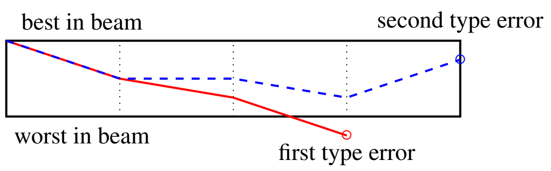

Figure 2: The two types of errors in learning the incremental parsing model. The solid line represents the oracle sequence falls out of beam at the 3rd step. The dashed line represents the oracle sequence stays in beam at the final step but is not the highest scoring one.

The learning and decoding for the incremental parsing are closely related, which decodes the training instance and make corrective updates when search error is made in order to score the gold tree (derived by an *oracle action sequence*) highest. More specifically, during learning, the algorithm iterates through all the training instances. On each instance (x, y^*), the algorithm first generates an oracle action sequence $C_{0..m}$ from y^* (the ORACLEFINDING function), then performs beam-search on x to find the highest scoring action sequence $C'_{0..m}$. If search error is made, which means $C'_{0..m}$ is not identical to $C_{0..m}$, the algorithm makes *perceptron updates* (the UPDATE function) to enforce $s(x, c, \vec{w})$ to give a higher score to $C_{0..m}$ (*positive sample*) than $C'_{0..m}$ (*negative sample*).

To guarantee the correctness of perceptron updates under approximate beam-search decoding, search errors can be categorized into two types: 1) $s(x, c, \vec{w})$ fails to uncover gold tree before search finishes, in which the gold state $c_j \in C_{0..m}$ falls out of the beam $\mathcal{B}_j$ at jth step; 2) $s(x, c, \vec{w})$ fails to score gold tree highest, in which the gold state c_m stays in the final step beam $\mathcal{B}_m$ but is not the highest-scored state [Collins and Roark, 2004; Huang *et al.*, 2012]. Figure 2 illustrates these two types of errors. When the first type of error occurs, the beam-search stops and the algorithm updates the parameters with partial sequence pair $(C_{0..j}, C'_{0..j})$. For the second type of error, the algorithm updates with full sequence pair $(C_{0..m}, C'_{0..m})$. The learning algorithm is shown in Algorithm 2. Line 8-11 illustrates the condition for the first type of error, and line 12-14 is for the second type of error.

3 $\mathcal{HC}$-search for Incremental Parsing

3.1 $\mathcal{HC}$-search

By reviewing the learning algorithm for standard incremental parsing, we can see that the single scoring function serves two roles: 1) guiding the search process towards the gold tree by minimizing the first type of error and 2) scoring the gold

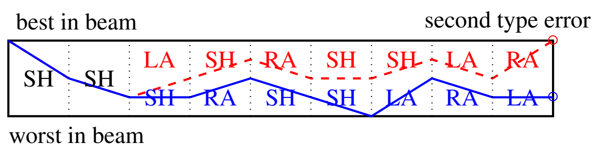

Figure 3: Example of the learning algorithm for standard incremental parsing being inconsistent in minimizing the two types of errors and improperly punishing the "non-oracle but correct" action sequence. The parse tree in Figure 1 can be derived by 1) the oracle sequence (SH, SH, SH, RA, SH, SH, LA, RA, LA) in solid line and 2) the "non-oracle but correct" sequence (SH, SH, LA, SH, RA, SH, SH, LA, RA) in dashed line. In this case, the second type of error occurs and the learning algorithm updates parameters with 2) as a negative sample which increases the first type of error.

tree as the highest one by minimizing the second type of error. For standard incremental parsing, these two roles are almost consistent with each other because scoring the oracle action sequence highest indicates minimizing both the first and the second type of errors. However, improper punishment on "non-oracle but correct" sequence can be resulted sometimes because spurious ambiguity makes minimizing these two types of errors inconsistent. Figure 3 illustrates this.

Doppa *et al.* [2014a] studied structured prediction via output space search and proposed the $\mathcal{HC}$-search framework to solve the deficiency of a single scoring function in guiding search and simultaneously ranking generated outputs. Instead of learning a single scoring function, $\mathcal{HC}$-search decomposes the problem into three steps. Step 1 searches for an initial structured output from the input over the *primary space* [Doppa *et al.*, 2014b]; Step 2 explores a set of alternative outputs rooted at the initial output, which is a search process guided by the heuristic function $\mathcal{H}$ over the *output space*, named as the $\mathcal{H}$-step; Step 3 scores each output with a cost function $\mathcal{C}$ and selects the highest scoring one as final output, named as the $\mathcal{C}$-step. To learn $\mathcal{H}$ and $\mathcal{C}$, they decomposed the output space search error $\mathcal{E}_{\mathcal{HC}}$ into two parts: 1) *Generation error* $\epsilon_{\mathcal{H}}$ due to $\mathcal{H}$ not generating high-quality outputs and 2) *Selection error* $\epsilon_{\mathcal{C}|\mathcal{H}}$ due to $\mathcal{C}$ not selecting the best loss output. Given an instance (x, y^*), this error is described as

$$\mathcal{E}_{\mathcal{HC}} = \underbrace{L(x, y^*_{\mathcal{H}}, y^*)}_{\epsilon_{\mathcal{H}}} + \underbrace{L(x, \hat{y}, y^*) - L(x, y^*_{\mathcal{H}}, y^*)}_{\epsilon_{\mathcal{C}|\mathcal{H}}}, \quad (1)$$

where $y^*_{\mathcal{H}}$ is the best loss output of those $\mathcal{H}$-step returns. Based on this decomposition, Doppa *et al.* [2014a] proposed a stage-wised learning method by first training $\mathcal{H}$ to minimize $\epsilon_{\mathcal{H}}$, then training $\mathcal{C}$ to minimize $\epsilon_{\mathcal{C}|\mathcal{H}}$ conditioned on $\epsilon_{\mathcal{H}}$.

Our method is inspired by the $\mathcal{HC}$-search framework. We find the similarity between standard incremental parsing and $\mathcal{HC}$-search in term of loss decomposition. Following the $\mathcal{HC}$-search, we decompose the errors in standard incremental parsing into 1) *Generation error* for not generating the gold tree before search finishes, which consists with the first type of error and 2) *Selection error* for not selecting the best loss tree from the trees in final step beam, which consists with the second type of error. Different from the $\mathcal{HC}$-search which

Algorithm 3: Heuristic function learning algorithm.

```
Input: $\mathcal{D}$ = the training instances, $R$ = the max iteration.
Output: $\vec{w}$, The weight for the heuristic function.
$\vec{w}_0 \leftarrow \vec{0}$, $t \leftarrow 0$
for $r \leftarrow 1..R$ do
    for each training instance $(x, y^*) \in \mathcal{D}$ do
        $C_{0,m} \leftarrow$ ORACLEFINDING$(y^*)$
        $\mathcal{B}_0 \leftarrow \{c_0\}$
        for $j \leftarrow 1..m$ do
            $\mathcal{B}_j \leftarrow$ BEST$_k(x, \mathcal{B}_{j-1}, \vec{w}_t)$
            if $c_j \notin \mathcal{B}_j$ then
                $c'_j \leftarrow$ SELECTNEGATIVESAMPLE$(\mathcal{B}_j)$
                $\vec{w}_{t+1} \leftarrow$UPDATE$(C_{0..j}, C'_{0..j}, \vec{w}_t)$
                break
        $t \leftarrow t + 1$
return $\vec{w}$, which is average of $\vec{w}_0, .., \vec{w}_t$
```

performs heuristic search over the *output space*, we generate our candidate outputs by beam-search, which is guided by a heuristic function $\mathcal{H}$ and searches over the *primary space* (i.e. the space defined by the transition system). More specifically, outputs derived by the sequences in final step beam are collected as the candidate outputs for the $\mathcal{C}$-step. In the $\mathcal{C}$-step, a cost function $\mathcal{C}$ is adopted to select the best loss one as the final output. To learn our $\mathcal{H}$ and $\mathcal{C}$ functions, we follow Doppa *et al.* [2014a] and minimize $\epsilon_{\mathcal{H}}$ and $\epsilon_{\mathcal{H}|\mathcal{C}}$ in a stage-wised manner.

3.2 Learning Heuristic Function

We parameterize our heuristic function in the same way with standard incremental parsing as $\mathcal{H}(x, y) = \sum_{c \in C_{0..m}} \vec{w}\Phi(x, c)$ where y is derived by $C_{0..m}$, and $\mathcal{H}(x, y)$ can be trained similarly by imitating the oracle sequence. However, different from the standard incremental parsing, minimizing the learning objective in our $\mathcal{H}$-step means guiding the search towards the final step and uncovering the gold tree by scoring the oracle sequence higher than the sequences not in beam, but not necessarily scoring it highest. This difference indicates that the second type of error for learning standard incremental parsing model doesn't violate our $\mathcal{H}$-step learning objective because the gold tree is uncovered in this condition. To learn a better heuristic function, we modify the learning algorithm by omitting the second type of error and only performing corrective updates on the first one. Algorithm 3 outlines this learning algorithm, in which updates on the second type of error (line 12-14 in Algorithm 2) are omitted. By learning the heuristic function in such way, some improper updates like the case in Figure 3 can be voided.

In the learning algorithm for standard incremental parsing, making corrective updates with the "highest scoring incorrect" sequence as a negative sample $C'_{0..m}$ will learn a model that gives the oracle sequence $C_{0..m}$ the highest score. While, without the necessity of scoring $C_{0..m}$ highest in our $\mathcal{H}$-step, we are allowed to select any sequence in beam as a negative sample, because updates with any of these sequences $C'_{0..m}$ as a negative sample will enforce the model to score $C_{0..m}$ higher than $C'_{0..m}$ and uncover the gold tree derived by $C_{0..m}$

in the outputs. In Algorithm 3, we generalize the negative sample selection into a SELECTNEGATIVESAMPLE function (line 9). In this paper, we investigate two different strategies for selecting negative sample, which are 1) BEST by selecting the highest scoring sequence in beam and 2) WORST by selecting the lowest scoring sequence. Both these two strategies satisfy our learning objective, but lead to different models. Intuitively, the WORST strategy is more conservative than the BEST because it only encourages the model to score the oracle sequence higher than all the sequences not in beam.

3.3 Learning Cost Function

Given a set of $\mathcal{H}$-step outputs $\mathcal{Y}_{\mathcal{H}}(x)$, we want our cost function correctly rank the "best loss" output in the $\mathcal{C}$-step. We formulate the learning problem for the cost function as a *ranking* problem. More specifically, we want the "better loss" outputs to rank better than the "worse loss" ones, which is a bipartite ranking problem [Agarwal and Roth, 2005]. In general, our learning algorithm for the cost function firstly generates a collection of bipartite ranking examples, then learns parameters from these examples.

A widely used approach [Huang, 2008; Doppa *et al.*, 2014a] for generating the ranking examples is by collecting the "best loss" outputs from $\mathcal{Y}_{\mathcal{H}}(x)$ as $\mathcal{Y}_{best}$ and generating the examples as $(y^+, y^-) \in \mathcal{Y}_{best} \times \mathcal{Y}_{\mathcal{H}}(x) \setminus \mathcal{Y}_{best}$. Owing to that it categorizes the outputs into the "best loss" and "non-best loss" groups and only cares for the ranks between these two groups with omitting the ranks among the "non-best loss" outputs, we name it as COARSE approach.

To make use of ranks among the "non-best loss" outputs and refine the COARSE approach, we categorize the outputs into m groups $(\mathcal{Y}_{\mathcal{H},1}, .., \mathcal{Y}_{\mathcal{H},m})$ by their losses to the gold tree. Outputs in the same group share the same loss and higher ranked group has better loss than the lower ranked. Thus, $\mathcal{Y}_{\mathcal{H},1}$ is equivalent to $\mathcal{Y}_{best}$ in the COARSE approach. For the $j$the group $\mathcal{Y}_{\mathcal{H},j}$, we collect all its lower ranked outputs $\cup_{k=j+1}^{m}\mathcal{Y}_{\mathcal{H},k}$ and generate the ranking examples as $(y^+, y^-) \in \mathcal{Y}_{\mathcal{H},j} \times \cup_{k=j+1}^{m}\mathcal{Y}_{\mathcal{H},k}$. By generating examples with FINE grain approach, we allow our model to capture the sophisticated relation between outputs of different qualities.

In this paper, we parameterize our cost function as $\mathcal{C}(x, y) = \vec{w}\Phi(x, y)$, where $\Phi(x, y)$ extracts features from an output tree y at arbitrary order. For a pair-wised ranking example (y^+, y^-), we learn the parameters $\vec{w}$ by constraining $\mathcal{C}$ to score y^+ higher than y^-, i.e. $\mathcal{C}(x, y^+) > \mathcal{C}(x, y^-)$. Further, we require that $\mathcal{C}(x, y^+)$ is greater than $\mathcal{C}(x, y^-)$ by a margin $\Delta(y^+, y^-) = L(x, y^-, y^*) - L(x, y^+, y^*)$. By imposing the margin, we enforce our cost function to score the tree far from "best loss" tree even smaller.

Learning from every pair of ranking examples between "better loss" group $\mathcal{Y}^+$ and "worse loss" group $\mathcal{Y}^-$ can result in an over-constrained problem. In this case, we relax the constraint by enforcing $\mathcal{C}(x, y)$ to give the lowest-scored output in $\mathcal{Y}^+$ a higher score than the highest-scored output in $\mathcal{Y}^-$. Putting them together, we formulate our constraint as

$$\mathcal{C}(x, y^+) - \mathcal{C}(x, y^-) \geq \Delta(y^+, y^-),$$
$$\text{where } y^+ = \text{argmin}_{y \in \mathcal{Y}^+} \mathcal{C}(x, y)$$
$$y^- = \text{argmax}_{y \in \mathcal{Y}^-} \mathcal{C}(x, y)$$

Algorithm 4: Cost function learning algorithm.

```
   Input: $\mathcal{D}$ = the training instances, $R$ = the max iteration.
   Output: $\vec{w}$, The weight for the cost function.
1  $\mathcal{G} \leftarrow \emptyset$      ▷ $\mathcal{G}$, a set of ranking group pairs
2  for each training instance $(x, y^*, \mathcal{Y}_{\mathcal{H}}(x)) \in \mathcal{D}$ do
3      $\mathcal{Y}_{\mathcal{H},1}, .., \mathcal{Y}_{\mathcal{H},m} \leftarrow$ RANK$(x, y^*, \mathcal{Y}_{\mathcal{H}}(x))$
4      for $j \leftarrow 1..m-1$ do
5          $\mathcal{G} \leftarrow \mathcal{G} \cup (x, y^*, \mathcal{Y}_{\mathcal{H},j}, \cup_{k=j+1}^{m}\mathcal{Y}_{\mathcal{H},k})$
6  $\vec{w}_0 \leftarrow \vec{0}, t \leftarrow 0$
7  for $r \leftarrow 1..R$ do
8      for each ranking group pair $(x, y^*, \mathcal{Y}_{\mathcal{H}}^+, \mathcal{Y}_{\mathcal{H}}^-) \in \mathcal{G}$ do
9          $y^+ \leftarrow \text{argmin}_{y \in \mathcal{Y}_{\mathcal{H}}^+} \vec{w}_t\Phi(x, y)$
10         $y^- \leftarrow \text{argmax}_{y \in \mathcal{Y}_{\mathcal{H}}^-} \vec{w}_t\Phi(x, y)$
11         $\nabla f \leftarrow \vec{w}_t\Phi(x, y^+) - \vec{w}_t\Phi(x, y^-)$
12         $\Delta \leftarrow L(x^{(i)}, y^-, y^*) - L(x^{(i)}, y^+, y^*)$
13         if $\Delta \neq 0$ and $\nabla f < \Delta$ then
14             $\vec{w}_{t+1} \leftarrow$ UPDATEPA$(\nabla f, \Delta, \vec{w}_t)$
15         $t \leftarrow t + 1$
16 return $\vec{w}$, which is average of $\vec{w}_0, .., \vec{w}_t$
```

We outline our learning method for the cost function in Algorithm 4. Line 1-5 shows the ranking example generation approach. The categorizing part of COARSE and FINE approaches is generalized into a RANK function. Line 6-15 shows the model learning procedure. We use the online passive-aggressive algorithm [Crammer *et al.*, 2006] to learn the parameters $\vec{w}$. For each pair of bipartite ranking groups, lowest-scored "better loss" output y^+ and highest-scored "worst loss" output y^- are picked out (line 9, 10). If $\mathcal{C}(x, y)$ fails to score y^+ higher than the y^- by $\Delta(y^+, y^-)$, passive-aggressive updates are performed (line 14).

Cross Validation
We need to note that mis-matched output distribution between training and testing will be resulted if we generate the ranking examples for learning the cost function on the data we train our heuristic function. To mitigate this problem, ranking examples are generated via cross validation.

4 Experiment

4.1 Settings

We conduct our experiments on the Penn WSJ Treebank (PTB) for English and Chinese Treebank 5 (CTB5) for Chinese. Data are split into training, development and test set in the same way with the previous study [Zhang and Nivre, 2011]. Bracket sentences in PTB are converted to dependency formats with Penn2Malt.[2] For those in CTB, head-finding rules from Zhang and Nivre [2011] are used. For the

[2]http://stp.lingfil.uu.se/~nivre/research/Penn2Malt.html

Parser	PTB			CTB5		
	Dev	Test	SPD	Dev	Test	SPD
BASELINE	92.95	92.48	1x	86.76	86.44	1x
BEST+FINE	**93.13**	**92.76** (+0.28)	1.25x	87.25	87.04 (+0.60)	1.08x
BEST+COARSE	92.94	92.44 (-0.04)	1.30x	86.61	86.51 (+0.07)	1.07x
WORST+FINE	93.12	92.73 (+0.25)	1.33x	**87.27**	**87.15** (+0.71)	1.22x
WORST+COARSE	92.89	92.47 (-0.01)	1.30x	86.95	86.82 (+0.38)	1.20x
BASELINE+FINE	93.06	92.53 (+0.05)		87.07	86.70 (+0.26)	

Table 1: Experimental results on PTB and CTB5. The *Dev* and *Test* show the results on development and test set respectively. The *SPD* shows the relative parsing time compared to the BASELINE. BEST and WORST represents the SELECTNEGATIVESAMPLE strategy in the $\mathcal{H}$-step. COARSE and FINE represents the ranking example generation approach in the $\mathcal{C}$-step.

English data, automatically assigned POS tags are obtained using 10-fold jackknifing with an accuracy of 97.17, 97.19 and 97.31 on training, development and test set. For the Chinese data, gold segmentation and POS tags are used. Parsing performance is measured by unlabeled attachment score (UAS) excluding punctuations.[3]

We set the beam size to 64 for both training and testing in the $\mathcal{H}$-step, thus, the size of the candidate outputs is 64 for each instance in the $\mathcal{C}$-step. During training our cost function with cross validation, we set the number of folds as 20. Best iterations for learning our heuristic and cost functions are tuned on the development set. Hamming distance is used as loss function L throughout this paper.

In the $\mathcal{H}$-step, we adopt the feature templates used in Zhang and Nivre [2011] to capture information on each search state. Zhang *et al.* [2014] proposed rich global features for their sampling-based parser and we adopt their feature templates in the $\mathcal{C}$-step. Details for the feature templates can be referred in the supplemental materials.

4.2 Baseline

Our BASELINE parser is a standard incremental parser with the same feature templates and beam size to our $\mathcal{H}$-step. On the PTB and CTB5 test set, our BASELINE parser achieves UAS scores of 92.48 and 86.44 respectively. Since the PTB score is higher than its *transition-based* counterparts in Table 4, we can confirm our BASELINE parser as a strong one.

4.3 Results

Table 1 shows our results. We report the performances of our parsers combining of different negative sample selection strategies in learning the heuristic function and different ranking example generation approaches in learning the cost function. Our best-setting parsers outperform the BASELINE by 0.28 on PTB and 0.71 on CTB5 test data and the error reductions are 3.63% and 5.21% respectively. Significance test shows the improvements are statistically significant with $p < 0.01$.

A natural question raises that "Can we achieve similar improvement by adding an additional ranking step (i.e. the $\mathcal{C}$-step) over our BASELINE?" We replace the heuristic function in the BEST+FINE parser with our BASELINE and perform the same ranking process on the BASELINE outputs. We show the result in the last block of Table 1. From this result, improvements over the BASELINE are observed, but they are of a smaller margin than the best-setting parsers.

[3]Following Chen and Manning [2014], a token is a punctuation if its gold POS tag is {" " : , .} for English and PU for Chinese.

Parser	PTB			CTB5		
	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\vert\mathcal{H}}$	$\mathcal{E}_{\mathcal{HC}}$	$\epsilon_{\mathcal{H}}$	$\epsilon_{\mathcal{C}\vert\mathcal{H}}$	$\mathcal{E}_{\mathcal{HC}}$
BEST+FINE	3.69	**3.90**	**6.87**	8.77	**5.72**	12.75
BEST+COARSE		4.14	7.06		6.93	13.39
WORST+FINE	**3.05**	4.62	6.88	**7.75**	7.33	**12.73**
WORST+COARSE		5.09	7.11		7.58	13.05
BASELINE+FINE	3.70	4.10	6.94	8.81	6.27	12.93

Table 2: Error decomposition of the heuristic and cost functions in different parsers on the development set.

4.4 Loss Decomposition Analysis

We analyze the decomposed loss (the *Generation loss* $\epsilon_{\mathcal{H}}$ and the *Selection loss* $\epsilon_{\mathcal{C}|\mathcal{H}}$) on the development set to investigate the reason for improvements. Given a set of $\mathcal{H}$-step outputs $\mathcal{Y}_{\mathcal{H}}(x)$, according to Equation 1, $\epsilon_{\mathcal{H}}$ is resulted by the heuristic function losing gold tree and is measured by $L(x, y^*_{\mathcal{H}}, y^*)$, which is the number of errors between the best loss output $y^*_{\mathcal{H}}$ in $\mathcal{Y}_{\mathcal{H}}(x)$ and the gold tree y^*. $\epsilon_{\mathcal{C}|\mathcal{H}}$ is resulted by the cost function not selecting the best loss output and is measured by the average number of errors between the final output $\hat{y}$ and the best loss outputs $\mathcal{Y}_{best}$, which is $\sum_{y \in \mathcal{Y}_{best}} L(x, \hat{y}, y) \,/\, ||\mathcal{Y}_{best}||$.

Table 2 shows the loss decomposition analysis results. For the $\mathcal{H}$-step, the parsers with relaxed learning objective (BEST, WORST) generally achieve smaller $\epsilon_{\mathcal{H}}$ than the BASELINE. And the parsers with conservative SELECTNEGATIVESAMPLE strategy (i.e. the WORST) achieve consistently smaller loss than those with aggressive strategy (i.e. the BEST). This table shows the relaxed learning objective is more helpful for uncovering high-quality outputs in the $\mathcal{H}$-step. We attribute the smaller loss to the fact that the relaxed learning objective mitigates improper updates resulted from search ambiguities. Our observation of the conservative strategy achieving better $\epsilon_{\mathcal{H}}$ also confirms that. For the $\mathcal{C}$-step, the parsers with FINE grain approach consistently achieve better $\epsilon_{\mathcal{C}|\mathcal{H}}$ than those with COARSE approach. This shows the necessity to learn from fine grain ranking examples.

In Table 2, we can see that a smaller $\epsilon_{\mathcal{H}}$ always couples with a larger $\epsilon_{\mathcal{C}|\mathcal{H}}$. It's difficult to confirm which step contributes more to the final performance because our $\mathcal{C}$-step error is effected by the $\mathcal{H}$-step outputs. But by comparing

Parser	*non-mixture*	*mixture*
BASELINE	92.48	
BASELINE+FINE	92.53	92.94
BEST+FINE	92.76	93.02
WORST+FINE	92.73	93.05

Table 3: Comparison of the *non-mixture* and *mixture* parsers on the PTB test data.

Parser	UAS
Transition-based	
Chen and Manning [2014]	92.00
Huang and Sagae [2010]	92.10
Our BASELINE	92.48
Dual-Decomposition	
Martins *et al.* [2011]	93.07
Mixture-model	
Le and Zuidema [2014]	93.12
Hayashi *et al.* [2013]	93.12
Our Mixture WORST+FINE	93.05

Table 4: Comparison with the state-of-the-art parsers on the PTB test data.

the BEST+FINE, WORST+FINE with BASELINE+FINE, the parsers trained with relaxed learning objective achieve better overall performance, which indicates the importance to learn the model according to the loss decomposition rather than minimizing $\epsilon_{\mathcal{C}|\mathcal{H}}$ on both steps like the BASELINE+FINE.

4.5 Mixing $\mathcal{H}$ and $\mathcal{C}$ functions

Following many approaches that mix scores from two stages [Hayashi *et al.*, 2013; Le and Zuidema, 2014], we define our *mixture* parser as a combination of our $\mathcal{H}$ and $\mathcal{C}$ functions with a linear interpolation, i.e. $\hat{y} = \text{argmax}_{y \in \mathcal{Y}_{\mathcal{H}}(x)} \alpha \mathcal{H}'(x,y) + (1-\alpha)\mathcal{C}'(x,y)$, where $\mathcal{H}'(x,y)$ and $\mathcal{C}'(x,y)$ are the rescaled scores and $\alpha \in [0,1]$ is a hyper parameter tuned on the development set. [4] We compare our mixture and non-mixture parsers in Table 3. Our mixture parsers outperforms the non-mixture counterparts by 0.26 and 0.32 on BEST+FINE and WORST+FINE. The gains in UAS score suggest that our method can be further improved by considering the quality of heuristic search in the final decision. Also, our BEST+FINE and WORST+FINE mixture parsers perform better than the BASELINE+FINE mixture parser, which further shows the effectiveness of the relaxed learning objective in the $\mathcal{H}$-step and the necessity to learn the model based on the decomposed loss.

Finally, we compare our best-setting parser with the state-of-the-art parsers in Table 4. Our mixture parser gets superior accuracy than the transition-based parsers and achieves comparable performance with the state-of-the-art dual-decomposition and mixture-model parsers.

[4]In this paper, we rescale the $\mathcal{H}(x,y)$ and $\mathcal{C}(x,y)$ over $\mathcal{Y}_{\mathcal{H}}(x)$ as $\mathcal{F}'(x,y) = \frac{\mathcal{F}(x,y)-\min_{y'}\mathcal{F}(x,y')}{\max_{y'}\mathcal{F}(x,y')-\min_{y'}(x,y')}$. α is 0.395 for BASELINE+FINE, 0.210 for BEST+FINE and 0.225 for WORST+FINE in our experiments.

4.6 Parsing Speed

One advantage of incremental parsing lies on the property that parsing is bounded to linear time. Outputting k-best candidates can be done without increasing the time-complexity compared to the exact search algorithm based on dynamic programming [McDonald and Pereira, 2006]. In our approach, only an additional ranking model is imposed to score and select the best loss output from a fix-sized set of outputs, thus, it maintains the linear complexity. Empirically, we show the relative time consumption of our $\mathcal{HC}$-search parsers compared with the BASELINE in Table 1. This comparison shows that our parsers can perform accurate parsing without sacrificing the speed.

5 Related Work

Previous studies on standard incremental parsing mainly focus on proposing new transition systems, speeding up decoding and incorporating global features [Huang and Sagae, 2010; Zhang and Nivre, 2011; Bohnet and Nivre, 2012]. In this paper, we study the model learning in stage-wised manner based on loss decomposition.

Two-stage parsing methods are widely adopted. One genre of such work is performing reranking on a k-best list [Collins and Duffy, 2002; Charniak and Johnson, 2005; Huang, 2008; Le and Zuidema, 2014]. Another genre is doing a second-time decoding with the first stage output as features [Hayashi *et al.*, 2013]. Our method can be viewed as a reranking-styled method. However we analyzed the theoretical intuition for our two-stage method and learn our model following loss decomposition, which shows superior to the model without following such decomposition. What's more, we learn the ranking model from fine grain ranking examples which is less studied in previous works.

Zhang *et al.* [2014] proposed a sampling-based inference method for dependency parsing, which can be considered as an instance of the output space search. Our work resembles theirs by incorporating global features in the $\mathcal{C}$-step, but differs in both decoding and model learning.

In this paper, we base our method on the framework of $\mathcal{HC}$-search [Doppa *et al.*, 2014a], but we employ a different $\mathcal{H}$-step. In their work, heuristic search is performed on the *output space* while ours is performed on the *primary space* (i.e. the space defined by the transition system). We show that primary space search can also be effective for generating high-quality outputs.

6 Conclusion

In this paper, we proposed a new approach for incremental parsing based on the loss decomposition of $\mathcal{HC}$-search framework. In the $\mathcal{H}$-step, we uncover high-quality candidate outputs with beam-search. In the $\mathcal{C}$-step, we select the best loss output with a ranking model. We learn the $\mathcal{H}$-step model with a relaxed objective and the $\mathcal{C}$-step model from fine grain ranking examples. Experimental results show our parser outperforms a strong baseline. Further improvements are achieved when combining two steps together.

Acknowledgments

This work was supported by the National Key Basic Research Program of China via grant 2014CB340503 and the National Natural Science Foundation of China (NSFC) via grant 61133012 and 61370164.

References

[Agarwal and Roth, 2005] Shivani Agarwal and Dan Roth. Learnability of bipartite ranking functions. In Peter Auer and Ron Meir, editors, *Learning Theory*, volume 3559 of *Lecture Notes in Computer Science*, pages 16–31. Springer Berlin Heidelberg, 2005.

[Bohnet and Nivre, 2012] Bernd Bohnet and Joakim Nivre. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *EMNLP-2012*, pages 1455–1465, Jeju Island, Korea, July 2012. ACL.

[Charniak and Johnson, 2005] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL-2005*, pages 173–180, Ann Arbor, Michigan, June 2005. ACL.

[Chen and Manning, 2014] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *EMNLP-2014*, pages 740–750, Doha, Qatar, October 2014. ACL.

[Collins and Duffy, 2002] Michael Collins and Nigel Duffy. Convolution kernels for natural language. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS*, pages 625–632. MIT Press, 2002.

[Collins and Roark, 2004] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *ACL-2004*, pages 111–118, Barcelona, Spain, July 2004.

[Collins, 2002] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *EMNLP-2002*, pages 1–8. ACL, July 2002.

[Crammer *et al.*, 2006] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, December 2006.

[Daumé *et al.*, 2009] Hal Daumé, Iii, John Langford, and Daniel Marcu. Search-based structured prediction. *Mach. Learn.*, 75(3):297–325, June 2009.

[Doppa *et al.*, 2014a] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Hc-search: A learning framework for search-based structured prediction. *J. Artif. Intell. Res. (JAIR)*, 50:369–407, 2014.

[Doppa *et al.*, 2014b] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured prediction via output space search. *J. Mach. Learn. Res.*, 15(1):1317–1350, January 2014.

[Goldberg and Nivre, 2012] Yoav Goldberg and Joakim Nivre. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India, December 2012. The COLING 2012 Organizing Committee.

[Hayashi *et al.*, 2013] Katsuhiko Hayashi, Shuhei Kondo, and Yuji Matsumoto. Efficient stacked dependency parsing by forest reranking. *Transactions of the Association for Computational Linguistics*, 1:139–150, 2013.

[Huang and Sagae, 2010] Liang Huang and Kenji Sagae. Dynamic programming for linear-time incremental parsing. In *ACL-2010*, pages 1077–1086, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[Huang *et al.*, 2012] Liang Huang, Suphan Fayong, and Yang Guo. Structured perceptron with inexact search. In *NAACL-2012*, pages 142–151, Montréal, Canada, June 2012. ACL.

[Huang, 2008] Liang Huang. Forest reranking: Discriminative parsing with non-local features. In *ACL-2008*, pages 586–594, Columbus, Ohio, June 2008. ACL.

[Le and Zuidema, 2014] Phong Le and Willem Zuidema. The inside-outside recursive neural network model for dependency parsing. In *EMNLP-2014*, pages 729–739, Doha, Qatar, October 2014. ACL.

[Martins *et al.*, 2010] Andre Martins, Noah Smith, Eric Xing, Pedro Aguiar, and Mario Figueiredo. Turbo parsers: Dependency parsing by approximate variational inference. In *EMNLP-2010*, pages 34–44, Cambridge, MA, October 2010. ACL.

[Martins *et al.*, 2011] Andre Martins, Noah Smith, Mario Figueiredo, and Pedro Aguiar. Dual decomposition with many overlapping components. In *EMNLP-2011*, pages 238–249, Edinburgh, Scotland, UK., July 2011. ACL.

[McDonald and Pereira, 2006] Ryan T McDonald and Fernando CN Pereira. Online learning of approximate dependency parsing algorithms. In *EACL*, 2006.

[Nivre, 2008] Joakim Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553, 2008.

[Zhang and Nivre, 2011] Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *ACL-2011*, pages 188–193, Portland, Oregon, USA, June 2011. ACL.

[Zhang *et al.*, 2014] Yuan Zhang, Tao Lei, Regina Barzilay, Tommi Jaakkola, and Amir Globerson. Steps to excellence: Simple inference with refined scoring of dependency trees. In *ACL-2014*, pages 197–207, Baltimore, Maryland, June 2014. ACL.