Branch and Bound Algorithm for Dependency Parsing with Non-local Features

Xian Qian and **Yang Liu**
Computer Science Department
The University of Texas at Dallas
{qx,yangl}@hlt.utdallas.edu

Abstract

Graph based dependency parsing is inefficient when handling non-local features due to high computational complexity of inference. In this paper, we proposed an exact and efficient decoding algorithm based on the Branch and Bound (B&B) framework where non-local features are bounded by a linear combination of local features. Dynamic programming is used to search the upper bound. Experiments are conducted on English PTB and Chinese CTB datasets. We achieved competitive Unlabeled Attachment Score (UAS) when no additional resources are available: 93.17% for English and 87.25% for Chinese. Parsing speed is 177 words per second for English and 97 words per second for Chinese. Our algorithm is general and can be adapted to non-projective dependency parsing or other graphical models.

1 Introduction

For graph based projective dependency parsing, dynamic programming (DP) is popular for decoding due to its efficiency when handling local features. It performs cubic time parsing for arc-factored models (Eisner, 1996; McDonald et al., 2005a) and biquadratic time for higher order models with richer sibling and grandchild features (Carreras, 2007; Koo and Collins, 2010). However, for models with general non-local features, DP is inefficient.

There have been numerous studies on global inference algorithms for general higher order parsing. One popular approach is reranking (Collins, 2000; Charniak and Johnson, 2005; Hall, 2007). It typically has two steps: the low level classifier generates the top k hypotheses using local features, then the high level classifier reranks these candidates using global features. Since the reranking quality is bounded by the oracle performance of candidates, some work has combined candidate generation and reranking steps using cube pruning (Huang, 2008; Zhang and McDonald, 2012) to achieve higher oracle performance. They parse a sentence in bottom up order and keep the top k derivations for each span using k best parsing (Huang and Chiang, 2005). After merging the two spans, non-local features are used to rerank top k combinations. This approach is very efficient and flexible to handle various non-local features. The disadvantage is that it tends to compute non-local features as early as possible so that the decoder can utilize that information at internal spans, hence it may miss long historical features such as long dependency chains.

Smith and Eisner modeled dependency parsing using Markov Random Fields (MRFs) with global constraints and applied loopy belief propagation (LBP) for approximate learning and inference (Smith and Eisner, 2008). Similar work was done for Combinatorial Categorial Grammar (CCG) parsing (Auli and Lopez, 2011). They used posterior marginal beliefs for inference to satisfy the tree constraint: for each factor, only legal messages (satisfying global constraints) are considered in the partition function.

A similar line of research investigated the use of integer linear programming (ILP) based parsing (Riedel and Clarke, 2006; Martins et al., 2009). This

Transactions of the Association for Computational Linguistics, 1 (2013) 37–48. Action Editor: Ryan McDonald.
Submitted 11/2012; Revised 2/2013; Published 3/2013.

method is very expressive. It can handle arbitrary non-local features determined or bounded by linear inequalities of local features. For local models, LP is less efficient than DP. The reason is that, DP works on a small number of dimensions in each recursion, while for LP, the popular revised simplex method needs to solve a m dimensional linear system in each iteration (Nocedal and Wright, 2006), where m is the number of constraints, which is quadratic in sentence length for projective dependency parsing (Martins et al., 2009).

Dual Decomposition (DD) (Rush et al., 2010; Koo et al., 2010) is a special case of Lagrangian relaxation. It relies on standard decoding algorithms as oracle solvers for sub-problems, together with a simple method for forcing agreement between the different oracles. This method does not need to consider the tree constraint explicitly, as it resorts to dynamic programming which guarantees its satisfaction. It works well if the sub-problems can be well defined, especially for joint learning tasks. However, for the task of dependency parsing, using various non-local features may result in many overlapped sub-problems, hence it may take a long time to reach a consensus (Martins et al., 2011).

In this paper, we propose a novel Branch and Bound (B&B) algorithm for efficient parsing with various non-local features. B&B (Land and Doig, 1960) is generally used for combinatorial optimization problems such as ILP. The difference between our method and ILP is that the sub-problem in ILP is a relaxed LP, which requires a numerical solution, while ours bounds the non-local features by a linear combination of local features and uses DP for decoding as well as calculating the upper bound of the objective function. An exact solution is achieved if the bound is tight. Though in the worst case, time complexity is exponential in sentence length, it is practically efficient especially when adopting a pruning strategy.

Experiments are conducted on English PennTree Bank and Chinese Tree Bank 5 (CTB5) with standard train/develop/test split. We achieved 93.17% Unlabeled Attachment Score (UAS) for English at a speed of 177 words per second and 87.25% for Chinese at a speed of 97 words per second.

2 Graph Based Parsing

2.1 Problem Definition

Given a sentence $x = x_1, x_2, \ldots, x_n$ where x_i is the i^{th} word of the sentence, dependency parsing assigns exactly one head word to each word, so that dependencies from head words to modifiers form a tree. The root of the tree is a special symbol denoted by x_0 which has exactly one modifier. In this paper, we focus on unlabeled projective dependency parsing but our algorithm can be adapted for labeled or non-projective dependency parsing (McDonald et al., 2005b).

The inference problem is to search the optimal parse tree y^*

$$y^* = \arg\max_{y \in \mathcal{Y}(x)} \phi(x, y)$$

where $\mathcal{Y}(x)$ is the set of all candidate parse trees of sentence x. $\phi(x, y)$ is a given score function which is usually decomposed into small parts

$$\phi(x, y) = \sum_{c \subseteq y} \phi_c(x) \tag{1}$$

where c is a subset of edges, and is called a factor. For example, in the all grandchild model (Koo and Collins, 2010), the score function can be represented as

$$\phi(x, y) = \sum_{e_{hm} \in y} \phi_{e_{hm}}(x) + \sum_{e_{gh}, e_{hm} \in y} \phi_{e_{gh}, e_{hm}}(x)$$

where the first term is the sum of scores of all edges $x_h \rightarrow x_m$, and the second term is the sum of the scores of all edge chains $x_g \rightarrow x_h \rightarrow x_m$.

In discriminative models, the score of a parse tree y is the weighted sum of the fired feature functions, which can be represented by the sum of the factors

$$\phi(x, y) = \mathbf{w}^T \mathbf{f}(x, y) = \sum_{c \subseteq y} \mathbf{w}^T \mathbf{f}(x, c) = \sum_{c \subseteq y} \phi_c(x)$$

where $\mathbf{f}(x, c)$ is the feature vector that depends on c. For example, we could define a feature for grandchild $c = \{e_{gh}, e_{hm}\}$

$$f(x, c) = \begin{cases} 1 & \text{if } x_g = \text{would } \wedge\, x_h = \text{be} \\ & \wedge x_m = \text{happy } \wedge\, c \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

2.2 Dynamic Programming for Local Models

In first order models, all factors c in Eq(1) contain a single edge. The optimal parse tree can be derived by DP with running time $O(n^3)$ (Eisner, 1996). The algorithm has two types of structures: complete span, which consists of a headword and its descendants on one side, and incomplete span, which consists of a dependency and the region between the head and modifier. It starts at single word spans, and merges the spans in bottom up order.

For second order models, the score function $\phi(x, y)$ adds the scores of siblings (adjacent edges with a common head) and grandchildren

$$\begin{aligned}\phi(x,y) &= \sum_{e_{hm}\in y} \phi_{e_{hm}}(x) \\ &+ \sum_{e_{gh},e_{hm}\in y} \phi_{e_{hm},e_{gh}}(x) \\ &+ \sum_{e_{hm},e_{hs}\in y} \phi_{e_{hm},e_{hs}}(x)\end{aligned}$$

There are two versions of second order models, used respectively by Carreras (2007) and Koo et al. (2010). The difference is that Carreras' only considers the outermost grandchildren, while Koo and Collin's allows all grandchild features. Both models permit $O(n^4)$ running time.

Third-order models score edge triples such as three adjacent sibling modifiers, or grand-siblings that score a word, its modifier and its adjacent grandchildren, and the inference complexity is $O(n^4)$ (Koo and Collins, 2010).

In this paper, for all the factors/features that can be handled by DP, we call them the local factors/features.

3 The Proposed Method

3.1 Basic Idea

For general high order models with non-local features, we propose to use Branch and Bound (B&B) algorithm to search the optimal parse tree. A B&B algorithm has two steps: branching and bounding. The branching step recursively splits the search space $\mathcal{Y}(x)$ into two disjoint subspaces $\mathcal{Y}(x) = \mathcal{Y}_1 \bigcup \mathcal{Y}_2$ by fixing assignment of one edge. For each subspace $\mathcal{Y}_i$, the bounding step calculates the upper bound of the optimal parse tree score in the subspace: $UB_{\mathcal{Y}_i} \geq \max_{y\in\mathcal{Y}_i} \phi(x, y)$. If this bound is no more than any obtained parse tree score $UB_{\mathcal{Y}_i} \leq \phi(x, y')$, then all parse trees in subspace $\mathcal{Y}_i$ are no more optimal than y', and $\mathcal{Y}_i$ could be pruned safely.

The efficiency of B&B depends on the branching strategy and upper bound computation. For example, Sun et al. (2012) used B&B for MRFs, where they proposed two branching strategies and a novel data structure for efficient upper bound computation. Klenner and Ailloud (2009) proposed a variation of Balas algorithm (Balas, 1965) for coreference resolution, where candidate branching variables are sorted by their weights.

Our bounding strategy is to find an upper bound for the score of each non-local factor c containing multiple edges. The bound is the sum of new scores of edges in the factor plus a constant

$$\phi_c(x) \leq \sum_{e\in c} \psi_e(x) + \alpha_c$$

Based on the new scores $\{\psi_e(x)\}$ and constants $\{\alpha_c\}$, we define the new score of parse tree y

$$\psi(x, y) = \sum_{c\subseteq y} \left(\sum_{e\in c} \psi_e(x) + \alpha_c \right)$$

Then we have

$$\psi(x, y) \geq \phi(x, y), \forall y \in \mathcal{Y}(x)$$

The advantage of such a bound is that, it is the sum of new edge scores. Hence, its optimum tree $\max_{y\in\mathcal{Y}(x)} \psi(x, y)$ can be found by DP, which is the upper bound of $\max_{y\in\mathcal{Y}(x)} \phi(x, y)$, as for any $y \in \mathcal{Y}(x)$, $\psi(x, y) \geq \phi(x, y)$.

3.2 The Upper Bound Function

In this section, we derive the upper bound function $\psi(x, y)$ described above. To simplify notation, we drop x throughout the rest of the paper. Let z_c be a binary variable indicating whether factor c is selected in the parse tree. We reformulate the score function in Eq(1) as

$$\phi(y) \equiv \phi(z) = \sum_c \phi_c z_c \qquad (2)$$

Correspondingly, the tree constraint is replaced by $z \in \mathcal{Z}$. Then the parsing task is

$$z^* = \arg\max_{z \in \mathcal{Z}} \phi_c z_c \tag{3}$$

Notice that, for any z_c, we have

$$z_c = \min_{e \in c} z_e$$

which means that factor c appears in parse tree if and only if all its edges $\{e|e \in c\}$ are selected in the tree. Here z_e is short for $z_{\{e\}}$ for simplicity.

Our bounding method is based on the following fact: for a set $\{a^1, a^2, \dots a^r\}$ (a^j denotes the j^{th} element) , its minimum

$$\min\{a^j\} = \min_{p \in \Delta} \sum_j p^j a^j \tag{4}$$

where Δ is probability simplex

$$\Delta = \{p|p^j \geq 0, \sum_j p^j = 1\}$$

We discuss the bound for $\phi_c z_c$ in two cases: $\phi_c \geq 0$ and $\phi_c < 0$.

If $\phi_c \geq 0$, we have

$$\begin{aligned} \phi_c z_c &= \phi_c \min_{e \in c} z_e \\ &= \phi_c \min_{p_c \in \Delta} \sum_{e \in c} p_c^e z_e \\ &= \min_{p_c \in \Delta} \sum_{e \in c} \phi_c p_c^e z_e \end{aligned}$$

The second equation comes from Eq(4). For simplicity, let

$$g_c(p_c, z) = \sum_{e \in c} \phi_c p_c^e z_e$$

with domain $\mathbf{dom} g_c = \{p_c \in \Delta; z_e \in \{0,1\}, \forall e \in c\}$. Then we have

$$\phi_c z_c = \min_{p_c} g_c(p_c, z) \tag{5}$$

If $\phi_c < 0$, we have two upper bounds. One is commonly used in ILP when all the variables are binary

$$\begin{aligned} a^* &= \min_j \{a^j\}_{j=1}^r \\ &\Leftrightarrow \\ a^* &\leq a^j \\ a^* &\geq \sum_j a^j - (r-1) \end{aligned}$$

According to the last inequality, we have the upper bound for negative scored factors

$$\phi_c z_c \leq \phi_c \left(\sum_{e \in c} z_e - (r_c - 1) \right) \tag{6}$$

where r_c is the number of edges in c. For simplicity, we use the notation

$$\sigma_c(z) = \phi_c \left(\sum_{e \in c} z_e - (r_c - 1) \right)$$

The other upper bound when $\phi_c < 0$ is simple

$$\phi_c z_c \leq 0 \tag{7}$$

Notice that, for any parse tree, one of the upper bounds must be tight. Eq(6) is tight if c appears in the parse tree: $z_c = 1$, otherwise Eq(7) is tight. Therefore

$$\phi_c z_c = \min\{\sigma_c(z), 0\}$$

Let

$$h_c(p_c, z) = p_c^1 \sigma_c(z) + p_c^2 \cdot 0$$

with $\mathbf{dom} h_c = \{p_c \in \Delta; z_e \in \{0,1\}, \forall e \in c\}$. According to Eq(4), we have

$$\phi_c z_c = \min_{p_c} h_c(p_c, z) \tag{8}$$

Let

$$\psi(p, z) = \sum_{c, \phi_c \geq 0} g_c(p_c, z) + \sum_{c, \phi_c < 0} h_c(p_c, z)$$

Minimize ψ with respect to p, we have

$$\begin{aligned} &\min_p \psi(p, z) \\ =\ &\min_p \left(\sum_{c, \phi_c \geq 0} g_c(p_c, z) + \sum_{c, \phi_c < 0} h_c(p_c, z) \right) \\ =\ &\sum_{c, \phi_c \geq 0} \min_{p_c} g_c(p_c, z) + \sum_{c, \phi_c < 0} \min_{p_c} h_c(p_c, z) \\ =\ &\sum_{c, \phi_c \geq 0} \phi_c z_c + \sum_{c, \phi_c < 0} \phi_c z_c \\ =\ &\phi(z) \end{aligned}$$

The second equation holds since, for any two factors, c and c', g_c (or h_c) and $g_{c'}$ (or $h_{c'}$) are separable. The third equation comes from Eq(5) and Eq(8).

Based on this, we have the following proposition:

Proposition 1. *For any* $p, p_c \in \Delta$*, and* $z \in \mathcal{Z}$*,* $\psi(p, z) \geq \phi(z)$.

Therefore, $\psi(p, z)$ is an upper bound function of $\phi(z)$. Furthermore, fixing p, $\psi(p, z)$ is a linear function of z_e , see Eq(5) and Eq(8), variables z_c for large factors are eliminated. Hence $z' = \arg\max_z \psi(p, z)$ can be solved efficiently by DP.

Because

$$\psi(p, z') \geq \psi(p, z^*) \geq \phi(z^*) \geq \phi(z')$$

after obtaining z' , we get the upper bound and lower bound of $\phi(z^*)$: $\psi(p, z')$ and $\phi(z')$.

The upper bound is expected to be as tight as possible. Using min-max inequality, we get

$$\begin{aligned} \max_{z \in \mathcal{Z}} \phi(z) &= \max_{z \in \mathcal{Z}} \min_{p} \psi(p, z) \\ &\leq \min_{p} \max_{z \in \mathcal{Z}} \psi(p, z) \end{aligned}$$

which provides the tightest upper bound of $\phi(z^*)$.

Since ψ is not differentiable w.r.t p, projected sub-gradient (Calamai and Moré, 1987; Rush et al., 2010) is used to search the saddle point. More specifically, in each iteration, we first fix p and search z using DP, then we fix z and update p by

$$p_{new} = P_\Delta \left(p + \frac{\partial \psi}{\partial p} \alpha \right)$$

where $\alpha > 0$ is the step size in line search, function $P_\Delta(q)$ denotes the projection of q onto the probability simplex Δ. In this paper, we use Euclidean projection, that is

$$P_\Delta(q) = \min_{p \in \Delta} \|p - q\|_2$$

which can be solved efficiently by sorting (Duchi et al., 2008).

3.3 Branch and Bound Based Parsing

As discussed in Section 3.1, the B&B recursive procedure yields a binary tree structure called Branch and Bound tree. Each node of the B&B tree has some fixed z_e, specifying some must-select edges and must-remove edges. The root of the B&B tree has no constraints, so it can produce all possible parse trees including z^*. Each node has two children. One adds a constraint $z_e = 1$ for a free edge

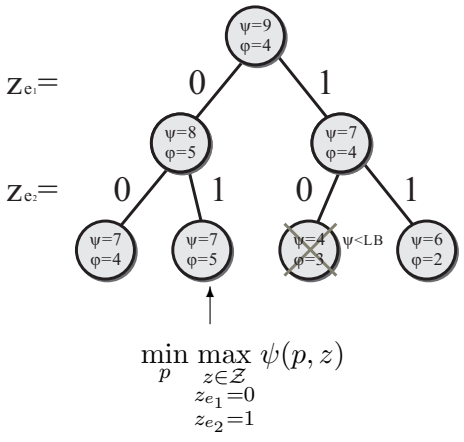


Figure 1: A part of B&B tree. ϕ, ψ are short for $\phi(z')$ and $\psi(p', z')$ respectively. For each node, some edges of the parse tree are fixed. All parse trees that satisfy the fixed edges compose the subset of $S \subseteq \mathcal{Z}$. A min-max problem is solved to get the upper bound and lower bound of the optimal parse tree over S. Once the upper bound ψ is less than LB, the node is removed safely.

e and the other fixes $z_e = 0$. We can explore the search space $\{z | z_e \in \{0, 1\}\}$ by traversing the B&B tree in breadth first order.

Let $S \subseteq \mathcal{Z}$ be subspace of parse trees satisfying the constraint, i.e., in the branch of the node. For each node in B&B tree, we solve

$$p', z' = \arg\min_{p} \max_{z \in S} \psi(p, z)$$

to get the upper bound and lower bound of the best parse tree in S. A global lower bound LB is maintained which is the maximum of all obtained lower bounds. If the upper bound of the current node is lower than the global lower bound, the node can be pruned from the B&B tree safely. An example is shown in Figure 1.

When the upper bound is not tight: $\psi > LB$, we need to choose a good branching variable to generate the child nodes. Let $G(z') = \psi(p', z') - \phi(z')$ denote the gap between the upper bound and lower bound. This gap is actually the accumulated gaps of all factors c. Let G_c be the gap of c

$$G_c = \begin{cases} g_c(p'_c, z') - \phi_c z'_c & \text{if } \phi_c \geq 0 \\ h_c(p'_c, z') - \phi_c z'_c & \text{if } \phi_c < 0 \end{cases}$$

We choose the branching variable heuristically: for each edge e, we define its gap as the sum of the gaps of factors that contain it

$$G^e = \sum_{c, e \in c} G_c$$

The edge with the maximum gap is selected as the branching variable.

Suppose there are N nodes on a level of B&B tree, and correspondingly, we get N branching variables, among which, we choose the one with the highest lower bound as it likely reaches the optimal value faster.

3.4 Lower Bound Initialization

A large lower bound is critical for efficient pruning. In this section, we discuss an alternative way to initialize the lower bound LB. We apply the similar trick to get the lower bound function of $\phi(z)$.

Similar to Eq(8), for $\phi_c \geq 0$, we have

$$\begin{aligned} \phi_c z_c &= \max\{\phi_c \left(\sum_{e \in c} z_e - (r_c - 1) \right), 0\} \\ &= \max\{\sigma_c(z), 0\} \end{aligned}$$

Using the fact that

$$\max\{a^j\} = \max_{p \in \Delta} \sum_j p^j a^j$$

we have

$$\begin{aligned} \phi_c z_c &= \max_{p_c \in \Delta} p_c^1 \sigma_c(z) + p_c^2 \cdot 0 \\ &= \max_{p_c} h_c(p_c, z) \end{aligned}$$

For $\phi_c < 0$, we have

$$\begin{aligned} \phi_c z_c &= \max_{e \in c} \{\phi_c z_e\} \\ &= \max_{p_c \in \Delta} \sum_{e \in c} p_c^e \phi_c z_e \\ &= \max_{p_c} g_c(p_c, z) \end{aligned}$$

Put the two cases together, we get the lower bound function

$$\pi(p, z) = \sum_{c, \phi_c \geq 0} h_c(p_c, z) + \sum_{c, \phi_c < 0} g_c(p_c, z)$$

Algorithm 1 Branch and Bound based parsing

Require: $\{\phi_c\}$
Ensure: Optimal parse tree z^*
  Solve $p^*, z^* = \arg\max_{p,z} \pi(p, z)$
  Initialize $\mathcal{S} = \{\mathcal{Z}\}$, $LB = \pi(p^*, z^*)$
  while $\mathcal{S} \neq \emptyset$ **do**
    Set $\mathcal{S}' = \emptyset$ {nodes that survive from pruning}
    foreach $S \in \mathcal{S}$
      Solve $\min_p \max_z \psi(p, z)$ to get LB_S, UB_S
    $LB = \max\{LB, LB_{S \in \mathcal{S}}\}$, update z^*
    foreach $S \in \mathcal{S}$, add S to $\mathcal{S}'$, if $UB_S > LB$
    Select a branching variable z_e.
    Clear $\mathcal{S} = \emptyset$
    foreach $S \in \mathcal{S}'$
      Add $S_1 = \{z | z \in S, z_e = 1\}$ to $\mathcal{S}$
      Add $S_2 = \{z | z \in S, z_e = 0\}$ to $\mathcal{S}$.
  end while

For any $p, p_c \in \Delta, z \in \mathcal{Z}$

$$\pi(p, z) \leq \phi(z)$$

$\pi(p, z)$ is not concave, however, we could alternatively optimize z and p to get a good approximation, which provides a lower bound for $\phi(z^*)$.

3.5 Summary

We summarize our B&B algorithm in Algorithm 1.

It is worth pointing out that so far in the above description, we have used the assumption that the backbone DP uses first order models, however, the backbone DP can be the second or third order version. The difference is that, for higher order DP, higher order factors such as adjacent siblings, grandchildren are directly handled as local factors.

In the worst case, all the edges are selected for branching, and the complexity grows exponentially in sentence length. However, in practice, it is quite efficient, as we will show in the next section.

4 Experiments

4.1 Experimental Settings

The datasets we used are the English Penn Tree Bank (PTB) and Chinese Tree Bank 5.0 (CTB5). We use the standard train/develop/test split as described in Table 1.

We extracted dependencies using Joakim Nivre's Penn2Malt tool with standard head rules: Yamada and Matsumoto's (Yamada and Matsumoto, 2003)

	Train	*Develop*	*Test*
PTB	sec. 2-21	sec. 22	sec. 23
CTB5	sec. 001-815 1001-1136	sec. 886-931 1148-1151	sec. 816-885 1137-1147

Table 1: Data split in our experiment

for English, and Zhang and Clark's (Zhang and Clark, 2008) for Chinese. Unlabeled attachment score (UAS) is used to evaluate parsing quality[1]. The B&B parser is implemented with C++. All the experiments are conducted on the platform Intel Core i5-2500 CPU 3.30GHz.

4.2 Baseline: DP Based Second Order Parser

We use the dynamic programming based second order parser (Carreras, 2007) as the baseline. Averaged structured perceptron (Collins, 2002) is used for parameter estimation. We determine the number of iterations on the validation set, which is 6 for both corpora.

For English, we train the POS tagger using linear chain perceptron on training set, and predict POS tags for the development and test data. The parser is trained using the automatic POS tags generated by 10 fold cross validation. For Chinese, we use the gold standard POS tags.

We use five types of features: unigram features, bigram features, in-between features, adjacent sibling features and outermost grand-child features. The first three types of features are firstly introduced by McDonald et al. (2005a) and the last two types of features are used by Carreras (2007). All the features are the concatenation of surrounding words, lower cased words (English only), word length (Chinese only), prefixes and suffixes of words (Chinese only), POS tags, coarse POS tags which are derived from POS tags using a simple mapping table, distance between head and modifier, direction of edges. For English, we used 674 feature templates to generate large amounts of features, and finally got $86.7M$ non-zero weighted features after training. The baseline parser got 92.81% UAS on the testing set. For Chinese, we used 858 feature templates, and finally got $71.5M$ non-zero weighted features after training. The baseline parser got 86.89% UAS on the testing set.

4.3 B&B Based Parser with Non-local Features

We use the baseline parser as the backbone of our B&B parser. We tried different types of non-local features as listed below:

- All grand-child features. Notice that this feature can be handled by Koo's second order model (Koo and Collins, 2010) directly.
- All great grand-child features.
- All sibling features: all the pairs of edges with common head. An example is shown in Figure 2.
- All tri-sibling features: all the 3-tuples of edges with common head.
- Comb features: for any word with more than 3 consecutive modifiers, the set of all the edges from the word to the modifiers form a comb.[2]
- Hand crafted features: We perform cross validation on the training data using the baseline parser, and designed features that may correct the most common errors. We designed 13 hand-craft features for English in total. One example is shown in Figure 3. For Chinese, we did not add any hand-craft features, as the errors in the cross validation result vary a lot, and we did not find general patterns to fix them.

4.4 Implementation Details

To speed up the solution of the min-max subproblem, for each node in the B&B tree, we initialize p with the optimal solution of its parent node, since the child node fixes only one additional edge, its optimal point is likely to be closed to its parent's. For the root node of B&B tree, we initialize $p_c^e = \frac{1}{r_c}$ for factors with non-negative weights and $p_c^1 = 0$ for

[1]For English, we follow Koo and Collins (2010) and ignore any word whose gold-standard POS tag is one of { " " : , . }. For Chinese, we ignore any word whose POS tag is *PU*.

[2]In fact, our algorithm can deal with non-consecutive modifiers; however, in such cases, factor detection (detect regular expressions like $x_1. * x_2. * \dots$) requires the longest common subsequence algorithm (LCS), which is time-consuming if many comb features are generated. Similar problems arise for sub-tree features, which may contain many non-consecutive words.

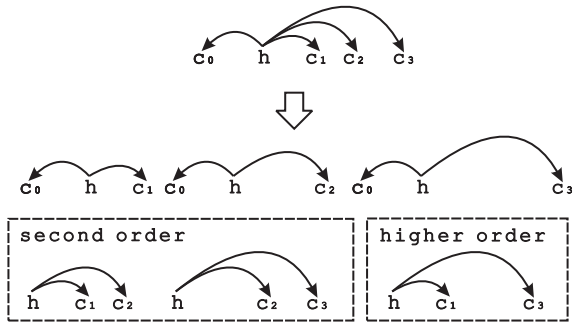


Figure 2: An example of all sibling features. Top: a sub-tree; Bottom: extracted sibling features. Existing higher order DP systems can not handle the siblings on both sides of head.

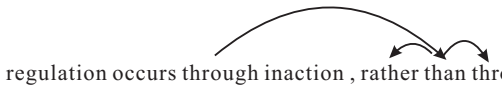


Figure 3: An example of hand-craft feature: for the word sequence *A ... rather than A*, where *A* is a preposition, the first *A* is the head of *than*, *than* is the head of *rather* and the second *A*.

negative weighted factors. Step size α is initialized with $\max_{c,\phi_c \neq 0}\{\frac{1}{|\phi_c|}\}$, as the vector p is bounded in a unit box. α is updated using the same strategy as Rush et al. (2010). Two stopping criteria are used. One is $0 \leq \psi_{old} - \psi_{new} \leq \epsilon$, where $\epsilon > 0$ is a given precision[3]. The other checks if the bound is tight: $UB = LB$. Because all features are boolean (note that they can be integer), their weights are integer during each perceptron update, hence the scores of parse trees are discrete. The minimal gap between different scores is $\frac{1}{N \times T}$ after averaging, where N is the number of training samples, and T is the iteration number for perceptron training. Therefore the upper bound can be tightened as $UB = \frac{\lfloor NT\psi \rfloor}{NT}$.

During testing, we use the pre-pruning method as used in Martins et al. (2009) for both datasets to balance parsing quality and speed. This method uses a simple classifier to select the top k candidate heads for each word and exclude the other heads from search space. In our experiment, we set $k = 10$.

[3]we use $\epsilon = 10^{-8}$ in our implementation

System	*PTB*	*CTB*
Our baseline	92.81	86.89
B&B +all grand-child	92.97	87.02
+all great grand-child	92.78	86.77
+all sibling	93.00	87.05
+all tri-sibling	92.79	86.81
+comb	92.86	86.91
+hand craft	92.89	N/A
+all grand-child + all sibling + comb + hand craft	**93.17**	**87.25**
3rd order re-impl.	93.03	87.07
TurboParser (reported)	92.62	N/A
TurboParser (our run)	92.82	86.05
Koo and Collins (2010)	93.04	N/A
Zhang and McDonald (2012)	93.06	86.87
Zhang and Nivre (2011)	92.90	86.00
System integration		
Bohnet and Kuhn (2012)	93.39	87.5
Systems using additional resources		
Suzuki et al. (2009)	93.79	N/A
Koo et al. (2008)	93.5	N/A
Chen et al. (2012)	92.76	N/A

Table 2: Comparison between our system and the-state-of-art systems.

4.5 Main Result

Experimental results are listed in Table 2. For comparison, we also include results of representative state-of-the-art systems. For the third order parser, we re-implemented Model 1 (Koo and Collins, 2010), and removed the longest sentence in the CTB dataset, which contains 240 words, due to the $O(n^4)$ space complexity [4]. For ILP based parsing, we used TurboParser[5], a speed-optimized parser toolkit. We trained full models (which use all grandchild features, all sibling features and head bigram features (Martins et al., 2011)) for both datasets using its default settings. We also list the performance in its documentation on English corpus.

The observation is that, the all-sibling features are most helpful for our parser, as some good sibling features can not be encoded in DP based parser. For example, a matched pair of parentheses are always siblings, but their head may lie between them. An-

[4]In fact, Koo's algorithm requires only $O(n^3)$ space. Our implementation is $O(n^4)$ because we store the feature vectors for fast training.

[5]http://www.ark.cs.cmu.edu/TurboParser/

other observation is that all great grandchild features and all tri-sibling features slightly hurt the performance and we excluded them from the final system.

When no additional resource is available, our parser achieved competitive performance: 93.17% Unlabeled Attachment Score (UAS) for English at a speed of 177 words per second and 87.25% for Chinese at a speed of 97 words per second. Higher UAS is reported by joint tagging and parsing (Bohnet and Nivre, 2012) or system integration (Bohnet and Kuhn, 2012) which benefits from both transition based parsing and graph based parsing. Previous work shows that combination of the two parsing techniques can learn to overcome the shortcomings of each non-integrated system (Nivre and McDonald, 2008; Zhang and Clark, 2008). System combination will be an interesting topic for our future research. The highest reported performance on English corpus is 93.79%, obtained by semi-supervised learning with a large amount of unlabeled data (Suzuki et al., 2009).

4.6 Tradeoff Between Accuracy and Speed

In this section, we study the trade off between accuracy and speed using different pre-pruning setups. In Table 3, we show the parsing accuracy and inference time in testing stage with different numbers of candidate heads k in pruning step. We can see that, on English dataset, when $k \geq 10$, our parser could gain $2-3$ times speedup without losing much parsing accuracy. There is a further increase of the speed with smaller k, at the cost of some accuracy. Compared with TurboParser, our parser is less efficient but more accurate. Zhang and McDonald (2012) is a state-of-the-art system which adopts cube pruning for efficient parsing. Notice that, they did not use pruning which seems to increase parsing speed with little hit in accuracy. Moreover, they did labeled parsing, which also makes their speed not directly comparable.

For each node of B&B tree, our parsing algorithm uses projected sub-gradient method to find the saddle point, which requires a number of calls to a DP, hence the efficiency of Algorithm 1 is mainly determined by the number of DP calls. Figure 4 and Figure 5 show the averaged parsing time and number of calls to DP relative to the sentence length with different pruning settings. Parsing time grows smoothly when sentence length ≤ 40. There is some fluctuation for the long sentences. This is because there are very few sentences for a specific long length (usually 1 or 2 sentences), and the statistics are not stable or meaningful for the small samples.

	PTB		*CTB*	
System	*UAS*	*w/s*	*UAS*	*w/s*
Ours (no prune)	93.18	52	87.28	73
Ours ($k = 20$)	93.17	105	87.28	76
Ours ($k = 10$)	93.17	177	87.25	97
Ours ($k = 5$)	93.10	264	86.94	108
Ours ($k = 3$)	92.68	493	85.76	128
TurboParser(full)	92.82	402	86.05	192
TurboParser(standard)	92.68	638	85.80	283
TurboParser(basic)	90.97	4670	82.28	2736
Zhang and McDonald (2012)†	93.06	220	86.87	N/A

Table 3: Trade off between parsing accuracy (UAS) and speed (words per second) with different pre-pruning settings. k denotes the number of candidate heads of each word preserved for B&B parsing. †Their speed is not directly comparable as they performs labeled parsing without pruning.

Without pruning, there are in total $132,161$ calls to parse $2,416$ English sentences, that is, each sentence requires 54.7 calls on average. For Chinese, there are $84,645$ calls for $1,910$ sentences, i.e., 44.3 calls for each sentence on average.

5 Discussion

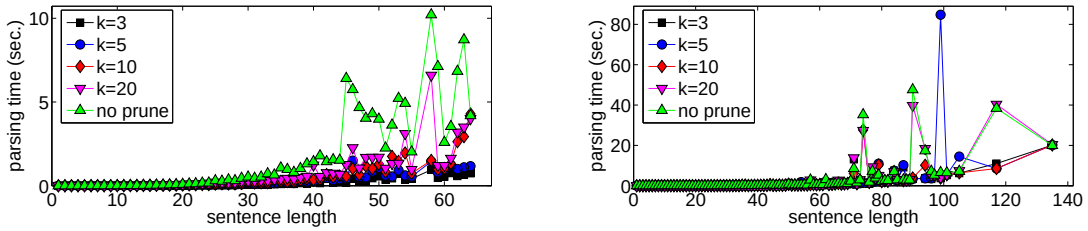
5.1 Polynomial Non-local Factors

Our bounding strategy can handle a family of non-local factors that can be expressed as a polynomial function of local factors. To see this, suppose

$$z_c = \sum_i \alpha_i \prod_{e \in E_i} z_e$$

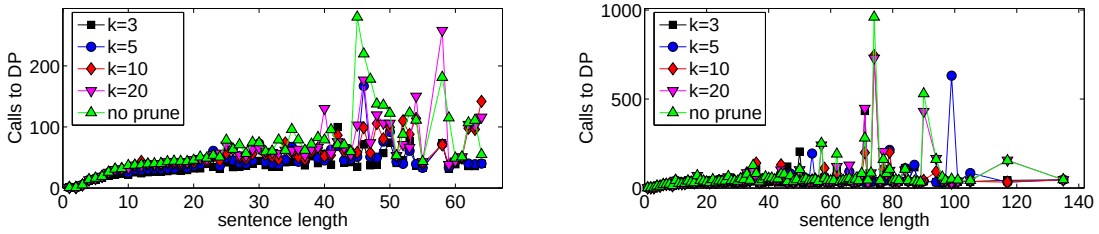
For each i, we introduce new variable $z_{E_i} = \min_{e \in E_i} z_e$. Because z_e is binary, $z_{E_i} = \prod_{e \in E_i} z_e$. In this way, we replace z_c by several z_{E_i} that can be handled by our bounding strategy.

We give two examples of these polynomial non-local factors. First is the OR of local factors: $z_c = \max\{z_e, z'_e\}$, which can be expressed by $z_c = z_e + z'_e - z_e z'_e$. The second is the factor of valency feature



(a) PTB corpus (b) CTB corpus

Figure 4 Averaged parsing time (seconds) relative to sentence length with different pruning settings, k denotes the number of candidate heads of each word in pruning step.



(a) PTB corpus (b) CTB corpus

Figure 5 Averaged number of Calls to DP relative to sentence length with different pruning settings, k denotes the number of candidate heads of each word in pruning step.

(Martins et al., 2009). Let binary variable v_{ik} indicate whether word i has k modifiers. Given $\{z_e\}$ for the edges with head i, then $\{v_{ik}|k = 1, \ldots, n-1\}$ can be solved by

$$\sum_k k^j v_{ik} = \left(\sum_e z_e\right)^j \qquad 0 \le j \le n-1$$

The left side of the equation is the linear function of v_{ik}. The right side of the equation is a polynomial function of z_e. Hence, v_{ik} could be expressed as a polynomial function of z_e.

5.2 k Best Parsing

Though our B&B algorithm is able to capture a variety of non-local features, it is still difficult to handle many kinds of features, such as the depth of the parse tree. Hence, a reranking approach may be useful in order to incorporate such information, where k parse trees can be generated first and then a second pass model is used to rerank these candidates based on more global or non-local features. In addition, k-best parsing may be needed in many applications to use parse information and especially utilize information from multiple candidates to optimize task-specific performance. We have not conducted any experiment for k best parsing, hence we only discuss the algorithm.

According to proposition 1, we have

Proposition 2. *Given p and subset $S \subseteq \mathcal{Z}$, let z^k denote the k^{th} best solution of $\max_{z \in S} \psi(p, z)$. If a parse tree $z' \in S$ satisfies $\phi(z') \ge \psi(p, z^k)$, then z' is one of the k best parse trees in subset S.*

Proof. Since z^k is the k^{th} best solution of $\psi(p, z)$, for z^j, $j > k$, we have $\psi(p, z^k) \ge \psi(p, z^j) \ge \phi(z^j)$. Since the size of the set $\{z^j|j > k\}$ is $|S| - k$, hence there are at least $|S| - k$ parse trees whose scores $\phi(z^j)$ are less than $\psi(p, z^k)$. Because $\phi(z') \ge \psi(p, z^k)$, hence z' is at least the k^{th} best parse tree in subset S. □

Therefore, we can search the k best parse trees in this way: for each sub-problem, we use DP to derive the k best parse trees. For each parse tree z, if $\phi(z) \ge \psi(p, z^k)$, then z is selected into the k best set. Algorithm terminates until the k^{th} bound is tight.

6 Conclusion

In this paper we proposed a new parsing algorithm based on a Branch and Bound framework. The motivation is to use dynamic programming to search for the bound. Experimental results on PTB and CTB5 datasets show that our method is competitive in terms of both performance and efficiency. Our method can be adapted to non-projective dependency parsing, as well as the k best MST algorithm (Hall, 2007) to find the k best candidates.

Acknowledgments

We'd like to thank Hao Zhang, Andre Martins and Zhenghua Li for their helpful discussions. We also thank Ryan McDonald and three anonymous reviewers for their valuable comments. This work is partly supported by DARPA under Contract No. HR0011-12-C-0016 and FA8750-13-2-0041. Any opinions expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

Michael Auli and Adam Lopez. 2011. A comparison of loopy belief propagation and dual decomposition for integrated CCG supertagging and parsing. In *Proc. of ACL-HLT*.

Egon Balas. 1965. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 39(4).

Bernd Bohnet and Jonas Kuhn. 2012. The best of bothworlds – a graph-based completion model for transition-based parsers. In *Proc. of EACL*.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proc. of EMNLP-CoNLL*.

Paul Calamai and Jorge Moré. 1987. Projected gradient methods for linearly constrained problems. *Mathematical Programming*, 39(1).

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proc. of EMNLP-CoNLL*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proc. of ACL*.

Wenliang Chen, Min Zhang, and Haizhou Li. 2012. Utilizing dependency language models for graph-based dependency parsing models. In *Proc. of ACL*.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proc. of ICML*.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*.

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. 2008. Efficient projections onto the l1-ball for learning in high dimensions. In *Proc. of ICML*.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: an exploration. In *Proc. of COLING*.

Keith Hall. 2007. K-best spanning tree parsing. In *Proc. of ACL*.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proc. of IWPT*.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proc. of ACL-HLT*.

Manfred Klenner and Étienne Ailloud. 2009. Optimization in coreference resolution is not needed: A nearly-optimal algorithm with intensional constraints. In *Proc. of EACL*.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proc. of ACL*.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proc. of ACL-HLT*.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proc. of EMNLP*.

Ailsa H. Land and Alison G. Doig. 1960. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520.

Andre Martins, Noah Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proc. of ACL*.

Andre Martins, Noah Smith, Mario Figueiredo, and Pedro Aguiar. 2011. Dual decomposition with many overlapping components. In *Proc. of EMNLP*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proc. of ACL*.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of HLT-EMNLP*.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proc. of ACL-HLT*.

Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization*. Springer, 2nd edition.

Sebastian Riedel and James Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proc. of EMNLP*.

Alexander M Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proc. of EMNLP*.

David Smith and Jason Eisner. 2008. Dependency parsing by belief propagation. In *Proc. of EMNLP*.

Min Sun, Murali Telaprolu, Honglak Lee, and Silvio Savarese. 2012. Efficient and exact MAP-MRF inference using branch and bound. In *Proc. of AISTATS*.

Jun Suzuki, Hideki Isozaki, Xavier Carreras, and Michael Collins. 2009. An empirical study of semi-supervised structured conditional models for dependency parsing. In *Proc. of EMNLP*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. of IWPT*.

Yue Zhang and Stephen Clark. 2008. A tale of t-wo parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proc. of EMNLP*.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proc. of EMNLP*.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proc. of ACL-HLT*.