Chapter 6
Hybrid Dependency Parser with Segmented Treebanks and Reparsing

Fuxiang Wu and Fugen Zhou

Abstract We propose a hybrid dependency parsing pipeline which combines transition-based parser and graph-based parser, and use segmented treebanks to train transition-based parsers as subparsers in front end, and then propose a constrained Eisner's algorithm to reparse their outputs. We build the pipeline to investigate the influence on parsing accuracy when training with different segmentations of training data and find a convenient method to obtain parsing reliability score while achieving state-of-the-art parsing accuracy. Our results show that the pipeline with segmented training dataset could improve accuracy through reparsing while providing parsing reliability score.

Keywords Hybrid dependency parsing · Constrained Eisner's algorithm · Parsing reliability score · Transition-based parser · Graph-based parser

6.1 Introduction

A good amount of research has been devoted to parsing technology, due to the importance of dependency parsing, and many natural language applications, such as information retrieval and Q&A system [10], employing it as a base component, and their performance may highly rely on the parsing result.

Recent methods of dependency parsing can be divided into two classes: data-driven methods and rule-based methods. Data-driven methods usually are statistical parser and use some machine learning algorithms to catch the statistical features of data in order to produce syntactic relations of words in sentences.

F. Wu (✉) · F. Zhou
Image Processing Center, Beihang University, Beihang, China
e-mail: fxwuedu@buaa.edu.cn

F. Zhou
e-mail: zhfugen@buaa.edu.cn

Z. Deng and H. Li (eds.), *Proceedings of the 2015 Chinese Intelligent Automation Conference*, Lecture Notes in Electrical Engineering 336,
DOI 10.1007/978-3-662-46469-4_6

Most of the state-of-the-art parsers are statistical parser, parsing accuracy of which highly relies on the quality and quantity of treebank [3, 6, 11]. There are several treebanks in China for syntactic parsing, such as Penn Chinese Treebank (CTB) [8] and Chinese Dependency Treebank (CDT). The CTB is constituency annotation and was retrieved from Xinhua Newswire, Hong Kong news, Sinorama and ACE broadcast news, while the CDT is a dependency treebank which was retrieved from People's Daily newswire stories. Occurrence frequency of syntactic substructure (subtree) in one treebank may vary from another. Some would frequently occur in the treebank but others would not. The rare syntactic substructures of some sentences in the treebank are well formed for human, but would be abandoned when there are some common syntactic substructures which are conflict with the rare ones and prevent the parser from handling rare syntactic structures correctly. This would lead to label attachment recall rate degradation partly.

In order to examine the assumption, we segment treebank into k-parts in one round, and train a transition-based parser (subparser) for each part, and then a result set generated by the subparsers is compared with *k*-best generated by parser trained with full treebank. We learn that label attachment recall rate of k-segments would be higher than corresponding *k*-best's, and this confirms the assumption. Depending on this phenomenon, we further employ a parser to post-reparse the subparsers output. Since transition-based parser and graph-based parser have different training and inference algorithms [5, 7] and have different behaviors, we construct the post-reparser with constrained Eisner's algorithm [2, 4] to find maximum spanning trees (MST). The experiment shows that the pipeline could improve the parsing accuracy while computing the parsing reliability score.

6.2 Parsing Pipeline

The pipeline in this paper addresses the general structural prediction problem, which map an input sentence $x \in \mathbf{X}$ to an output dependency structure $y \in \mathbf{Y}$, which is composed of edge e,

$$e = \begin{cases} \langle i, j, \leftarrow, l \rangle \\ \langle i, j, \rightarrow, l \rangle \end{cases} \tag{6.1}$$

where i and j are relation endpoints, l is dependent label in label set $\mathbf{L}$. We employ transition-based parser with beam search [9] as subparser and use MST parser with conditional random field as post-reparser. In CRF model, the output y probability would be,

$$p(y|x) = \exp(f(y, x) \cdot \boldsymbol{\lambda}) / Z(x) \tag{6.2}$$

where $f(y,x)$ maps y and x to a feature vector, $\boldsymbol{\lambda}$ is a corresponding weight vector, and $Z(x)$ is the normalization factor. For a sentence x, the parsing result y is calculated by finding the highest probability one among the all possible results,

$$O(x) = \arg\max_{y \in \mathbf{PSET}(x)} p(y|x) \tag{6.3}$$

where $\mathbf{PSET}(x)$ denotes the set of the possible result for the sentence x.

The pipeline is composed of a training procedure and a parsing procedure. The training procedure mainly creates a set of subparsers which are transition-based parser. And in parsing procedure, we first use this set of subparsers to achieve a result set, and then reparse the set to compute the best output.

6.2.1 Training

The training procedure is as follows:

- Segmenting treebank into equally sized sub-treebanks $\boldsymbol{\Omega} = \{b_i\}_{i=1,\ldots,N}$
- Training subparser t_i with sub-treebank $b_i \in \Omega$ to build a set of subparsers $\boldsymbol{\Gamma}_N = \{t_i\}_{i=1,\cdots,N}$.
- Training the MST parser T with the whole treebank to calculate the weight vector $\boldsymbol{\lambda}_T$ for features.

Through this procedure, we get a trained model $\{N, \boldsymbol{\Gamma}_N, \boldsymbol{\lambda}_T, f_T\}$, where N is the number of subparsers trained by segmented treebanks; f_T is a feature extract function of MST parser, built by feature temples.

6.2.2 Post-reparsing

In parsing step, a set of result $\mathbf{R} = \{r_i\}_{i=1,\ldots,N}$, which is different from N-best result, have been generated by subparser in $\boldsymbol{\Gamma}_N$ for an input sentence x. We use them to constrain the searching space for the sentence (far small than the full searching space), and then employ constrained Eisner's algorithm to extract the best result. The constraint scores are obtained as follows:

$$s_c(e, \mathbf{R}) = \begin{cases} f_T(e) \cdot \boldsymbol{\lambda}_T & \text{if} \quad e \in r_i, i = 1, \ldots, N \\ 0 & \text{else} \end{cases} \tag{6.4}$$

where $f_T(e)$ maps edge e to a feature vector. And mixture score is as follows:

$$s_{\text{mixc}}(e, \mathbf{R}, \alpha) = \alpha \cdot f_T(e) \cdot \boldsymbol{\lambda}_T + (1 - \alpha) \cdot s_c(e, \mathbf{R}) \tag{6.5}$$

Table 6.1 Pseudo-code for constrained Eisner's algorithm

Allocate $E[N][N][2][2][N_L]$ array for edge's scores
Initialize each score in the array to zero
for i in $[1,\cdots,N]$
  for j in $[1,\cdots,N]$
    $t = i + j$
    if $t > N$ then break
    $E[i][j][\leftarrow][1][l] = \max_{i \le k < j}(\max_{u \in L} E[i][k][\rightarrow][0][u] +$
      $\max_{u \in L} E[k][j][\leftarrow][0][u] + s_{mixc}(\langle i, j, \leftarrow, l\rangle, \mathbf{R}, \alpha)$
    $E[i][j][\rightarrow][1][l] = \max_{i \le k < j}(\max_{u \in L} E[i][k][\rightarrow][0][u] +$
      $\max_{u \in L} E[k][j][\leftarrow][0][u] + s_{mixc}(\langle i, j, \rightarrow, l\rangle, \mathbf{R}, \alpha)$
    $E[i][j][\leftarrow][0][l] = \max_{i \le k < j}(\max_{u \in L} E[i][k][\leftarrow][0][u] + \max_{u \in L} E[k][j][\leftarrow][1][u]$
    $E[i][j][\rightarrow][0][l] = \max_{i \le k < j}(\max_{u \in L} E[i][k][\rightarrow][1][u] + \max_{u \in L} E[k][j][\rightarrow][0][u]$
Where N_L is the size of label set L.

where α is a mixture factor which controls the strength of constraint from the result set, given a sentence $S = w_0 w_1 \cdots w_N$ and the corresponding $\mathbf{R}$. The post-reparsing procedure is as follows (Table 6.1).

6.2.3 Reliability Score of Dependency Relation

With the pipeline, we can get a set of subparsers $\mathbf{\Gamma}_N$, in which each subparser is trained by different parts of training corpora. Because each part of corpora can be seen as unseen data from other part, we can assume that the parsing result of each subparser for a sentence is supported by the corresponding part of training corpora. Thus, reliability score can be calculated like a weighted voting scheme [1] as follows:

$$c(e_i) = \sum_{\varepsilon \in \mathbf{E}_i, \varepsilon = e_i} \exp(\xi \cdot f_T(\varepsilon) \cdot \boldsymbol{\lambda}_T) \Big/ \sum_{\varepsilon \in R_i} \exp(\xi \cdot f_T(\varepsilon) \cdot \boldsymbol{\lambda}_T) \tag{6.6}$$

where $\mathbf{E}_i$ is a set of dependency relationships $\langle i, \cdot, \cdot, \cdot\rangle$, which is ith relationship of dependency structure in set $\mathbf{R}$, ξ is an adjusting factor, and when $\xi = 0$, $c(e_i)$ is normal voting score for edge e_i.

Table 6.2 The training, development, and test data for CTB6

	File index	Sentences
Training	1–1129; 2019–2923	24,092
Dev	2924–3012; 3108–3145	1191
Test	1130–1151; 2000–2018; 3013–3107	2846

Table 6.3 The test results of the baseline parsers

	ZPar	MSTParser1	MSTParser2	crfParser
LAS	0.824361	0.775656	0.763978	0.782913
UAS	0.83939	0.81595	0.82032	0.8024

6.3 Baseline and Experiments

This section presents the pipeline experiments of segmentation and post-reparsing. Before this, we only evaluate the pipeline with Chinese Penn Treebank corpora as heavy computation cost for the CRF training without loss of generality. We split sentences in the Penn Treebank 6.0 into training, development, and test set as Table 6.2, and then employ the head-finding rule to translate them into dependency structures.

Baseline parsers are ZPar[1] dependency parser, MSTParser[2], and crfParser,[3] which are open source projects and have achieved state-of-the-art accuracy. They are trained with the training data in Table 6.2, and use their default feature temple, respectively. The test results are as follows (Table 6.3).

where MSTParser1 and crfParser are first order graph-based parsers, MSTParser is second order parser, and ZPar is transition-based dependency parser.

In order to explore the phenomenon brought by corpora segmentation, we segment the training data into parts with different number, namely, $\mathbf{\Theta} = [2, 3, 4, 6, 12]$, and then build the set of subparser $\mathbf{\Gamma}_K$ for each $k \in \mathbf{\Theta}$.

6.3.1 Attachment Recall Rate

Labeled/unlabeled attachment recall rate (LAR/UAR) is the ratio of correct labeled/unlabeled attachment among the dependency structure of result set,

[1] http://sourceforge.net/projects/zpar/.

[2] http://sourceforge.net/projects/mstparser/.

[3] http://sourceforge.net/projects/crfparser/.

$$\begin{cases} \mathrm{LAR}(\Pi) = \sum\limits_{(R_K, r_c) \in \Pi} \delta_L(\mathbf{R}_K, r_c) \Big/ \sum_{(\mathbf{R}_K, r_c) \in \Pi} |r_c| \\ \mathrm{UAR}(\Pi) = \sum\limits_{(R_K, r_c) \in \Pi} \delta_U(\mathbf{R}_K, r_c) \Big/ \sum_{(\mathbf{R}_K, r_c) \in \Pi} |r_c| \end{cases}$$

where $\mathbf{R}_K$ and r_c are a set of parsing result and gold dependency structure for a sentence, $\mathbf{\Pi}$ generated from test data is a set of $(\mathbf{R}_K, r_c)$, $|r_c|$ is the number of relationship in dependency r_c, function $\delta_L(\mathbf{R}_k, r_c)$ counts the correct dependency relationships with label in r_c which coexist in result set $\mathbf{R}_k$, and function δ_U counts similarly without label.

We calculate LAR and UAR for the k-segment's result set R_K and baseline parser's k-best result. The relationship between number of parts and attachment recall rate is as follows.

With data segmentation, we can achieve higher LAR and UAR then k-best parsing result, this means that the dataset $\mathbf{R}_K$ would cover more correct dependency relationship than the k-best dataset. It would be beneficial to postprocessing in the pipeline, such as reparsing and reranking, with small searching space.

6.3.2 Post-reparsing

In post-reparsing state, we analyze the result set generated by the set of subparser $\mathbf{\Gamma}_N$ or the N-best result made by baseline parser to get final parsing result, and their accuracy is as follows.

For each $k \in \mathbf{\Theta}$, we search the best $\alpha \in [0, 1]$ for calculating the LAS/UAS of each k-segment's or k-best result. The highest LAS of 2-segmentation is 83.2112, and is 0.7751 % higher than the ZPar in baseline parsers, and 2-best's is 83.1275 %. From Fig. 6.2, we could see that the LAS of k-segment and k-best is lower than the ZPar's when $k > 2$, meanwhile, k-segment's LAS is lower than k-best's. The reason may be the postparser, which is first-order minimum spanning tree parser with local features, is not powerful enough to utilize the higher label attachment recall rate. That is why 2-segment's LAS is higher than 2-best's. From Fig. 6.1, we could find that k-segment's LAR ascends faster than k-best's, and the k-best's LAS descend slower than k-segment's since $k > 3$ in Fig. 6.2, this is also shown that the post-parser needs a finer design. Besides, we employ reliability score $c(e_i)$ to rerank the result set $\mathbf{R}_K$, the result is as follows:

From Fig. 6.3, we can find that using reliability score $c(\boldsymbol{e}_i)$ to directly select dependency relationship is feasible. Their LAS/UAS are higher than each element in k-segment's result set, but their output may be not a tree, and need further process. The LAS/UAS of reranking result is lower than the baseline ZPar, this may also due to weakness of the postparser as well.

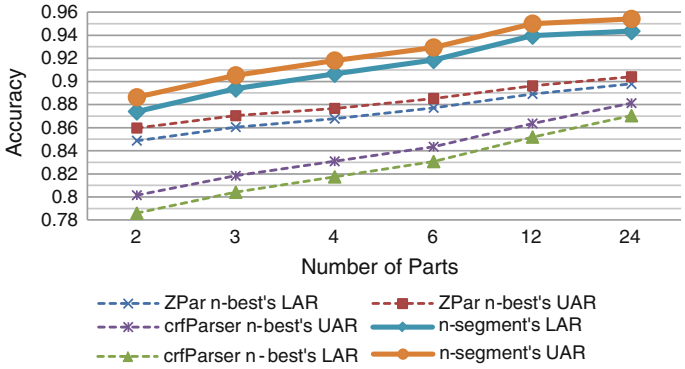

Fig. 6.1 The labeled/unlabeled attachment recall rate (LAR/UAR)—number of parts curve

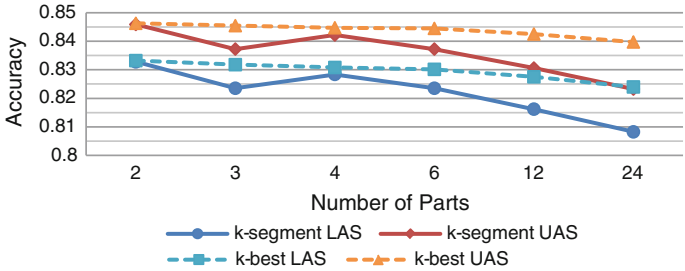

Fig. 6.2 The LAS/UAS of reparsing for *k*-segment's and *k*-best result set

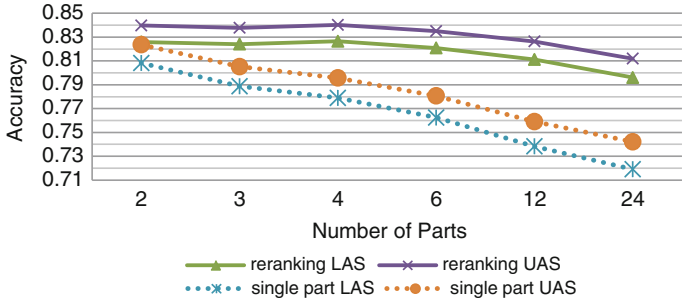

Fig. 6.3 The LAS/UAS of reranking for *k*-segment's result set. The dot line is the maximum LAS/UAS of element in *k*-segment's set

6.4 Conclusions and Future Work

We build a hybrid parsing pipeline, which employs transition-based dependency parser as subparser in front end, and then use graph-based dependency parser in next stage. Finally, we investigate the influence on the pipeline with different *k*-segment dataset. From the experiment, we found that using segmentation of training data would largely improve the labeled/unlabeled attachment recall rate with some final LAS/UAS drop, and the result set generated by the subparsers with high attachment recall rate could be used to calculate reliability score, such as simple voting scheme used in this paper. Besides, the hybrid pipeline could improve the final LAS/UAS when using 2-segment. But it cannot further improve the accuracy due to weak postparser. In future, we would try to use more sophisticated parser as postparser to explore the searching space constructed by the subparsers effectively.

References

1. Collins M (2002) Ranking algorithms for named-entity extraction: boosting and the voted perceptron. In: Proceedings of the 40th annual meeting on association for computational linguistics (ACL' 02), pp 489–496
2. Eisner J (1996) Three new probabilistic models for dependency parsing: an exploration. In: Proceedings of the 16th international conference on computational linguistics (COLING-96), pp 340–345
3. Li ZH, Liu T, Che WX (2012) Exploiting multiple treebanks for parsing with quasi-synchronous grammars. In: Proceedings of the 50th annual meeting of the association for computational linguistics (ACL' 12), pp 675–684
4. McDonald R, Pereira F (2006) Online learning of approximate dependency parsing algorithms. In: Proceedings of the 11th international conference of the European chapter of the association for computational linguistics (EACL 2006), pp 81–88
5. Nivre J, McDonald R (2008) Integrating graph-based and transition-based dependency parsers. In: Proceedings of the 46th annual meeting of the association for computational linguistics, pp 950–958
6. Niu ZY, Wang HF, Wu H (2009) Exploiting heterogeneous treebanks for parsing. 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, pp 46–54
7. Plank B, Noord GV (2010) Grammar-driven versus data-driven: which parsing system is more affected by domain shifts? In: Proceedings of the 2010 workshop on NLP and linguistics: finding the common ground (NLPLING' 10), pp 25–33
8. Xue NW, Xia F, Chiou FD, Palmer M (2005) The Penn Chinese treebank: phrase structure annotation of a large corpus. Nat Lang Eng 11(2):207–238
9. Zhang Y, Clark S (2011) Syntactic processing using the generalized perceptron and beam search. Comput Linguist 37(1):105–151
10. Zhou GY, Cai L, Zhao J, Liu K (2011) Phrase-based translation model for question retrieval in community question answer archives. In: Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies (HLT' 11), pp 653–662
11. Zhou GY, Zhao J (2013) Joint inference for heterogeneous dependency parsing. In: The 51st annual meeting of the association for computational linguistics, pp 104–109