Combining PCFG-LA Models with Dual Decomposition: A Case Study with Function Labels and Binarization

Joseph Le Roux[†], **Antoine Rozenknop**[†], **Jennifer Foster**[*]
[†] Université Paris 13, Sorbonne Paris Cité, LIPN, F-93430, Villetaneuse, France
[*] NCLT/CNGL, School of Computing, Dublin City University, Dublin 9, Ireland
joseph.leroux@lipn.fr antoine.rozenknop@lipn.fr jfoster@computing.dcu.ie

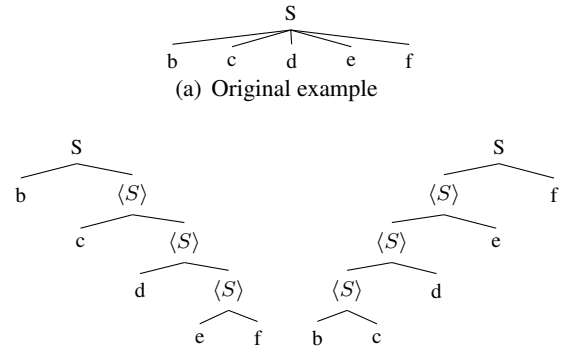
Abstract

It has recently been shown that different NLP models can be effectively combined using dual decomposition. In this paper we demonstrate that PCFG-LA parsing models are suitable for combination in this way. We experiment with the different models which result from alternative methods of extracting a grammar from a treebank (retaining or discarding function labels, left binarization versus right binarization) and achieve a labeled Parseval F-score of 92.4 on Wall Street Journal Section 23 – this represents an absolute improvement of 0.7 and an error reduction rate of 7% over a strong PCFG-LA product-model baseline. Although we experiment only with binarization and function labels in this study, there is much scope for applying this approach to other grammar extraction strategies.

1 Introduction

Because of the large amount of possibly contradictory information contained in a treebank, learning a phrase-structure-based parser implies making several choices regarding the prevalent annotations which have to be kept – or discarded – in order to guide the learning algorithm. These choices, which include whether to keep function labels and empty nodes, how to binarize the trees and whether to alter the granularity of the tagset, are often motivated empirically by parsing performance rather than by the different aspects of the language they may be able to capture.

Recently Rush et al. (2010), Martins et al. (2011) and Koo et al. (2010) have shown that Dual Decomposition or Lagrangian Relaxation is an elegant framework for combining different types of NLP tasks or for building parsers from simple *slave* processes that only check partial well-formedness. Here we propose to follow this idea, but with a different objective. We want to mix different parsers trained on different versions of a treebank each of which makes some annotation choices in order to learn more specific or richer information. We will use state-of-the-art unlexicalized probabilistic context-free grammars with latent annotations (PCFG-LA) in order to compare our approach with a strong baseline of high-quality parses. Dual Decomposition is used to mix several systems (between two and four) that may in turn be combinations of grammars, here products of PCFG-LAs (Petrov, 2010). The systems being combined make different choices with regard to i) function labels and ii) grammar binarization.



(a) Original example
(b) Left Binarized example (c) Right Binarized example

Figure 1: Binarization with markovization

Common sense would suggest that information in the form of function labels – syntactic labels such as SBJ and PRD and semantic labels such as TMP and LOC – might help in obtaining a fine-grained analysis. On the other hand, the independence hypothe-

sis on which CFGs rely and on which most popular parsers are based may be too strong to learn the dependencies between functions across the parse trees. Also, the number of parameters increases with the use of function labels and this can affect the learning process.

At first glance, binarization need not be an issue, as CFGs admit a binarized form recognizing exactly the same language. But binarization can be associated with horizontal markovization and in this case the recognized language will differ. Furthermore this can impose an unwanted emphasis on what frontier information is more relevant to learning (beginning or end of constituents). In the toy example of Figure 1, the original grammar consisting of a unique rule extracted from one tree only recognizes the string `bcdef`, while the grammar learned from the left binarized and markovized tree recognizes (among others) `bcdef` and `bdcef` and the grammar learned from the right binarized and markovized tree recognizes (among others) `bcdef` and `bcedf`.

We find that i) retaining the function labels in non-terminal categories loses its negative impact on parsing as the number of grammars increases in PCFG-LA product models, ii) the function labels themselves can be recovered with near state-of-the-art-accuracy, iii) combining grammars with and without function labels using dual decomposition is beneficial, iv) combining left and right-binarized grammars using dual decomposition also leads to better trees and, v) our best results (a Parseval labeled F-score of 92.4, a Stanford labeled attachment score (LAS) of 93.0 and a penn2malt unlabeled attachment score (UAS) of 94.3 on Section 23 of the Wall Street Journal) are obtained by combining three grammars which encode different function label/binarization decisions.

The paper is organized as follows. § 2 reviews related work. § 3 presents approximate PCFG-LA parsers as linear models, while § 4 shows how we can use dual decomposition to derive an algorithm for combining these models. Experimental results are presented and discussed in § 5.

2 Related Work

Parser Model Combination It is well known that improved parsing performance can be achieved by leveraging the alternative perspectives provided by several parsing models rather than relying on just one. Examples are parser co-training (Steedman et al., 2003; Sagae and Tsujii, 2007), voting over phrase structure constituents or dependency arcs (Henderson and Brill, 1999; Sagae and Tsujii, 2007; Surdeanu and Manning, 2010), dependency parsing stacking (Nivre and McDonald, 2008), product model PCFG-LA parsing (Petrov, 2010), using dual decomposition to combine dependency and phrase structure models (Rush et al., 2010) or several non-projective dependency parsing models (Koo et al., 2010; Martins et al., 2011), and using expectation propagation, a related approach to dual decomposition, to combine lexicalized, unlexicalized and PCFG-LA models (Hall and Klein, 2012). In this last example, the models must factor in the same way: in other words, the grammars must use the same binarization scheme. In our study, we employ PCFG-LA product models with dual decomposition, and we relax the constraints on factorization, as we require only a loose coupling of the models.

Function Label Parsing Although function labels have been available in the Penn Treebank (PTB) for almost twenty years (Marcus et al., 1994), they have been to a large extent overlooked in English parsing research — most studies that report parsing results on Section 23 of the Wall Street Journal (WSJ) use parsing models that are trained on a version of the WSJ trees where the function labels have been removed. Notable exceptions are Merlo and Musillo (2005) and Gabbard et al. (2006) who each trained a parsing model on a version of the PTB with function labels intact. Gabbard et al. (2006) found that parsing accuracy was not affected by keeping the function labels. There have also been attempts to use machine learning to recover the function labels post-parsing (Blaheta and Charniak, 2000; Chrupala et al., 2007). We recover function labels as part of the parsing process, and use dual decomposition to combine parsing models with and without function labels. We are not aware of any other work that leverages the benefits of both types of models.

Grammar Binarization Matsuzaki et al. (2005) compare binarization strategies for PCFG-LA parsing, and conclude that the differences between them have a minor effect on parsing accuracy as the num-

ber of latent annotations increases beyond two. Hall and Klein (2012) are forced to use head binarization when combining their lexicalized and unlexicalized parsers. Dual decomposition allows us to combine models with different binarization schemes.

3 Approximation of PCFG-LAs as Linear Models

In this section, we explain how we can use PCFG-LAs to devise linear models suitable for the dual decomposition framework.

3.1 PCFG-LA

Let us recall that PCFG-LAs are defined as tuples $G = (\mathcal{N}, \mathcal{T}, \mathcal{H}, \mathcal{R}_\mathcal{H}, S, p)$ where:

- $\mathcal{N}$ is a set of observed non-terminals, among which S is the distinguished initial symbol,
- $\mathcal{T}$ is a set of terminals (words),
- $\mathcal{H}$ is a set of *latent annotations* or hidden states,
- $\mathcal{R}_\mathcal{H}$ is a set of annotated rules, of the form $a[h_1] \rightarrow b[h_2]\ c[h_3]$ for internal rules[1] and $a[h_1] \rightarrow w$ for lexical rules. Here $a, b, c \in \mathcal{N}$ are non-terminals, $w \in \mathcal{T}$ is a terminal and $h_1, h_2, h_3 \in \mathcal{H}$ are latent annotations. Following Cohen et al. (2012) we also define the set of skeletal rules $\mathcal{R}$, in other words, rules without hidden states, of the form $a \rightarrow b\ c$ or $a \rightarrow w$.
- $p : \mathcal{R}_\mathcal{H} \rightarrow \mathbb{R}_{\geq 0}$ defines the probabilities associated with rules conditioned on their left-hand side. Like Petrov and Klein (2007), we impose that the initial symbol S has only one latent annotation. In other words, among rules with S on the left-hand side, only those of the form $S[0] \rightarrow \gamma$ are in $\mathcal{R}_\mathcal{H}$.

With such a grammar G we can define probabilities over trees in the following way. We will consider two types of trees, annotated trees and skeletal trees. An annotated tree is a sequence of rules from $\mathcal{R}_\mathcal{H}$, while a skeletal tree is a sequence of skeletal rules from R. An annotated tree $T_\mathcal{H}$ is obtained by left-most derivation from $S[0]$. Its probability is:

$$p(T_\mathcal{H}) = \prod_{r \in T_\mathcal{H}} p(r) \tag{1}$$

We define a projection ρ from annotated trees to skeletal trees. $\rho(T_\mathcal{H})$ is a tree T isomorphic to $T_\mathcal{H}$ with the same terminal and non-terminal symbols labeling nodes, without hidden states. The probability of a skeletal tree T is a sum of the probabilities of all annotated trees that admit T as their projection.

$$p(T) = \sum_{T_\mathcal{H} \in \rho^{-1}(T)} \prod_{r \in T_\mathcal{H}} p(r) \tag{2}$$

PCFG-LA parsing amounts to, given a sequence of words, finding the most probable skeletal tree with this sequence as its yield according to a grammar G:

$$T^* = \arg\max_T \sum_{T_\mathcal{H} \in \rho^{-1}(T)} \prod_{r \in T_\mathcal{H}} p(r) \tag{3}$$

Because of this alternation of sum and products, the parsing problem is intractable. Moreover, the PCFG-LAs do not belong to the family of linear models that are assumed in the Lagrangian framework of (Rush and Collins, 2012). We now turn to approximations for the parsing problem in order to address both issues.

3.2 Variational Inference and MaxRule

Variational inference is a common technique to approximate a probability distribution p with a cruder one q, as close as possible to the original one, by minimizing the Kullback-Liebler divergence between the two – see for instance (Smith, 2011), chapter 5 for an introduction. Matsuzaki et al. (2005) showed that one can easily find such a cruder distribution for PCFG-LAs and demonstrated experimentally that this approximation gives good results. More precisely, they find a PCFG that only recognizes the input sentence where the probabilities $q(r_s)$ of the rules are set according to their marginal probabilities in the original PCFG-LA parse forest. The parameters r_s are skeletal rules with *span information*. Distribution q is defined in Figure 2.

Other approximations are possible. In particular, Petrov and Klein (2007) found that normalizing by the forest probability (in other words the inside probability of the root node) give better exper-

[1] For brevity and without loss of generality, we omit unary and n-ary rules, as PCFG-LA admit a Chomsky normal form.

$$
\begin{aligned}
score(a \to b\ c, i, j, k) &= \sum_{x,y,z \in \mathcal{H}} P_{\text{out}}^{i,k}\big(a[x]\big) \cdot p\big(\ a[x] \to b[y]\ c[z]\ \big) \cdot P_{\text{in}}^{i,j}\big(b[y]\big) \cdot P_{\text{in}}^{j,k}\big(c[z]\big) \\
norm(a \to b\ c, i, j, k) &= \sum_{x \in \mathcal{H}} P_{\text{in}}^{i,k}\big(a[x]\big) \cdot P_{\text{out}}^{i,k}\big(a[x]\big) \\
score(a \to w, i) &= \sum_{x \in \mathcal{H}} P_{\text{out}}^{i,i}\big(a[x]\big) \cdot p\big(a[x] \to w\big) \\
norm(a \to w, i) &= \sum_{x \in \mathcal{H}} P_{\text{in}}^{i,i}\big(a[x]\big) \cdot P_{\text{out}}^{i,i}\big(a[x]\big) \\
q(r_s) &= \left[\frac{score(r_s)}{norm(r_s)} \text{ (Variational Inference)}\right] \text{ or } \left[\frac{score(r_s)}{P_{\text{in}}^{0,n}\left(S[0]\right)} \text{ (MaxRule-Product)}\right]
\end{aligned}
$$

Figure 2: Variational Inference for PCFG-LA. P_{in} and P_{out} denote inside and outside probabilities.

imental results although its interpretation as variational inference is still unclear. This approximation is called MaxRule-Product and amounts to replacing the $norm$ function (see Figure 2).

In both cases, the probability of a skeletal tree now becomes a simple product of parameters associated with anchored skeletal rules. For our purpose, the consequence is twofold:

1. The parsing problem becomes tractable by applying standard PCFG algorithms relying on dynamic programming (CKY for example).

2. Equivalent to probability, a score σ can be defined as the logarithm of the probability. The parsing problem becomes[2]:

$$
\begin{aligned}
T^* &= \arg\max_T \prod_{r_s \in T} q(r_s) \\
&= \arg\max_T \sum_{r_s \in T} \log q(r_s) \\
&= \arg\max_T \sum_{r_s \in \mathcal{F}} w_{r_s} \cdot \mathbf{1}\{r_s \in T\} \\
&= \arg\max_T \sigma(T)
\end{aligned}
$$

Thus, from a PCFG-LA we are able to define a linear model whose parameters are the log-probabilities of the rules in distribution q.

[2]We denote the parse forest of a sentence by $\mathcal{F}$ and the characteristic function of a set by $\mathbf{1}$.

3.3 Products of PCFG-LAs

Although PCFG-LA training is beyond the scope of this paper, it is worthwhile mentioning that the most common way to learn their parameters relies on Expectation-Maximization which is not guaranteed to find the optimal estimation. Fortunately, this can be partly overcome by combining grammars that only differ on the initial parameterization of the EM algorithm. The probability of a skeletal tree is the product of the probabilities assigned by each single grammar G_i.

$$
T^* = \arg\max_T \prod_{i=1}^{n} q_{G_i}(T) \tag{4}
$$

Since grammars only differ by their numerical parameters (i.e. skeletal rules are the same), inference can be efficiently implemented using dynamic programming (Petrov, 2010).

Scoring with n such grammars now becomes:

$$
\begin{aligned}
T^* &= \arg\max_T \sum_{i=1}^{n} \sum_{r \in T} \log q_{G_i}(r) &(5)\\
&= \arg\max_T \sum_{r \in T} \sum_{i=1}^{n} \log q_{G_i}(r) &(6)
\end{aligned}
$$

The distributions q_{G_i} still have to be computed independently – and possibly in parallel – but the final decoding can be performed jointly. This is still a linear model for PCFG-LA parsing, but restricted to grammars that share the same skeletal rules.

4 Dual Decomposition

In this section, we show how we derive an algorithm to work out the best parse according to a set of n grammars that do not share the exact same skeletal rules. As such, the grammars' product cannot be easily conducted inside the parser to produce and score a same and unique best tree, and we now consider a *c(ompound)-parse* as a tuple $(T_1 \dots T_n)$ of n *compatible* trees. Each grammar G_i is responsible for scoring tree T_i, and we seek to obtain the *c-parse* that maximizes the sum of the scores of its different trees. For a *c-parse* to be consistent, we have to precisely define the parts on which the trees must agree to be compatible with each other, so that we can model these as agreement constraints.

4.1 Compound Parse Consistency

Let us suppose we have a set of phrase-structure parsers trained on different versions of the same treebank. Hence, some elements in the charts will either be the same or can be mapped to each other provided an equivalence relation and we define consensus between parsers on these elements.

When the grammar is not functionally annotated, phrase-structure trees can be decomposed into a set of anchored (syntactical) categories X_s, asserting that a category X is in the tree at position[3] s. Thus, such a tree T can be described by means of a boolean vector $z(T)$ indexed by anchored labels X_s, where $z(T)_{X_s} = 1$ if X_s is in T and 0 otherwise.

We will differentiate the set of *natural* non-terminals that occur in the treebanks from the set of *artificial* non-terminals that do not occur in the treebank and are the results of a binarization with markovization. As these artificial non-terminals disappear after reversing binarization in solution trees, they do not play any role in the consensus between parsers, and we only consider natural non-terminals in the set of anchored labels.

When the grammar is functionally annotated, each label $\bar{X}$ in a tree is a pair (X, F), where X is a syntactical category and F is a function label. In this case, in order to manage the consensus with non-functional grammars, we decompose such a tree into two sets: a set of anchored categories X_s and a set of anchored function labels F_s. Thus, a tree T can be described by means of two boolean vectors:

- $z(T)$ indexed by anchored categories X_s, $z(T)_{X_s} = 1$ if there exists a function label F so that $(X, F)_s$ is in T, and 0 otherwise;
- $\zeta(T)$ indexed by anchored function labels F_s, $\zeta(T)_{F_s} = 1$ if there exists a category X so that $(X, F)_s$ is in T, and 0 otherwise.

In the present work, a compound parse $(T_1 \dots T_n)$ is said to be consistent iff every tree shares the same set of *anchored categories*, i.e. iff:

$$\forall (i,j) \in [\![1,n]\!]^2,\ z(T_i) = z(T_j)$$

4.2 Combining Parsers through Dual Decomposition

Like previous applications, we base our reasoning on the assumption that computing the optimal score with each grammar G_i can be efficiently calculated, which is the case for approximate PCFG-LA parsing. We follow the presentation of the decomposition from (Martins et al., 2011) to explain how we can combine several PCFG-LA parsers together.

For a sentence s, we want to obtain the best consistent compound parse from a set of n parsers:

$$(P): \quad \textit{find} \quad \underset{(T_1 \dots T_n) \in \mathcal{C}}{\arg\max} \sum_{p=1}^{n} \sigma_p(T_p) \tag{7}$$

$$\textit{s.t.} \ \forall (i,j) \in [\![1,n]\!]^2,\ z(T_i) = z(T_j) \tag{8}$$

where $\mathcal{C} = \mathcal{F}_1(s) \times \dots \times \mathcal{F}_n(s)$ is the product of parse forests $\mathcal{F}_i(s)$, and $\mathcal{F}_i(s)$ is the set of trees in grammar G_i whose yields are the input sentence s.

Solving this problem with an exact algorithm is intractable. While artificial nodes could be inferred using a traditional parsing algorithm based on dynamic programming (i.e. CKY), the natural nodes require a coupling of the parsers' items to enforce the fact that natural daughter nodes must be identical (or equivalent) with the same spans for all parsers. Since the debinarization of markovized rules enables the creation of arbitrarily long n-ary rules, in the worst case the number of natural daughters to check is exponential in the size of the span to infer. Even if

[3] The anchor s of a label is composed of the span (i, j), denoting that the label covers terminals of the input sentence from index i to index j. In case the grammar contains unary non-lexical rules, the anchor also discriminates the different positions in a sequence of unary rules.

we bound the length of debinarized rules, the problem is hardly tractable.

As this problem is intractable, even for approximate PCFG-LA parsing, we apply the iterate method presented in (Komodakis et al., 2007) for MRFs, also applied for joint tasks in NLP such as combined parsing and POS tagging in (Rush et al., 2010).

First, we introduce a *witness* vector u in order to simplify constraints in (8). Problem (P) can then be written in an equivalent form :

$$(P): \quad \textit{find} \quad o_P = \max_{(T_1 \dots T_n) \in \mathcal{C}} \sum_{i=1}^{n} \sigma_i(T_i) \tag{9}$$

$$\textit{s.t.} \quad \forall i \in [\![1, n]\!],\ z(T_i) = u \tag{10}$$

Next, we proceed to a Lagrangian decomposition. This decomposition is a two-step process:

Step 1 (Relaxation): the coupling constraints (10) are removed by introducing a vector of Lagrange multipliers $\Lambda_i = (\lambda_{i,X_s})_{X_s}$ for each parser i, indexed by anchored categories X_s, and writing the equivalent problem:

$$(RP): \quad \mathrm{o}_{RP} = \max_{u, T_{1\dots n}} \min_{\Lambda} f(u, T_{1\dots n}, \Lambda)$$

where:

$$f(u, T_{1\dots n}, \Lambda) = \sum_i \sigma_i(T_i) + \sum_i (z(T_i) - u) \cdot \Lambda_i$$

Intuitively, we can see the equivalence of (RP) and (P) with the following reasoning:

- whenever all constraints (10) are met, the second sum in f is nullified and $f(u, T_{1\dots n}, \Lambda) = \sum_i \sigma_i(T_i)$, which is a finite value and precisely the objective function maximized in (P);
- if there is at least one (i, X, s) such that $z(T_i)_{X_s} \neq u_{X_s}$, then the value of $\sum_i (z(T_i) - u) \cdot \Lambda_i$ can be made arbitrarily small by an appropriate choice of λ_{i,X_s}; in this case, $\min_\Lambda f(u, T_{1\dots n}, \Lambda) = -\infty$. Thus, (RP) can not reach its maximum at a point where constraints (10) are not satisfied.

Step 2 (dualization): the dual problem (LP) is obtained by permuting $\max$ and $\min$ in (RP):

$$(LP1): \quad \mathrm{o}_{LP} = \min_{\Lambda} \max_{u, T_{1\dots n}} f(u, T_{1\dots n}, \Lambda)$$

Finally, u can be removed from $(LP1)$ by adding the constraint: $\sum_i \Lambda_i = 0$. As a matter of fact, one can see that if this constraint is not matched, $\max_{u, T_{1\dots n}} f(u, T_{1\dots n}, \Lambda) = +\infty$ and $(LP1)$ can not reach its minimum on such a point. We can now find the maximum of f by maxing each T_i independently of each other. The dual problem becomes:

$$(LP): \quad \mathrm{o}_{LP} = \min_{\Lambda} \sum_{i=1}^{n} \max_{T_i \in \mathcal{F}_i} (\sigma_i(T_i) + z(T_i) \cdot \Lambda_i)$$

$$s.t. \quad \sum_i \Lambda_i = 0$$

Minimization in (LP) can be solved iteratively using the projected subgradient method. Finding a subgradient amounts to computing the optimal solution (Rush and Collins, 2012) for each of the n subproblems (the *slave* problems in the terminology of (Martins et al., 2011) and (Komodakis et al., 2007)) which can be done efficiently, by incorporating the calculation of the penalties in the parsing algorithm, and in parallel. Until the agreement constraints are met (or a maximal number of iterations τ), the Lagrangian multipliers are updated according to the deviations from the average solutions (i.e. updates are zeros for a natural span if the parsers agree on it). This leads to Algorithm 1.

It should be noted that the DP charts are built and pruned during the first iteration only ($t = 0$); further iterations do not require recreating the DP chart, which is memory intensive and time consuming, nor recomputing the approximate distribution for variational inference. As DP on the pruned charts is a fast process, the bottleneck of the algorithm still is in the first calculation of slave solutions.

The stepsize sequence $(\alpha_t)_{0 \leq t}$ must be diminishing and non-summable, that is to say: $\forall t, \alpha_t \geq 0$, $\lim_{t\to\infty} \alpha_t = 0$ and $\sum_{t=0}^{\infty} \alpha_t = \infty$. In practice, we set $\alpha_t = \frac{1}{1+c(t)}$ where $c(t)$ is the number of times the objective function o_P has increased since iterations began.

Solving $(\mathbf{P})$: it is easy to see that o_{LP} is an upper bound of o_P, but we do not necessarily have

Algorithm 1 Find best compound parse with constraints on natural spans

Require: n parsers $\{p_i\}_{1 \leq i \leq n}$
  for all i, syntactical category X, anchor s **do**
    $\lambda^{(0)}_{i,X_s} = 0$
  end for
  for $t = 0 \rightarrow \tau$ **do**
    for all parsers p_i **do**
    $T_i^{(t)} \leftarrow \arg\max_{T \in \mathcal{F}_i} \left(\sigma_i(T) + z(T) \cdot \Lambda_i^{(t)}\right)$
    end for
    for all parsers p_i **do**
      $\Delta_i^{(t)} \leftarrow \alpha_t \left(z\left(T_i^{(t)}\right) - \frac{\sum_{1 \leq j \leq n} z\left(T_j^{(t)}\right)}{n}\right)$
      $\Lambda_i^{(t+1)} \leftarrow \Lambda_i^{(t)} + \Delta_i^{(t)}$
    end for
    if $\Delta_i^{(t)} = 0$ for all i **then**
      Exit loop
    end if
  end for
  return $(T_1^{(\tau)}, \cdots, T_n^{(\tau)})$

strong duality (i.e. $o_{LP} = o_P$) due to the facts that parse forests are discrete sets. Furthermore, they get pruned independently of each other. Thus, the algorithm is not guaranteed to find a t such that $z(T_i^{(t)})$ is the same for every parser i. However – see (Koo et al., 2010) – if it does reach such a state, then we have the guarantee of having found an exact solution of the primal problem (P). We show in the experiments that this occurs very frequently.

5 Experiments

5.1 Experimental Setup

We perform our experiments on the WSJ sections of the PTB with the usual split: sections 2 to 21 for training, section 23 for testing, and we run benchmarks on section 22. `evalb` is used for evaluation.

We use the LORG parser modified with Algorithm 1. [4] All grammars are trained using 6 split/merge EM cycles. For the handling of unknown words, we removed all words occurring once in the training set and replaced them by their morphological signature (Attia et al., 2010). Grammars for products are obtained by training with 16 random seeds for each setting. We use the approximate algorithm MaxRule-Product (Petrov and Klein, 2007).

The basic settings are a combination of the two following parameters:

left or right binarization: we conjecture that this affects the quality of the parsers by impacting the recognition of left and right constituent frontiers. We set vertical markovization to 1 (no parent annotation) and horizontal markovization to 0 (we drop all left/right annotations).

with or without functional annotations: in particular when non-terminals are annotated with multiple functions, all are kept.

5.2 Products of Grammars

We first evaluate each setting on its own before combining them. We test the 4 different settings on the development set, using a single grammar or a product of n grammars. Results are reported on Figure 3. We can see that right binarization performs better than left binarization. Contrary to the results of Gabbard et al. (2006), function labels are detrimental for parsing performance for one grammar only. However, they do not penalize performance when using the product model with 8 grammars or more.

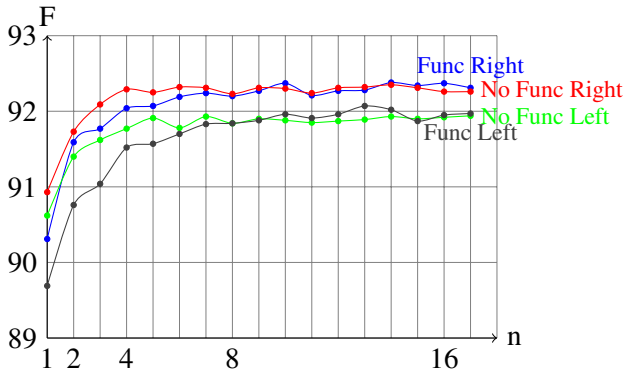


Figure 3: F1 for products of n grammars on the dev. set

EM is not guaranteed to find the optimal model and the problem is made harder by the increased number of parameters. Product models effectively alleviate this *curse of dimensionality* by letting some models compensate for the errors made by others.

On the other hand, as differences between left and right binarization settings remain over all product sizes, right binarization seems more useful on its own. The first part of Table 1 gives F-score and

[4] The LORG parser is available at `https://github.com/CNGLdlab/LORG-Release` and the modification at `https://github.com/jihelhere/LORG-Release/tree/functional_c11`.

Exact Match results of the product models with 16 grammars on the development set.

5.3 Combinations with Dual Decomposition

We now turn to a series of experiments combining product models of 16 grammars. In all these experiments, we set the maximum number of iterations in Algorithm 1 to 1000. The system then returns the first element of the c-parse. We first try to combine two settings in four different combinations:

DD Right Bin the two right-binarized systems – with and without functions – the system returns the function-labeled parse;

DD Left Bin the two left-binarized systems – with and without functions – the system returns the function-labeled parse;

DD Func the two systems with functions – left and right binarization – the system returns the right-binarized parse;

DD No Func the two systems without functions – left and right binarization – the system returns the right-binarized parse;

Results are in the second part of Table 1. Unsurprisingly, the best configuration is the one combining the two best product systems (with right binarization) but all combined systems perform better than their single components.

Setting	F	EX
No Func Right	92.26	42.97
No Func Left	91.92	42.91
Func Right	92.37	43.35
Func Left	91.95	43.15
DD Right Bin	92.71	44.44
DD Left Bin	92.23	43.97
DD Func	92.51	44.79
DD No Func	92.52	44.08
DD3	92.86	45.03
DD4	92.82	45.14

Table 1: Parse evaluation on development set.

We also combine 3 and 4 parsers to see if combining the above **DD Right Bin** setting with information that could improve the recognition of beginning of constituents can be helpful. We have 2 settings:

DD3 The 2 right-binarized parsers combined with the left binarized parser without functions,

DD4 The 4 parsers together.

In both cases the system returns the right-binarized function annotated parse. The results are shown in the last part of Table 1. These 2 new configurations give similar F-scores, better than all 2-parser configurations.

We conclude from these results that left-binarization and right-binarization capture different linguistic aspects, even in the case of heavy horizontal markovization, and that the method we propose enables a practical integration of these models.

Table 2 shows for each setting how often the systems agree before 1000 iterations of Algorithm 1. As one might expect, the more diverse the systems are, the lower the rate of agreement.

Setting	Rate
DD Right Bin	99.24
DD Left Bin	99.12
DD Func	98.53
DD No Func	99.12
DD3	96.18
DD4	94.53

Table 2: Rate of certificates of optimality on the dev set.

5.4 Evaluation of Function Labeling

We also evaluate the quality of the function labels. We compare the results obtained directly from the parser output with results obtained with Funtag, a state-of-the-art functional tagger that is applied on parser output, using a *gold model* trained on sections 02 to 21 of the WSJ (Chrupala et al., 2007).

Setting	SYSTEM FUN	FUNTAG
No Func Right	–	90.41
No Func Left	–	90.26
Func Right	89.61	90.37
Func Left	89.29	90.40
DD Right Bin	89.50	90.38
DD Left Bin	89.11	90.31
DD Func	89.54	90.49
DD No Func	–	90.36
DD3	89.48	90.42
DD4	89.57	90.45

Table 3: Function labeling F1 on development set.

The results are shown in Table 3. First, we can see that the parser output is always outperformed by Funtag. This is expected from a context-free parser

that has a limited domain of locality with strong independence constraints, compared to a voted-SVM classifier that can rely on arbitrarily rich features. Second, the quality of the Funtag prediction seems to be influenced by the fact that parser already handle functions and by the accuracy of the parser (Parseval F-score). This is because we use a model trained on the gold reference and so the closer the parser output is from the reference, the better the prediction. On the other hand, this is not the case with parser predicted functions, where the best system is the right-binarized product model with functions, with very similar performance obtained by the combinations consisting of 2 function parsers, settings **DD Func** and **DD4**. This tends to indicate that the constraints we have set to define consistencies in c-parses, focusing on syntactical categories, do not help in retrieving better function labels. This suggests some possible further improvements where parsers with functional annotations should be forced to agree on these too.

5.5 Evaluation of Dependencies

Setting	Stanford		LTH		p2m
	LAS	UAS	LAS	UAS	UAS
Func Right	92.18	94.32	89.51	93.92	94.2
No Func Right	92.03	94.47	65.31	92.22	94.2
Func Left	91.86	94.06	89.28	93.75	93.9
No Func Left	91.83	94.29	65.33	92.18	94.1
DD Right Bin	92.56	94.60	89.81	94.17	94.5
DD Left Bin	92.01	94.38	89.62	94.05	94.2
DD Func	92.19	94.36	89.67	94.06	94.2
DD No Func	92.19	94.57	65.44	92.37	94.3
DD3	92.77	94.79	90.04	94.33	94.5
DD4	92.59	94.62	89.95	94.24	94.4

Table 4: Dependency accuracies on the dev set

Dependency-based evaluation of phrase structure parser output has been used in recent years to provide a more rounded view on parser performance and to compare with direct dependency parsers (Cer et al., 2010; Petrov et al., 2010; Nivre et al., 2010; Foster et al., 2011; Petrov and McDonald, 2012). We evaluate our various parsing models on their ability to recover three types of dependencies: basic Stanford dependencies (de Marneffe and Manning, 2008)[5], LTH dependencies (Johansson and Nugues, 2007)[6] and *penn2malt* dependencies.[7] The latter are a simpler version of the LTH dependencies but are still used when reporting unlabeled attachment scores for dependency parsing.

The results, shown in Table 4, mirror the constituency evaluation results in that the dual decomposition results tend to outperform the basic product model results, and combining three or four grammars using dual decomposition yields the highest scores. The differences between the **Func** and **No Func** results highlight an important difference between the Stanford and LTH dependency schemes. The tool used to produce Stanford dependencies has been designed to work with phrase structure trees that do not contain function labels. In contrast, the LTH tool makes use of function label information in phrase structure trees. Thus, their availability results in only a moderate improvement in LAS for the Stanford dependencies and a very striking improvement for the LTH dependencies. By retaining function labels during parsing, we have shown that LTH dependencies can be recovered with a high level of accuracy without having to resort to a post-parsing function labeling step.

5.6 Test Set Results

We now evaluate our various systems on the test set (the first half of Table 5) and compare these results with state-of-the-art systems (the second half of Table 5). We present parser accuracy results, measured using Parseval F-score and *penn2malt* UAS, and, for our systems, function label accuracy for labels produced during parsing and after parsing using Funtag. We also carried out statistical significance testing[8] on the F-score differences between our various systems on the development and test sets. The results

[5]We used the latest version at the time of writing, i.e. 3.20.

[6]`nlp.cs.lth.se/software/treebank_converter`. It is recommended that LTH is used with the version of the Penn Treebank which contains the more detailed NP bracketing provided by Vadas and Curran (2007). However, to facilitate comparison with other parsers and dependency schemes, we did not use it in our experiments. We ran the converter with the *rightBranching=false* option to indicate that we are using the version without extra noun phrase bracketing.

[7]`stp.lingfil.uu.se/~nivre/research/Penn2Malt`. The English head-finding rules of Yamada and Matsumoto (2003), supplied on the website, are employed.

[8]We used Dan Bikel's `compare.pl` script which uses stratified shuffling to compute significance. We consider a p value < 0.05 to indicate a statistically significant difference.

Setting	F	UAS	Fun	Funtag
Func Right	91.73	93.9	91.02	91.88
No Func Right	91.76	93.8	–	91.80
Func Left	91.45	93.7	90.41	91.80
No Func Left	91.57	93.7	–	91.74
DD Right Bin	92.16	94.1	90.85	91.86
DD Left Bin	91.89	93.9	90.10	91.85
DD Func	92.23	94.1	91.02	91.91
DD No Func	92.09	94.0	–	91.86
DD3	92.45	94.3	90.86	91.98
DD4	92.44	94.3	90.97	92.04
(Shindo et al., 2012)	92.4			
(Zhang et al., 2009)	92.3			
(Petrov, 2010)	91.8			
(Huang, 2008)	91.7			
(Bohnet and Nivre, 2012)		93.7		

Table 5: Test Set Results: Parseval F-score, *penn2malt* UAS, Function Label Accuracy and Funtag Function Label Accuracy

are shown in Table 6.

Comparison	Dev	Test
Func Right vs. No Func Right	✗	✗
Func Left vs. No Func Left	✗	✗
Func Right vs. Func Left	✓	✗
No Func Right vs. No Func Left	✗	✗
DD Right Bin vs. Func Right	✓	✓
DD Right Bin vs. No Func Right	✓	✓
DD Left Bin vs. Func Left	✓	✓
DD Left Bin vs. No Func Left	✓	✓
DD Right Bin vs DD Left Bin	✓	✓
DD Func vs. Func Right	✗	✓
DD Func vs. Func Left	✓	✓
DD No Func vs. No Func Right	✓	✓
DD No Func vs. No Func Left	✓	✓
DD Func vs. DD No Func	✗	✗
DD3 vs. DD Right Bin	✗	✓
DD3 vs. No Func Left	✓	✓
DD3 vs. DD Func	✓	✓
DD4 vs. DD. Right Bin	✗	✓
DD4 vs. DD. Left Bin	✓	✓
DD4 vs. DD Func	✓	✓
DD4 vs. DD3	✗	✗

Table 6: Statistical Significance Testing

We measured the performance of DD4 on the test set. It is approximately 3 times slower than the slowest product model (left binarization with function labels) and 7 slower than the fastest one (right binarization without function labels). This system performs on average 85.5 iterations of the DD algorithm. If we exclude the non-converging cases (5.1% of the cases), this drops to 39.4.

Finally we compare our results with systems trained and evaluated on the PTB, see the lower half of Table 5. Our product models are not different from those presented in (Petrov, 2010) and it is not surprising to see that the F-scores are similar. More interestingly our DD4 setting improves on these results and compares favorably with systems relying on richer syntactic information, such as the discriminative parser of (Huang, 2008) that makes use of non-local features to score trees and the TSG parser of (Shindo et al., 2012) that can take into account larger tree fragments: this would indicate that by combining our parsers we extend the domain of locality, horizontally with binarization schemes and vertically with function labels. Our system also performs better than the combination system presented in (Zhang et al., 2009) that only relies on material from the PTB[9] but a more detailed comparison is difficult: this system does not use products of latent models and more generally their method is orthogonal to ours. We also include for comparison state-of-the-art dependency parsing results (Bohnet and Nivre, 2012).

6 Conclusion

We presented an algorithm and a set of experiments showing that grammar extraction strategies can be combined in an elegant way and give state-of-the-art results when applied to high-quality phrase-based parsers. As well as repeating these experiments for languages which rely more on function annotation, we also plan to apply our method to other types of annotations, e.g. more linguistically motivated binarization strategies or – of particular interest to us – annotation of empty elements.

Acknowledgments

We are grateful to the reviewers for their helpful comments. We also thank Joachim Wagner for providing feedback on an early version of the paper. This work has been partially funded by the Labex EFL (ANR/CGI).

[9]Their other system relying on the self-trained version of the BLLIP parser achieves 92.6 F1.

References

Mohammed Attia, Jennifer Foster, Deirdre Hogan, Joseph Le Roux, Lamia Tounsi, and Josef van Genabith. 2010. Handling unknown words in statistical latent-variable parsing models for Arabic, English and French. In *Proceedings of the First Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*.

Don Blaheta and Eugene Charniak. 2000. Assigning function tags to parsed text. In *Proceedings of the 1st Annual Meeting of the North American chapter of the ACL*.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465.

Daniel Cer, Marie-Catherine de Marneffe, Daniel Jurafsky, and Christopher D. Manning. 2010. Parsing to Stanford Dependencies: Trade-offs between speed and accuracy. In *Proceedings of LREC*.

Grzegorz Chrupala, Nicolas Stroppa, Josef van Genabith, and Georgiana Dinu. 2007. Better training for function labeling. In *Proceedings of the 2007 Conference on Recent Advances in Natural Language Processing (RANLP)*.

Shay B. Cohen, Karl Stratos, Michael Collins, Dean P. Foster, and Lyle Ungar. 2012. Spectral learning of latent-variable PCFGs. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL'12)*.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *Proceedings of the COLING Workshop on Cross-Framework and Cross-Domain Parser Evaluation*.

Jennifer Foster, Ozlem Cetinoglu, Joachim Wagner, Joseph Le Roux, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. 2011. From news to comment: Resources and benchmarks for parsing the language of web 2.0. In *Proceedings of IJCNLP*.

Ryan Gabbard, Mitchell Marcus, and Seth Kulick. 2006. Fully parsing the penn treebank. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the ACL*, pages 184–191.

David Hall and Dan Klein. 2012. Training factored PCFGs with expectation propagation. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing*, pages 649–652.

John C. Henderson and Eric Brill. 1999. Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of the 1999 Conference on Empirical Methods in Natural Language Processing*, pages 187–194.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pages 586–594.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In Joakim Nivre, Heiki-Jaan Kaalep, Kadri Muischnek, and Mare Koit, editors, *Proceedings of NODALIDA 2007*, pages 105–112.

Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. 2007. MRF optimization via dual decomposition: Message-passing revisited. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

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*.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Proceedings of the 1994 ARPA Speech and Natural Language Workshop*, pages 114–119.

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.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 75–82.

Paola Merlo and Gabriele Musillo. 2005. Accurate function parsing. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 620–627.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-08: HLT*, pages 950–958.

Joakim Nivre, Laura Rimell, Ryan Mc Donald, and Carlos Gómez-Rodríguez. 2010. Evaluation of dependency parsers on unbounded dependencies. In *Proceedings of COLING*.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of the conference on Human Language Technologies and the conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'07)*.

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web. In *Working Notes of the SANCL Workshop (NAACL-HLT)*.

Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. 2010. Uptraining for accurate deterministic question parsing. In *Proceedings of EMNLP*.

Slav Petrov. 2010. Products of random latent variable grammars. In *Proceedings of the conference on Human Language Technologies and the conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'10)*, pages 19–27.

Alexander Rush and Michael Collins. 2012. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*, 45:305–362.

Alexander M Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11.

Kenji Sagae and Jun'ichi Tsujii. 2007. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proceedings of the CoNLL shared task session of EMNLP-CoNLL*, pages 1044–1050.

Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. 2012. Bayesian symbol-refined tree substitution grammars for syntactic parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 440–448.

Noah A. Smith. 2011. *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, May.

Mark Steedman, Miles Osbourne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. 2003. Bootstrapping statistical parsers from small datasets. In *Proceedings of EACL*, pages 759–763.

Mihai Surdeanu and Christopher D. Manning. 2010. Ensemble models for dependency parsing: Cheap and good? In *Proceedings of the conference on Human Language Technologies and the conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL'10)*, pages 649–652.

David Vadas and James R. Curran. 2007. Adding noun phrase structure to the penn treebank. In *Proceedings of ACL*, pages 240–247.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, pages 195–206.

Hui Zhang, Min Zhang, Chew Lim Tan, and Haizhou Li. 2009. K-best combination of syntactic parsers. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1552–1560.