A subtree-based factorization of dependency parsing

Qiuye Zhao[1] and **Qun Liu**[2,1]
1. Key Laboratory of Intelligent Information Processing,
Institute of Computing Technology, Chinese Academy of Sciences
2. ADAPT Centre, School of Computing, Dublin City University

Abstract

We propose a dependency parsing pipeline, in which the parsing of long-distance projections and localized dependencies are explicitly decomposed at the input level. A chosen baseline dependency parsing model performs only on 'carved' sequences at the second stage, which are transformed from coarse constituent parsing outputs at the first stage. When k-best constituent parsing outputs are kept, a third-stage is required to search for an optimal combination of the overlapped dependency subtrees. In this sense, our dependency model is subtree-factored. We explore alternative approaches for scoring subtrees, including feature-based models as well as continuous representations. The search for optimal subset to combine is formulated as an ILP problem. This framework especially benefits the models poor on long sentences, generally improving baselines by 0.75-1.28 (UAS) on English, achieving comparable performance with high-order models but faster. For Chinese, the most notable increase is as high as 3.63 (UAS) when the proposed framework is applied to first-order parsing models.

1 Introduction

Incorporating 'non-local' features into syntactic parsing has been well-studied in literature. For exact parsing, we have seen cubic-time decoders for first and second-order models (Eisner, 1996; McDonald and Pereira, 2006), quadratic-time decoders for third-order models (Koo and Collins, 2010) and etc. In transition-based parsers, e.g. (Nivre et al., 2006), so-called non-local features can be easily extracted from parsing history. With a global inference framework, e.g. approximate Linear Programming (Martins et al., 2011; Koo et al., 2010), arbitrary structural constraints can be imposed. There are two main research goals underlying these works, one is to localize long-distance dependencies, which can be achieved by assuming the optimal substructure property or introducing parsing actions like reductions. The other goal is to appropriately factor tree score, over parts like arcs as well as over parsing actions.

In this work, we propose a dependency parsing pipeline, so that long-distance dependencies are explicitly localized at the input level. With the proposed factorization, dependency tree score sums over its subtrees. More specifically, we address a distinction between long-distance projections and localized dependencies, which can be characterized by word categories of their lexical heads. The long-distance projections can be captured by a coarse constituent parser, which only sees peripheral and head words of such long-distance projections. So as to reduce error propagation, we transform k-best constituent parsing outputs to 'carved' sequences for the following dependency parsing, which could be overlapped. Therefore, a third stage is required to search for the optimal subset of all candidate subtrees to combine.

Given previous work on parsing, e.g. (Charniak and Johnson, 2005),(McDonald et al., 2005),(Martins et al., 2013) and so on, reliable constituent-based parsers and dependency parsers are available. Our main implementation challenge is to select an subset of those subtrees over overlapped carved sequences of the original input to cover the original sentence. We propose to formulate this as an Integer Linear Programming (ILP) problem, which is similar to a set cover problem. Note that, since the search space at this stage is highly constrained by the input of the previous stages, an exact decoding is efficient enough.

Example 2.2. :

"The SEC will probably vote"
"vote on early next year"
"on the proposal"
"will , he said ."

Example 2.3. :

(VBDC (MDC (DT The) (NNP SEC)
(MD will) (RB probably)
(VBC (VB vote)
(INC (IN on) (DT the) (NN proposal))
(RB early) (JJ next) (NN year)))
(, ,) (PRP he) (VBD said) (. .))

Figure 1: Example 2.2 demonstrates 'carved sequences'. Example 2.3 demonstrates the coarse constituent parsing.

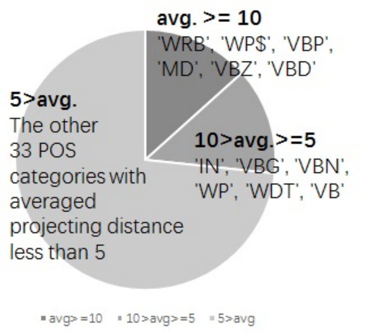

Figure 2: The averaged length of projections headed by each POS category, as computed from Penn WSJ treebank.

We experiment with two alternative approaches for scoring subtrees. A feature-based perceptron classifier, and Dependency-based Convolutional Neural Networks (DCNN) (Ma et al., 2015). Previous use of continuous representation in parsing mainly models parsing actions, phrases, words, or features. When dependency chains are modeled, they were only used for reranking of k-best lists of whole dependency trees, e.g. (Le and Zuidema, 2014; Zhu et al., 2015). Since transition-based parsers are known to act worse on long-distance dependencies, they benefit from this pipeline the most. With no algorithmic change to transition-based or first-order models, an increase of 1.28/1.21 (UAS) can be achieved solely due to the proposed factorization of input, thus achieving comparable performance with high-order models but faster. For Chinese, the most notable increase is as high as 3.63 (UAS), when the proposed framework is applied to first-order parsing models.

2 System Design

2.1 Motivation

Consider a sentence as follows:

Example 2.1. *The SEC will probably vote on the proposal early next year , he said .*

Suppose, magically, we could determine oracle 'carved' sequences of this input, such that all words in a sequence are dependent of some word in the same sequence, except for one head word, e.g. Example 2.2 in Figure 1. Further dependency parsing over these short sequences could be much easier than parsing the original input. This kind of localization is implicitly realized during a dynamic programming parsing process, assuming the optimal substructure property. We propose to explicitly capture this kind of localization by decomposing input space, especially motivated by the following two observations:

- There is a notable distinction in word categories with respect to their averaged projection scope.
- The input to a first-stage constituent-based parsing for long projections can be pruned to contain relevant peripheral and head words only (otherwise, the pipeline is of no practical interest at all.)

First of all, we acquire the distinction in word categories and define long-distance projecting POS categories according to statistics of the training corpus. Based on this statistically verifiable distinction we propose the following parsing pipeline:

(1) Constituent-based parsing for phrases headed by long-distance projecting word categories.
(2) Dependency parsing over carved sequences of input, which are transformed from the coarse constituent parsing outputs of the first stage.
(3) When k-best constituent-based parsing is employed at the first stage, search for an optimal subset of subtrees to combine into a whole dependency tree over the original input.

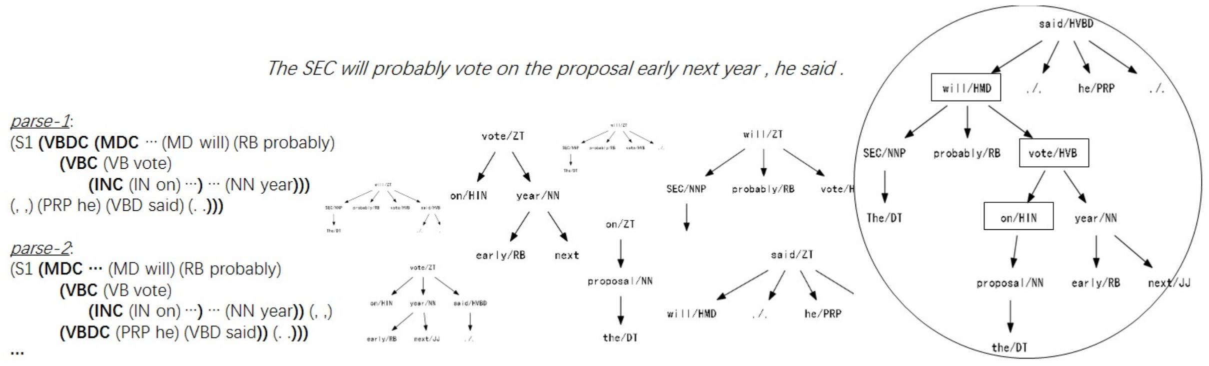
The SEC will probably vote on the proposal early next year , he said .
parse-1:
(S1 (VBDC (MDC ··· (MD will) (RB probably)
(VBC (VB vote)
(INC (IN on) ···) ··· (NN year)))
(, ,) (PRP he) (VBD said) (. .)))
parse-2:
(S1 (MDC ··· (MD will) (RB probably)
(VBC (VB vote)
(INC (IN on) ···) ··· (NN year)) (, ,)
(VBDC (PRP he) (VBD said)) (. .)))
...
vote/ZT
on/HIN
year/NN
early/RB
next
on/ZT
proposal/NN
the/DT
will/ZT
SEC/NNP
probably/RB
said/ZT
will/HMD
./.
he/PRP
said/HVBD
will/HMD
he/PRP
SEC/NNP
probably/RB
vote/HVB
The/DT
on/HIN
year/NN
proposal/NN
early/RB
next/JJ
the/DT

Figure a. First stage: K-best outputs of the coarse constituent-based parsing.

Figure b. Second stage: parse carved sequences into dependency trees. Those vague treelets at background demonstrate redundant subtrees that are not selected to the optimal subset for combination

Figure c. Third stage: combine subtrees by substitution.

Figure 3: A parsing framework of three stages.

Given previous work on parsing, e.g. (Charniak and Johnson, 2005),(McDonald et al., 2005),(Martins et al., 2013) and so on, reliable constituent-based parsers and dependency parsers are available. In other words, other than selecting the set of POS tags that are considered as long-distance projecting categories, there are few implementation details for the first and second stages. Our main implementation challenge seems lies in the third stage. However, since the search space at this stage is highly constrained by the input of the previous stages, we propose to straightforwardly model this search as an ILP problem, which can be efficiently solved by a general ILP decoder, e.g. cplex.

2.2 The first constituent parsing stage

In a projective dependency tree, we consider the projection of a head as the range of input covering all of its descendants. As shown in Figure 2, the POS categories that project longer distance in average is in accordance with our 'linguistic instinct'. Nominal categories, adverbs, adjectives and most closed-class categories, except for subordinating conjunction words and *wh-* holders, tend to project over local words only. And for other categories, we do not stipulate, but leave it as a variant in our experiments.

Given a projective tree, projections of heads can be expressed in brackets, e.g. Example 2.3 in Figure 1. At this stage, we are only interested in projections that are headed by words that fall in long-distance projecting POS categories. Given such a head word of tag 'X', for the sake of expressive convenience, we tag the corresponding bracket covering all its descendants as 'XC'. This simple rule could transform a dependency treebank to brackets of long-distance projections with no ambiguities. Any constituent-based parser that doesn't impose fixed head rules, can be trained over this type of structures, instead of a conventional Treebank. However, if this stage is as slow as a full constituent parsing process, the whole pipeline is of no practical use. As will be shown in Section 5.3, with pruned input and coarse grammar, the constituent parsing stage is no longer a painful bottleneck for parsing speed.

2.3 The second dependency parsing stage

For a bracket, e.g. *(VBC (VB vote) (INC (IN on) (DT the) (NN proposal))(RB early) (JJ next) (NN year))* in Example 2.3 of Figure 1 , we carve all its embedded brackets out, but leaving head words to hold the space. This transformation gives 'carved' sequences such as *"vote on early next year"*. As long as long-distance dependencies do not cross, this simple rule can transform long-distance constituent parsing outputs to 'carved' sequences of the original input without ambiguities. For each oracle sequence, all of its words are guaranteed to be dependant of some word in the same range except for one head word. For example, in Figure 1 brackets in Example 2.3 are transformed to sequences in Example 2.2 by this rule.

Dependency parsing over these carved sequences is much easier, in the sense that both performance and efficiency are dramatically improved. If the first-stage constituent parsing offers reliable 1-best output, we can combine subtrees over these sequences by substituting each word, r, with the subtree rooted at r, if any. For example, in Figure 3-c, a squared node indicates that a subtree is substituted here. We borrow the term 'substitution' from TAG formalism (Joshi and Schabes, 1997). However, when k-best constituent parsing outputs are kept for reducing error propagation, further search strategy for an optimal subset of these redundant subtrees will be discussed in Section 3.

3 Combination of subtrees

At the first k-best constituent parsing stage, we generate ”carved” sequences to be parsed at the second dependency parsing stage. At the third stage, dependency parsing is reduced to the search of an optimal subset to form a legal tree over the original whole input. In this section, we show how to formulate this search as an Integer linear Programming problem, and alternative approaches for scoring subtrees.

3.1 Formulated as an ILP problem

Given a set of subtrees, searching for its optimal subset over the original input can be straightforwardly formulated as an Integer Linear Programming (ILP) problem, which is similar to a set cover problem. The only practical concern is whether it can be solved efficiently by a general ILP decoder. As will be shown with experiments in Section 5.3, the search space of this stage is well-constrained by outputs from the previous stages, therefore this succinct idea nicely works out in the proposed pipeline.

Because subtree outputs from the second stage already satisfy tree requirements, we only need to impose the globally single-root and single-headed constraints on a compatible subset, but not to worry about the non-cyclic requirement. We introduce a designated root \$ for the final dependency tree. For each candidate subtree, add a dummy subtree that includes only a single arc from the designated root \$ to the subtree’s root node. Suppose there are n words in the original input sentence, $N = 1...n$, not including the designated root node. Let **T** be a set of subtrees, **T** $=t_1...t_s$. Our goal is to find an optimal solution $\mathbf{x} = x_1...x_s$ that maximizes $\sum_{i=1}^{s} score(t_i) \cdot x_i$, and subjects to

$$(1) \sum_{i:n\in t_i \text{as non-root node}} x_i = 1,\ \forall n \in N; \qquad (2)\ x_i \in \{0,1\}, 1 \le i \le s$$

The boolean value of x_i, as guaranteed by constraint (2), indicates whether t_i is selected in the combination solution or not. Constraint (1) requires every node has one and only one head, and the introduction of the dummy root trees guarantees single-root.

3.2 Scoring of subtrees

A dependency tree y is commonly factored into smaller *parts*, which can be scored in isolation. At the third stage of our parsing pipeline, an overall dependency tree is computed by the combination of a compatible set of subtrees, thus tree score sums over subtree scores. In this sense, our parsing model is *subtree-factored*. We consider two alternatives for scoring subtrees:

(1) further factor subtrees into smaller parts; or

(2) Dependency-based Convolutional Neural Networks to model dependency chains in trees.

3.3 Factor subtrees to smaller parts

A ’first-order’ parser decomposes dependency trees into arcs; and for second-order and third-order factorizations, parts of consecutive siblings, grandchild, grand-sibling and tri-sibling, have been widely studied in literature, e.g. (Eisner, 2000; McDonald and Pereira, 2006; Koo and Collins, 2010). Since our parsing model is subtree-factored, there is no limitation on high-order features for us to consider. However, in practice, we follow the second-order and third-order notations.

3.4 DCNN for scoring subtrees

Dependency-based Convolutional Neural Networks were originally proposed for sentence embedding (Ma et al., 2015) and evaluated for sentiment analysis and question classification tasks. We adopt this model for scoring subtrees since it captures dependency chains in trees including ancestor paths and siblings. Given parsed subtrees from the second stage, an oracle combination computes subtree scores as the number of gold arcs it contains. For any input sentence, if a subtree is selected into the oracle combination, it is considered as a 'legal' subtree otherwise 'illegal'. A subtree classifier, e.g. DCNN, is trained to tell legal subtrees apart from illegal subtrees. This is not a structured learning schema as we are familiar with for parsing tasks, since DCNN is not designed to capture specific structural factorization of dependency trees, such as arcs or RCNN units as defined in (Zhu et al., 2015).

4 Related work

Klein and Manning (2003) proposed to factor parsing model into a phrase-structure tree and a dependency tree, even before dependency parsing was well-studied later in literature. The idea of combining the merits of constituents and dependency parsing is not new, but our proposed factorization is novel.

Since k-best constituent parsing outputs are used, this work resembles the re-ranking works. However, we do not perform k-best list re-ranking, e.g. (Charniak and Johnson, 2005; Hall, 2007; Qian and Liu, 2015). Neither forest reranking (cube pruning), e.g. (Huang, 2008; Zhang and McDonald, 2012; Hayashi et al., 2011). Our search space is factored into pieces of subtrees, thus represents more combinatory candidates than a k-best list of whole dependency trees. This decomposition distinguishes our work from (Le and Zuidema, 2014) and (Zhu et al., 2015) which use NN-based models to re-rank whole dependency trees. These models could also be evaluated in our parsing pipeline. Furthermore, our work concerns very different aspects with (Ren et al., 2013), which uses dependency model for forest reranking of constituent-based parsing. Our dependency parsing process deals with local dependencies only, a complementary task to the constituent parsing process. The third combination stage performs a global search, neither as a simple reranking. It is more accurate to describe our pipeline as imposing constituent-based structural constraints to dependency parsing. Even though imposing constraints practically performs similarly as pruning, it provokes interesting work, e.g. recent work on approximate Linear Programming decoders for parsing, (Koo et al., 2010; Martins et al., 2011). In this work, we also formulate the search of optimal combination of subtrees as an ILP problem, and due to our efforts on constraining the search space in previous stages, it can be efficiently solved by exact decoding.

Furthermore, works on Vine parsing, e.g. (Dreyer et al., 2006; Rush and Petrov, 2012), especially relate to ours. This line of work pays special attention to lengths of arcs and consider it as an import factor to constrain the search. Instead of pruning arcs by distance like Vine parsing, we decompose parsing of long-distance projections and localized dependencies, which is characterized by pattern in word categories but not absolute distance. There is also a more recent related work, (Fernandez-Gonzalez et al., 2016), that also pays attention to length of arcs. In their work, they use simple criteria based on the length and position of dependency arcs to determine how to combine the outputs of an left-to right transition-based parser and its "mirrored" version.

5 Experiments

Our main experiments are performed on dependency trees extracted from English WSJ Treebank (Marcus et al., 1993). We use Yamada and Matsumoto (2003)'s head rules to convert phrase structures to dependency structures, considering the productivity assumption. Following the conventional split, we use sections 02-21 for training, section 22 for development and section 23 for testing. Dependency parsing is evaluated by unlabeled attachment score (UAS), which is the percentage of words that correctly identified their heads. Chinese experiments are performed on CTB5 with the conventional splits described in (Zhu et al., 2015).

For both English and Chinese experiments, the BLLIP parser (Charniak and Johnson, 2005) is used for the first-stage constituent parsing. Because the BLLIP parser doesn't require POS tag input, we do not impose gold POS tags or use an automatic POS tagger. We use 10-fold cross-validation training data at the third combination stage, since the subtree classifier needs to cope with noisy input from the previous stages. More specifically, we divide training corpus into 10 folds and train a coarse constituent-based parser for each divide, then combine all parsing outputs from each divide into a whole training corpus containing k-best constituent parsing outputs.

Oracle parsing is performed by the oracle combination of oracle-parsed subtrees transformed from k-best constituent parsing outputs. We have defined oracle combination in Section 3.4. Given a carved

	constituents recall 200-best	dependency UAS gold subtrees	oracle combination
avg.$>=5$	**98.34**	**96.21**	**95.24**
avg.$>=10$	97.97	92.19	91.75

Table 1: The threshold of averaged length for long-distance projecting POS categories affects constituent parsing and dependency parsing.

avg. $>=5$	constituents k-best recall	avg. number of subtrees	oracle parsing
$k=200$	98.34%	59.7	99.46
$k=500$	98.41%	72.75	99.51
$k=1000$	98.43%	80.75	99.52

Table 2: The choice of k in k-best constituent parsing.

sequence of input, each word's oracle head is either its gold head, or the root of this sequence when the word's gold head is not seen in the same sequence. Oracle-parsed subtrees are not used for training, but only for tuning parameters on development sets.

5.1 The first-stage constituent parsing

As discussed in section 2.2, there are two distinct jumps in the distribution over averaged lengths of projections headed by each POS category. We select the set of POS tags that are considered as long-distance projecting by tuning a threshold on the development set. As shown in Table 1, a proper setting of this threshold matters for all parsing stages. For following experiments on English (also Chinese and German), we consider a POS category as long-distance projecting, if the averaged distance of its projections is more than or equal to 5. For English, this setting gives us all verbal categories, *wh*-holders, and prepositions. For Chinese, it gives long-distance projecting categories of 'VA', 'DEC', 'P', 'CS', 'SP', 'VV', 'VE', 'LB', 'BA' and 'VC', whose meanings are referred to (Xia et al., 2000).

The choice of k defines k-best constituent parsing outputs that will be transformed to carved sequences in the following dependency parsing stage. As shown in Table 2, a larger k doesn't introduces sharply more subtrees per sentence, because most brackets in the top k-best constituent parsing outputs are the same. This observation shows a great advantage of our factorization and distinguishes our pipeline from k-best list reranking of whole trees. We can make use of a relatively large k to achieve higher recall, which is set to be 500 in the following experiments on English and Chinese.

5.2 Dependency Parsing Results

baseline parsing models	directly parse whole trees to compare	parsing subtrees in pipeline	oracle combination of parsed subtrees	combination by subtree scores of order-1 ptron	combination by subtree scores of order-2 ptron	combination by subtree scores of DCNN
NNDep	91.59	97.05	96.13	92.52	92.87(**+1.28**)	92.79
Mst-1	91.39	96.93	95.90	92.38	92.60 (+1.21)	92.45
Mst-2	92.13	97.05	96.05	92.51	92.88 (+0.75)	92.57
Turbo-standard	93.11	97.34	96.31	92.85	93.17 (+0.06)	92.88
Turbo-full	93.43	97.35	96.36	92.91	93.24 (-0.19)	92.93
merged	N/A	N/A	97.31	93.25	**93.57**	93.26

Table 3: Performance (UAS) of parsing by the combination of subtrees. The order-1/2 perceptron-based subtree classifier and DCNN are three alternative combination strategies. The highest increase of UAS for each baseline model is given in parentheses. A merge of the subtrees from all baseline models can be used to improve the second dependency parsing stage.

We could pick any dependency model for subtree dependency parsing, especially the following:

- An NN-based transition-based model, NNDep, (Chen and Manning, 2014).
- A first- (second-) order graph-based model, MST-1(2), (McDonald and Pereira, 2006).
- A third-order(and beyond) approximate model, Turbo-standard(full), (Martins et al., 2011).

For scoring subtrees, we experiment with a feature-based perceptron classifier of first/second- order MST features, as well as the DCNN model described in Section 3.4. Given the well-known over-fitting problem for re-ranking models, we adopt the same mixture strategies as both (Le and Zuidema, 2014)

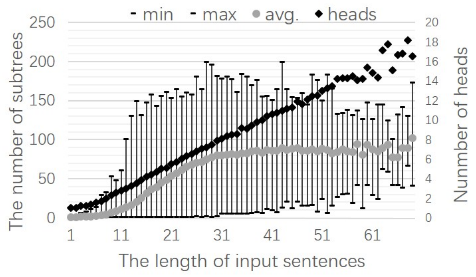

Figure 4: The range (min, max, average) of the number of subtrees (primary y-axis) and the averaged number of long-distance projecting heads (secondary y-axis) with respect to the length of input sentences as computed from 500-best constituent parsing outputs.

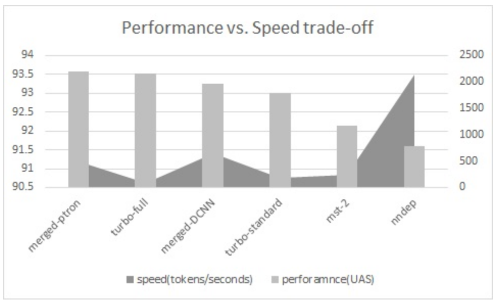

Figure 5: The trade-off between performance (UAS on the promary y-axis) and speed (tokens/seconds on the secondary y-axis).

and (Zhu et al., 2015) with arc scores by a standard Turbo parser. The classifiers are trained with merged dependency parsing outputs, since to do 10-fold cross-validation training for every baseline model is overloading. For a given set of carved sequences transformed from constituent parsing outputs, the merged dependency parsing model covers the subtrees from all baseline models, thus increasing the oracle-combination score by 0.95 even compared to Turbo-full, the best baseline model. More specifically, the feature-based classifiers are trained with online perceptron-based learning (Collins, 2002). DCNN is trained off-line with the same settings as (Ma et al., 2015). And oracle combination as described in Section 3.4 is used to guide training.

As shown in Table 3, with the merged dependency parsing model, our parsing system performs better than Turbo-full, by 0.14 (UAS). However, this is not what we push for in this work. We are more encouraged by the results that, with no algorithmic change to transition-based or first-order models, an increase of 1.28/1.21 (UAS) can be achieved solely due to a factorization of input. It is worth to notice that, on full dependency parsing, Turbo-full performs better than the transition-based and first-order models by nearly 2 (UAS). However, all baseline models achieve the same level of performance for subtree dependency parsing, no bigger difference than 0.5 (UAS). This is enough to show that the proposed factorization works. The use of DCNN doesn't introduce special gain in performance, however, we observe the same advantage as described in (Chen and Manning, 2014), i.e. the NN-based model is nearly 40 times faster than the feature-based discriminative classifier.

5.3 Efficiency

The efficiency of solving the ILP problem at the third combination stage depends on the number of possible subtrees generated at the second stage. As shown in Figure 4, the averaged number of long-distance projecting heads is linearly dependent on the input length, which suggests that the number of subtrees increases at most linearly. Furthermore, since we only keep k-best constituent parsing outputs, the averaged number of subtrees stays below a constant. When k is set as large as 500, the averaged number of subtrees still stays below 100, as shown in Figure 4. Therefore a general ILP decoder, e.g. cplex, is efficient enough. This efficient combination clearly shows the strength of the proposed factorization.

As shown in Table 3, all baseline models achieve comparable performance for subtree dependency parsing. In this sense, we can now make use of the fastest dependency parsing model for subtree parsing, without concerns in performance loss.

It turns out that the k-best constituent parsing stage is the efficiency bottleneck. Therefore, we propose to prune the input to this stage. As a first try, we tag all head words with 'H', any beginning or end words of some bracket with 'B', and other words with 'I'. Then those words of tag 'I' can be pruned for constituent parsing. However, this classification pattern cannot be acquired. This failure is worth special notice, it reminds us that the success of a classification task not only depends on powerful machine learning techniques but also crucially on appropriate representations. We then keep all punctuations, all words before head words and all words before punctuational bracket endings , i.e. tag them with label 'B'. This tagging task can be performed with a precision above 93%, with a perceptron tagger. However,

	grammar size	input avg./max	words per sec.	oracle parsing
full	69630	24/99	681	99.51
pruned	57256	18/76	912	99.50

Table 4: Compare full and pruned input to constituent parsing. Grammar size is the number of unlexicalized rewrite rules.

(UAS)	baseline	subtree	oracle combination	order-2 ptron
MST-1	80.48	96.61	88.54	84.11 (**+3.63**)
Turbo-full	84.74	97.08	89.07	84.35 (-0.39)

Table 5: Results on CTB5, the same terms of Table 3.

high recall of 'B' and 'H' is more crucial for our task, i.e. relevant input are not wrongly pruned. For decoding, we set $b = 20$ for tag 'B' and 'H', and 0 for tag 'I', achieving 98.4% recall of tag 'B' and 'H', still pruning 27.5% of the original input. As shown in Table 4, by feeding pruned input to the coarse constituent parsing, we can process about 25% words more per second at this stage.

In Figure 5, we show that the proposed pipeline provides a better trade-off between parsing performance and parsing efficiency. Merged-ptron achieves comparable performance (93.57 UAS) with Turbo-full (93.24 UAS) but 5 times faster. Even though NNDep (92.87 UAS) is about 6 times faster than merged-ptron, there is a loss of 1.3 (UAS) in performance.

6 Experiments on Chinese

It is well-known that, Chinese parsing performance is much lower than English, for example, with MST-1 and Turbo-full, it is only 80.48 and 84.74 (UAS) respectively for full dependency parsing. Thus it is a surprise for us to see in Table 5 that, for subtree dependency parsing, MST-1 and Turbo-full achieve the same level of parsing performance on Chinese as on English, 96.61 and 97.08 (UAS) respectively. It is the first constituent parsing stage that reveals the difficulty in parsing Chinese. With 500-best c-parsing over full input, the recall is merely 88.63%, and even with the proposed pruned input, the recall is improved by 1% only. However, if we prune the input for constituent parsing with oracle 'H', 'B', 'I' tags, a recall as high as 97% can be achieved. This big gap in results leaves a huge space for future work to explore. The proposed factorization intriguingly discovers the bottleneck in Chinese parsing.

For non-projective parsing, we take the German case as an example. There are about 3 percent of arcs are crossing in the TIGER corpus (Brants et al., 2002). Recall that the projective assumption is only essential to the first constituent parsing stage. If these crossing arcs are local, our pipeline still works. However, 98% of the crossing arcs involve long-distance projecting heads. We can employ techniques such as super-tagging instead of constituent parsing to deal with non-projective long-distance dependencies .

7 Conclusion and Discussion

We propose a novel factorization of parsing task that explicitly utilizes the distinction between long-distance projections and localized dependencies. This intuitive idea works out given that this factorization can be characterized by POS categories of each projection's head word. The first and second stages, which cope with long-distance projections and localized dependencies separately, are fed with complementary input instead of full input, thus taking great advantage of the constrained search space to perform better and faster.

In this work, the proposed factorization is realized in a parsing pipeline of three stages. At the second stage, any dependency parsing model can be used for subtree parsing, so 'baseline' models in our work are not only used for comparison, but to show how much increase in parsing performance can be introduced solely by the proposed factorization, with no change in parsing algorithms or learning strategies to baseline models. On the final stage, dependency parsing is subtree-factored, thus any high-order feature-based model and even continuous representation of dependency chains can be used for subtree scoring. Moreover, dependency parsing over carved sequences of the original input especially suits for parallel computing. Therefore, the proposed factorization provides a better trade-off in parsing speed and performance. In future work, we would like to experiment with end-to-end learning framework for future reducing error propagation.

Acknowledgements

We sincerely thank the anonymous reviewers for their thorough reviewing and valuable suggestions. Qun Liu's work is partially supported by Science Foundation Ireland at ADAPT Centre for Digital Content Platform Research (Grant 13/RC/2106).

References

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The tiger treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, volume 168.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar, October. Association for Computational Linguistics.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics.

Christopher D. Manning Dan Klein. 2003. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems 15: Proceedings of the 2002 Conference*, volume 15, page 3. MIT Press.

Markus Dreyer, David A. Smith, and Noah A. Smith. 2006. Vine parsing and minimum risk reranking for speed and precision. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 201–205, New York City, June. Association for Computational Linguistics.

Jason M Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics.

Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies*, pages 29–61. Springer.

Daniel Fernandez-Gonzalez, Carlos Gomez-Rodriguez, and David Vilares. 2016. Improving the arc-eager model with reverse parsing. *Computing & Informatics*, 35(3).

Keith Hall. 2007. K-best spanning tree parsing. In *Annual Meeting-Association for Computational Linguistics*, volume 45, page 392.

Katsuhiko Hayashi, Taro Watanabe, Masayuki Asahara, and Yuji Matsumoto. 2011. Third-order variational reranking on packed-shared dependency forests. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1479–1488. Association for Computational Linguistics.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pages 586–594, Columbus, Ohio, June. Association for Computational Linguistics.

Aravind K Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In *Handbook of formal languages*, pages 69–123. Springer.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In *In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–11, Uppsala.

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

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 729–739, Doha, Qatar, October. Association for Computational Linguistics.

Mingbo Ma, Liang Huang, Bowen Zhou, and Bing Xiang. 2015. Dependency-based convolutional neural networks for sentence embedding. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 174–179, Beijing, China, July. Association for Computational Linguistics.

Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330.

André FT Martins, Noah A Smith, Pedro MQ Aguiar, and Mário AT Figueiredo. 2011. Dual decomposition with many overlapping components. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 238–249. Association for Computational Linguistics.

Andre Martins, Miguel Almeida, and A. Noah Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622. Association for Computational Linguistics.

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

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Association for Computational Linguistics (ACL)*.

J. Nivre, J. Hall, J. Nilsson, G. Eryiğit, and S. Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 221–225.

Xian Qian and Yang Liu. 2015. Feature selection in kernel space: A case study on dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1180–1190, Beijing, China, July. Association for Computational Linguistics.

Xiaona Ren, Xiao Chen, Chunyu Kit, Tower D SOGOU-INC, and Tsinghua Tongfang High-tech Plaza. 2013. Combine constituent and dependency parsing via reranking. In *IJCAI*.

Alexander Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Montréal, Canada, June. Association for Computational Linguistics.

Fei Xia, Martha Palmer, Nianwen Xue, Mary Ellen Okurowski, John Kovarik, Fu dong Chiou, Shizhe Huang, Tony Kroch, and Mitch Marcus. 2000. Developing guidelines and ensuring consistency for chinese text annotation. In *In Proceedings of the Second Language Resources and Evaluation Conference*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *8th International Workshop on Parsing Technologies*.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331, Jeju Island, Korea, July. Association for Computational Linguistics.

Chenxi Zhu, Xipeng Qiu, Xinchi Chen, and Xuanjing Huang. 2015. A re-ranking model for dependency parser with recursive convolutional neural network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1159–1168, Beijing, China, July. Association for Computational Linguistics.