Greed Is Good If Randomized: New Inference for Dependency Parsing

Greed is Good if Randomized: New Inference for Dependency Parsing

Yuan Zhang[*], **Tao Lei**[*], **Regina Barzilay, and Tommi Jaakkola**
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{yuanzh, taolei, regina, tommi}@csail.mit.edu

Abstract

Dependency parsing with high-order features results in a provably hard decoding problem. A lot of work has gone into developing powerful optimization methods for solving these combinatorial problems. In contrast, we explore, analyze, and demonstrate that a substantially simpler randomized greedy inference algorithm already suffices for near optimal parsing: a) we analytically quantify the number of local optima that the greedy method has to overcome in the context of first-order parsing; b) we show that, as a decoding algorithm, the greedy method surpasses dual decomposition in second-order parsing; c) we empirically demonstrate that our approach with up to third-order and global features outperforms the state-of-the-art dual decomposition and MCMC sampling methods when evaluated on 14 languages of non-projective CoNLL datasets.[1]

1 Introduction

Dependency parsing is typically guided by parameterized scoring functions that involve rich features exerting refined control over the choice of parse trees. As a consequence, finding the highest scoring parse tree is a provably hard combinatorial inference problem (McDonald and Pereira, 2006). Much of the recent work on parsing has focused on solving these problems using powerful optimization techniques. In this paper, we follow a different strategy, arguing that a much simpler inference strategy suffices. In fact, we demonstrate that a randomized greedy method of inference surpasses the state-of-the-art performance in dependency parsing.

Our choice of a randomized greedy algorithm for parsing follows from a successful track record of such methods in other hard combinatorial problems. These conceptually simple and intuitive algorithms have delivered competitive approximations across a broad class of NP-hard problems ranging from set cover (Hochbaum, 1982) to MAX-SAT (Resende et al., 1997). Their success is predicated on the observation that most realizations of problems are much easier to solve than the worst-cases. A simpler algorithm will therefore suffice in typical cases. Evidence is accumulating that parsing problems may exhibit similar properties. For instance, methods such as dual decomposition offer certificates of optimality when the highest scoring tree is found. Across languages, dual decomposition has shown to lead to a certificate of optimality for the vast majority of the sentences (Koo et al., 2010; Martins et al., 2011). These remarkable results suggest that, as a combinatorial problem, parsing appears simpler than its broader complexity class would suggest. Indeed, we show that a simpler inference algorithm already suffices for superior results.

In this paper, we introduce a randomized greedy algorithm that can be easily used with any rich scoring function. Starting with an initial tree drawn uniformly at random, the algorithm makes only local myopic changes to the parse tree in an attempt to climb the objective function. While a single run of the hill-climbing algorithm may indeed get stuck in a locally optimal solution, multiple random restarts can help to overcome this problem. The same algorithm is used both for learning the parameters of the scoring function as well as for parsing test sentences.

The success of a randomized greedy algorithm is tied to the number of local maxima in the search space. When the number is small, only a few restarts will suffice for the greedy algorithm to find the highest scoring parse. We provide an al-

[*] Both authors contributed equally.

[1] Our code is available at https://github.com/taolei87/RBGParser.

gorithm for explicitly counting the number of local optima in the context of first-order parsing, and demonstrate that the number is typically quite small. Indeed, we find that a first-order parser trained with exact inference or using our randomized greedy algorithm delivers basically the same performance.

We hypothesize that parsing with high-order scoring functions exhibits similar properties. The main rationale is that, even in the presence of high-order features, the resulting scoring function remains first-order dominant. The performance of a simple arc-factored first-order parser is only a few percentage points behind higher-order parsers. The higher-order features in the scoring function offer additional refinement but only a few changes above and beyond the first-order result. As a consequence, most of the arc choices are already determined by a much simpler, polynomial time parser.

We use dual decomposition to show that the greedy method indeed succeeds as an inference algorithm even with higher-order scoring functions. In fact, with second-order features, regardless of which method was used for training, the randomized greedy method outperforms dual decomposition by finding higher scoring trees. For the sentences that dual decomposition is optimal (obtains a certificate), the greedy method finds the same solution in over 99% of the cases. Our simple inference algorithm is therefore likely to scale to higher-order parsing and we demonstrate empirically that this is indeed so.

We validate our claim by evaluating the method on the CoNLL dependency benchmark that comprises treebanks from 14 languages. Averaged across all languages, our method outperforms state-of-the-art parsers, including TurboParser (Martins et al., 2013) and our earlier sampling-based parser (Zhang et al., 2014). On seven languages, we report the best published results. The method is not sensitive to initialization. In fact, drawing the initial tree uniformly at random results in the same performance as when initialized from a trained first-order distribution. In contrast, sufficient randomization of the starting point is critical. Only a small number of restarts suffices for finding (near) optimal parse trees.

2 Related Work

Finding Optimal Structure in Parsing The use of rich-scoring functions in dependency parsing inevitably leads to the challenging combinatorial problem of finding the maximizing parse. In fact, McDonald and Pereira (2006) demonstrated that the task is provably NP-hard for non-projective second-order parsing. Not surprisingly, approximate inference has been at the center of parsing research. Examples of these approaches include easy-first parsing (Goldberg and Elhadad, 2010), inexact search (Johansson and Nugues, 2007; Zhang and Clark, 2008; Huang et al., 2012; Zhang et al., 2013), partial dynamic programming (Huang and Sagae, 2010) and dual decomposition (Koo et al., 2010; Martins et al., 2011).

Our work is most closely related to the MCMC sampling-based approaches (Nakagawa, 2007; Zhang et al., 2014). In our earlier work, we developed a method that learns to take guided stochastic steps towards a high-scoring parse (Zhang et al., 2014). In the heart of that technique are sophisticated samplers for traversing the space of trees. In this paper, we demonstrate that a substantially simpler approach that starts from a tree drawn from the uniform distribution and uses hill-climbing for parameter updates achieves similar or higher performance.

Another related greedy inference method has been used for non-projective dependency parsing (McDonald and Pereira, 2006). This method relies on hill-climbing to convert the highest scoring projective tree into its non-projective approximation. Our experiments demonstrate that when hill-climbing is employed as a primary learning mechanism for high-order parsing, it exhibits different properties: the distribution for initialization does not play a major role in the final outcome, while the use of restarts contributes significantly to the quality of the resulting tree.

Greedy Approximations for NP-hard Problems There is an expansive body of research on greedy approximations for NP-hard problems. Examples of NP-hard problems with successful greedy approximations include the traveling saleman problem problem (Held and Karp, 1970; Rego et al., 2011), the MAX-SAT problem (Mitchell et al., 1992; Resende et al., 1997) and vertex cover (Hochbaum, 1982). While some greedy methods have poor worst-case complexity, many

of them work remarkably well in practice. Despite the apparent simplicity of these algorithms, understanding their properties is challenging: often their "theoretical analyses are negative and inconclusive" (Amenta and Ziegler, 1999; Spielman and Teng, 2001). Identifying conditions under which approximations are provably optimal is an active area of research in computer science theory (Dumitrescu and Tóth, 2013; Jonsson et al., 2013).

In NLP, randomized and greedy approximations have been successfully used across multiple applications, including machine translation and language modeling (Brown et al., 1993; Ravi and Knight, 2010; Daumé III et al., 2009; Moore and Quirk, 2008; Deoras et al., 2011). In this paper, we study the properties of these approximations in the context of dependency parsing.

3 Method

3.1 Preliminaries

Let x be a sentence and $\mathcal{T}(x)$ be the set of possible dependency trees over the words in x. We use $y \in \mathcal{T}(x)$ to denote a dependency tree for x, and $y(m)$ to specify the head (parent) of the modifier word indexed by m in tree y. We also use m to denote the indexed word when there is no ambiguity. In addition, we define $\mathcal{T}(y, m)$ as the set of "neighboring trees" of y obtained by changing only the head of the modifier, i.e. $y(m)$.

The dependency trees are scored according to $S(x, y) = \theta \cdot \phi(x, y)$, where θ is a vector of parameters and $\phi(x, y)$ is a sparse feature vector representation of tree y for sentence x. In this work, $\phi(x, y)$ will include up to third-order features as well as a range of global features commonly used in re-ranking methods (Collins, 2000; Charniak and Johnson, 2005; Huang, 2008).

The parameters θ in the scoring function are estimated on the basis of a training set $D = \{(\hat{x}_i, \hat{y}_i)\}_{i=1}^{N}$ of sentences $\hat{x}_i$ and the corresponding gold (target) trees $\hat{y}_i$. We adopt a max-margin framework for this learning problem. Specifically, we aim to find parameter values that score the gold target trees higher than others:

$$\forall i \in \{1, \cdots, N\}, y \in \mathcal{T}(\hat{x}_i),$$
$$S(\hat{x}_i, \hat{y}_i) \geq S(\hat{x}_i, y) + \|\hat{y}_i - y\|_1 - \xi_i$$

where $\xi_i \geq 0$ is the slack variable (non-zero values are penalized against) and $\|\hat{y}_i - y\|_1$ is the hamming distance between the gold tree $\hat{y}_i$ and a candidate parse y.

In an online learning setup, parameters are updated successively after each sentence. Each update still requires us to find the "strongest violation", i.e., a candidate tree $\tilde{y}$ that scores higher than the gold tree $\hat{y}_i$:

$$\tilde{y} = \arg\max_{y \in \mathcal{T}(\hat{x}_i)} \{S(\hat{x}_i, y) + \|y - \hat{y}_i\|_1\}$$

The parameters are then revised so as to select against the offending $\tilde{y}$. Instead of a standard parameter update based on $\tilde{y}$ as in perceptron, stochastic gradient descent, or passive-aggressive updates, our implementation follows Lei et al. (2014) where the first-order parameters are broken up into a tensor. Each tensor component is updated successively in combination with the parameters corresponding to MST features (McDonald et al., 2005) and higher-order features (when included).[2]

3.2 Algorithm

During training and testing, the key combinatorial problem we must solve is that of decoding, i.e., finding the highest scoring tree $\tilde{y} \in \mathcal{T}(x)$ for each sentence x (or $\hat{x}_i$). In our notation,

$$\tilde{y} = \arg\max_{y \in \mathcal{T}(\hat{x}_i)} \{\theta \cdot \phi(\hat{x}_i, y) + \|y - \hat{y}_i\|_1\} \quad \text{(train)}$$

$$\tilde{y} = \arg\max_{y \in \mathcal{T}(x)} \{\theta \cdot \phi(x, y)\} \quad \text{(test)}$$

While the decoding problem with feature sets similar to ours has been shown to be NP-hard, many approximation algorithms work remarkably well. We commence with a motivating example.

Locality and Parsing One possible reason for why greedy or other approximation algorithms work well for dependency parsing is that typical sentences and therefore the learned scoring functions $S(x, y) = \theta \cdot \phi(x, y)$ are primarily "local". By this we mean that head-modifier decisions could be made largely without considering the surrounding structure (the context). For example, in English an adjective and a determiner are typically attached to the following noun.

We demonstrate the degree of locality in dependency parsing by comparing a first-order tree-based parser to the parser that predicts each head word independently of others. Note that the independent prediction of dependency arcs does not necessarily give rise to a tree. The parameters of

[2] We refer the readers to Lei et al. (2014) for more details about the tensor scoring function and the online update.

Dataset	Indp. Pred	Tree Pred
Slovene	83.7	84.2
Arabic	79.0	79.2
Japanese	93.4	93.7
English	91.6	91.9
Average	86.9	87.3

Table 1: Head attachment accuracy of a first-order local classifier (left) and a first-order structural prediction model (right). The two types of models are trained using the same set of features.

Input: parameter θ, sentence x
Output: dependency tree $\tilde{y}$

Randomly initialize tree $y^{(0)}$;
$t = 0$;
repeat
  list = bottom-up node list of $y^{(t)}$;
  for each word m in list **do**
    $y^{(t+1)} = \arg\max_{y \in \mathcal{T}(y^{(t)}, m)} S(x, y)$;
    $t = t + 1$;
  end for
until no change in this iteration
return $\tilde{y} = y^{(t)}$;

Figure 1: A randomized hill-climbing algorithm for dependency parsing.

the two parsers, the independent prediction and a tree-based parser, are trained separately with the corresponding decoding algorithm but with the same feature set.

Table 1 shows that the accuracy of the independent prediction ranges from 79% to 93% on four CoNLL datasets. The results are on par with the first-order structured prediction model. This experiment reinforces the conclusion in Liang et al. (2008), where a local classifier was shown to achieve comparable accuracy to a sequential model (e.g. CRF) in POS tagging and named-entity recognition.

Hill-Climbing with Random Restarts We build here on the motivating example and explore greedy algorithms as generalizations of purely local decoding. Greedy algorithms break the decoding problem into a sequence of simple local steps, each required to improve the solution. In our case, simple local steps correspond to choosing the head for each modifier word.

We begin with a tree $y^{(0)}$, which can be a sample drawn uniformly from $\mathcal{T}(x)$ (Wilson, 1996). Our greedy algorithm then updates $y^{(t)}$ to a better tree $y^{(t+1)}$ by revising the head of one modifier word while maintaining the constraint that the resulting structure is a tree. The modifiers are considered in the bottom-up order relative to the current tree (the word furthest from the root is considered first). We provide an analysis to motivate this bottom-up update strategy in Section 4.1. The algorithm continues until the score can no longer be improved by changing the head of a single word. The resulting tree represents a locally optimal prediction relative to a single-arc greedy algorithm. Figure 1 gives the algorithm in pseudo-code.

There are many possible variations of the simple randomized greedy hill-climbing algorithm. First, the Wilson sampling algorithm (Wilson, 1996) can be naturally extended to obtain i.i.d. samples from any first-order distributions. Therefore, we could initialize the tree $y^{(0)}$ with a tree from a first-order parser, or draw the initial tree from a first-order distribution other than uniform. However, perhaps surprisingly, as we demonstrate later, little is lost with uniform initialization. Second, since a single run of randomized hill-climbing is relatively cheap and runs are independent to each other, it is easy to execute multiple runs independently in parallel. The final predicted tree is then simply the highest scoring tree across the multiple runs. We demonstrate that only a small number of parallel runs are necessary for near optimal prediction.

4 Analysis

4.1 First-Order Parsing

We provide here a firmer basis for why the randomized greedy algorithm can be expected to work. While the focus of the rest of the paper is on higher-order parsing, we limit ourselves in this subsection to first-order parsing. The reasons for this are threefold. First, a simple greedy algorithm is already not guaranteed a priori to work in the context of a first-order scoring function. The conclusions from this analysis are therefore likely to carry over to higher-order parsing scenarios as well. Second, a first-order arc-factored scoring provides us an easy way to ascertain when the randomized greedy algorithm indeed found the highest scoring tree. Finally, we are able to count the

Dataset	Average Len.	# of local optima at percentile			fraction of finding global optima (%)	
		50%	70%	90%	0 <Len.≤ 15	Len.> 15
Turkish	12.1	1	1	2	100	100
Slovene	15.9	2	20	3647	100	98.1
English	24.0	21	121	2443	100	99.3
Arabic	36.8	2	35	>10000	100	99.1

Table 2: The left part of the table shows the local optimum statistics of the first-order model. The sentences are sorted by the number of local optima. Columns 3 to 5 show the number of local optima of a sentence at different percentile of the sorted list. For example, on English 50% of the sentences have no more than 21 local optimum trees. The right part shows the fraction of finding global optima using 300 uniform restarts for each sentence.

number of locally optimal solutions for a greedy algorithm in the context of first-order parsing and can therefore relate this property to the success rates of the algorithm.

Reachability We begin by highlighting a basic property of trees, namely that single arc changes suffice for transforming any tree to any other tree in a small number of steps while maintaining that each intermediate structure is also a tree. In this sense, a target tree is reachable from any starting point using only single arc changes. More formally, let y be any starting tree and y' the desired target. Let $m_1, m_2, \cdots, m_n$ be the bottom-up list of words (modifiers) corresponding to tree y, where m_1 is the word furthest from the root. We can simply change each head $y(m_i)$ to that of $y'(m_i)$ in this order $i = 1, \ldots, n$. The bottom-up order guarantees that no cycle is introduced with respect to the remaining (yet unmodified) nodes of y. The fact that y' is a valid tree implies no cycle will appear with respect to the already modified nodes.

Note that, according to this property, any tree is reachable from any starting point using only k modifications, where k is the number of head differences, i.e. $k = |\{m : y(m) \neq y'(m)\}|$. The result also suggests that it may be helpful to perform the greedy steps in the bottom-up order, a suggestion that we follow in our implementation.

Broadly speaking, we have established that the greedy algorithm is not inherently limited by virtue of its basic steps. Of course, it is a different question whether the scoring function supports such local changes towards the correct target tree.

Locally Optimal Trees While greedy algorithms are notoriously prone to getting stuck in locally optimal solutions, we establish here that decoding with learned scoring functions involves only a small number of local optima. In our case, a local optimum corresponds to a tree y where no single change of head $y(m)$ results in a higher scoring tree. Clearly, the highest scoring tree is also a local optimum in this sense. If there were many such local optima, finding the one with the highest score would be challenging for a greedy algorithm, even with randomization.

We begin with a worst case analysis and estab-

Function **CountOptima**($G = \langle V, E \rangle$)
$V = \{w_0, w_1, \cdots, w_n\}$ where w_0 is the root
$E = \{e_{ij} \in \mathbb{R}\}$ are the arc scores
Return: the number of local optima

1: Let $y(0) = \emptyset$ and $y(i) = \arg\max_j e_{ji}$;
2: **if** y is a tree (no cycle) **then return** 1;
3: Find a cycle $C \subset V$ in y;
4: count $= 0$;
// contract the cycle
5: create a vertex w_*;
6: $\forall j \notin C : e_{*j} = \max_{k \in C} e_{kj}$;
7: **for** each vertex $w_i \in C$ **do**
8: $\quad \forall j \notin C : e_{j*} = e_{ji}$;
9: $\quad V' = V \cup \{w_*\} \setminus C$;
10: $\quad E' = E \cup \{e_{*j}, e_{j*} \mid \forall j \notin C\}$
11: $\quad$ count += CountOptima($G' = \langle V', E' \rangle$);
12: **end for**
13: **return** count;

Figure 2: A recursive algorithm for counting local optima for a sentence with words $w_1, \cdots, w_n$ (first-order parsing). The algorithm resembles the Chu-Liu-Edmonds algorithm for finding the maximum directed spanning tree (Chu and Liu, 1965).

Dataset	Trained with Hill-Climbing (HC)				Trained with Dual Decomposition (DD)			
	%Cert (DD)	$s_{DD} > s_{HC}$	$s_{DD} = s_{HC}$	$s_{DD} < s_{HC}$	%Cert (DD)	$s_{DD} > s_{HC}$	$s_{DD} = s_{HC}$	$s_{DD} < s_{HC}$
Turkish	98.7	0.0	99.8	0.2	98.7	0.0	100.0	0.0
Slovene	94.5	0.0	98.7	1.3	92.3	0.2	99.0	0.8
English	94.5	0.3	98.7	1.0	94.6	0.5	98.7	0.8
Arabic	78.8	3.4	93.9	2.7	75.3	4.7	88.4	6.9

Table 3: Decoding quality comparison between hill-climbing (HC) and dual decomposition (DD). Models are trained either with HC (left) or DD (right). s_{HC} denotes the score of the tree retrieved by HC and s_{DD} gives the analogous score for DD. The columns show the percentage of all test sentences for which one method succeeds in finding a higher or the same score. "Cert" column gives the percentage of sentences for which DD finds a certificate.

lish a tight upper bound on the number of local optima for a first-order scoring function.

Theorem 1 *For any first-order scoring function that factorizes into the sum of arc scores* $S(x, y) = \sum S_{arc}(y(m), m)$*: (a) the number of locally optimal trees is at most* 2^{n-1} *for* n *words; (b) this upper bound is tight.*[3]

While the number of possible dependency trees is $(n + 1)^{n-1}$ (Cayley's formula), the number of local optima is at most 2^{n-1}. This is still too many for longer sentences, suggesting that, in the worst case, a randomized greedy algorithm is unlikely to find the highest scoring tree. However, the scoring functions we learn for dependency parsing are considerably easier.

Average Case Analysis In contrast to the worst-case analysis above, we will count here the actual number of local optima *per sentence* for a first-order scoring function learned from data with the randomized greedy algorithm. Figure 2 provides pseudo-code for our counting algorithm. The algorithm is derived by tailoring the proof of Theorem 1 to each sentence.

Table 2 shows the empirical number of locally optimal trees estimated by our algorithm across 4 different languages. Decoding with trained scoring functions in the average case is clearly substantially easier than the worst case. For example, on the English test set more than 70% of the sentences have at most 121 locally optimal trees. Since the average sentence length is 24, the discrepancy between the typical number (e.g., 121) and the worst case (2^{24-1}) is substantial. As a result, only a small number of restarts is likely to suffice for finding optimal trees in practice.

Optimal Decoding We can easily verify whether the randomized greedy algorithm indeed succeeds in finding the highest scoring trees with a learned first-order scoring function. We have established above that there are typically only a small number of locally optimal trees. We would therefore expect the algorithm to work. We show the results in the second part of Table 2. For short sentences of length up to 15, our method finds the global optimum for all the test sentences. Success rates remain high even for longer test sentences.

4.2 Higher-Order Parsing

Exact decoding with high-order features is known to be provably hard (McDonald et al., 2005). We begin our analysis here with a second-order (sibling/grandparent) model, and compare our randomized hill-climbing (HC) method to dual decomposition (DD), re-implementing Koo et al. (2010). Table 3 compares decoding quality for the two methods across four languages. Overall, in 97.8% of the sentences, HC obtains the same score as DD, in 1.3% of the cases HC finds a higher scoring tree, and in 0.9% of cases DD results in a better tree. The results follow the same pattern regardless of which method was used to train the scoring function. The average rate of certificates for DD was 92%. In over 99% of these sentences, HC reaches the same optimum.

We expect that these observations about the success of HC carry over to other high-order parsing models for several reasons. First, a large number of arcs are pruned in the initial stage, considerably reducing the search space and minimizing the number of possible locally optimal trees. Second, many dependencies can be determined already with independent arc prediction (see our motivating example above), predictions that are readily achieved with a greedy algorithm. Finally, high-order features represent smaller refinements, i.e., suggest only a few changes above and beyond the dominant first-order scores. Greedy al-

[3] A proof sketch is given in Appendix.

gorithms are therefore likely to be able to leverage at least some of this potential. We demonstrate below that this is indeed so.

Our methods are trained within the max-margin framework. As a result, we are expected to find the highest scoring competing tree for each training sentence (the "strongest violation"). One may question therefore whether possible sub-optimal decoding for some training sentences (finding "a violation" rather than the "strongest violation") impacts the learned parser. To this end, Huang et al. (2012) have established that weaker violations do suffice for separable training sets.

5 Experimental Setup

Dataset and Evaluation Measures We evaluate our model on CoNLL dependency treebanks for 14 different languages (Buchholz and Marsi, 2006; Surdeanu et al., 2008), using standard training and testing splits. We use part-of-speech tags and the morphological information provided in the corpus. Following standard practice, we use Unlabeled Attachment Score (UAS) excluding punctuation (Koo et al., 2010; Martins et al., 2013) as the evaluation metric in all our experiments.

Baselines We compare our model with the TurboParser (Martins et al., 2013) and our earlier sampling-based parser (Zhang et al., 2014). For both parsers, we directly compare with the recent published results on the CoNLL datasets. We also compare our parser against the best published results for the individual languages in our datasets. This comparison set includes four additional parsers: Martins et al. (2011), Koo et al. (2010), Zhang et al. (2013) and our tensor-based parser (Lei et al., 2014).

Features We use the same feature templates as in our prior work (Zhang et al., 2014; Lei et al., 2014)[4]. Figure 3 shows the first- to third-order feature templates that we use in our model. For the global features we use right-branching, coordination, PP attachment, span length, neighbors, valency and non-projective arcs features.

Implementation Details Following standard practices, we train our model using the passive-aggressive online learning algorithm (MIRA) and parameter averaging (Crammer et al., 2006; Collins, 2002). By default we use an adaptive strategy for running the hill-climbing algorithm – for a given sentence we repeatedly run the algorithm in parallel[5] until the best tree does not change for $K = 300$ consecutive restarts. For each restart, by default we initialize the tree $y^{(0)}$ by sampling from the first-order distribution using the current learned parameter values (and first-order scores). We train our first-order and third-order model for 10 epochs and our full model for 20 epochs for all languages, and report the average performance across three independent runs.

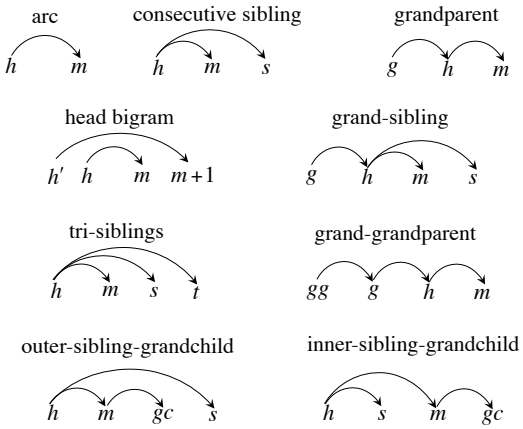


Figure 3: First- to third-order features.

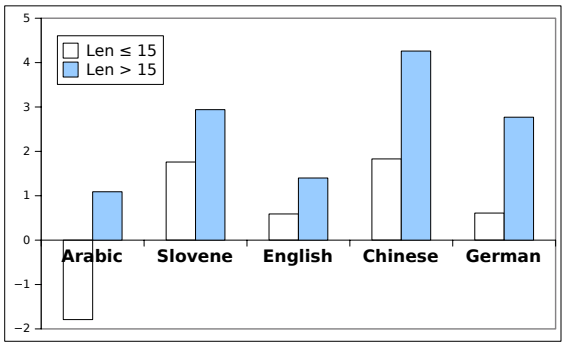


Figure 4: Absolute UAS improvement of our full model over the first-order model. Sentences in the test set are divided into 2 groups based on their lengths.

6 Results

Comparison with the Baselines Table 4 summarizes the results of our model, along with the state-of-the-art baselines. On average across 14 languages, our full model with the tensor component outperforms both TurboParser and the sampling-based parser. The direct comparison

[4]We refer the readers to Zhang et al. (2014) and Lei et al. (2014) for the detailed definition of each feature template.

[5]We use 8 threads in all the experiments.

	Our Model				Exact 1st	Turbo (MA13)	Sampling (ZL14)	Best Published
	1st	3rd	$\text{Full}_{\text{w/o tensor}}$	Full				
Arabic	78.98	79.95	79.38	80.24	79.22	79.64	80.12	81.12 (MS11)
Bulgarian	92.15	93.38	93.69	93.72	92.24	93.10	93.30	94.02 (ZH13)
Chinese	91.20	93.00	92.76	**93.04**	91.17	89.98	92.63	92.68 (LX14)
Czech	87.65	90.11	90.34	90.77	87.82	90.32	91.04	91.04 (ZL14)
Danish	90.50	91.43	91.66	91.86	90.56	91.48	91.80	92.00 (ZH13)
Dutch	84.49	86.43	87.04	**87.39**	84.79	86.19	86.47	86.47 (ZL14)
English	91.85	93.01	93.20	**93.25**	91.94	93.22	92.94	93.22 (MA13)
German	90.52	91.91	92.64	**92.67**	90.54	92.41	92.07	92.41 (MA13)
Japanese	93.78	**93.80**	93.35	93.56	93.74	93.52	93.42	93.74 (LX14)
Portuguese	91.12	92.07	92.60	92.36	91.16	92.69	92.41	93.03 (KR10)
Slovene	84.29	86.48	**87.06**	86.72	84.15	86.01	86.82	86.95 (MS11)
Spanish	85.52	87.87	88.17	**88.75**	85.59	85.59	88.24	88.24 (ZL14)
Swedish	89.89	91.17	91.35	91.08	89.78	91.14	90.71	91.62 (ZH13)
Turkish	76.57	76.80	76.13	76.68	76.40	76.90	77.21	77.55 (KR10)
Average	87.75	89.10	89.24	89.44	87.79	88.72	89.23	89.58

Table 4: Results of our model and several state-of-the-art systems. "Best Published UAS" includes the most accurate parsers among Martins et al. (2011), Martins et al. (2013), Koo et al. (2010), Zhang et al. (2013), Lei et al. (2014) and Zhang et al. (2014). For the third-order model, we use the feature set of TurboParser (Martins et al., 2013). The full model combines features of our sampling-based parser (Zhang et al., 2014) and tensor features (Lei et al., 2014).

Dataset	MAP-1st		Uniform		Rnd-1st	
	UAS	Init.	UAS	Init.	UAS	Init.
Slovene	85.2	80.1	86.7	13.7	86.7	34.2
Arabic	78.8	75.1	79.7	12.4	80.2	32.8
English	91.1	82.0	93.3	39.6	93.3	55.6
Chinese	87.2	75.3	93.2	36.8	93.0	54.5
Dutch	84.8	79.5	87.0	26.9	87.4	45.6
Average	85.4	78.4	88.0	25.9	88.1	44.5

Table 5: Comparison between different initialization strategies: (a) MAP-1st: only the MAP tree of the first-order score; (b) Uniform: random trees are sampled from the uniform distribution; and (c) Rnd-1st: random trees are sampled from the first-order distribution. For each method, the table shows the average accuracy of the initial tree and the final parsing accuracy.

with TurboParser is achieved by restricting our model to third order features which still outperforms TurboParser (89.10% vs 88.72%). To compare against the sampling-based parser, we employ our model without the tensor component. The two models achieve a similar average performance (89.24% and 89.23% respectively). Since relative parsing performance depends on a target language, we also include comparison with the best published results. The model achieves the best published results for seven languages.

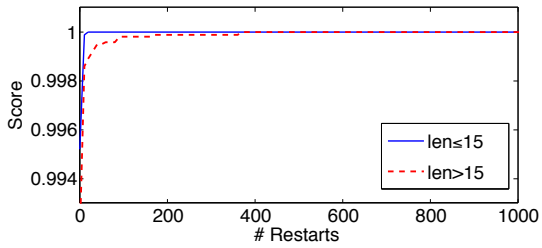
Another noteworthy comparison concerns first-order parsers. As Table 4 shows, the exact and approximate versions of the first-order parser deliver almost identical performance.

Impact of High-Order Features Table 4 shows that the model can effectively utilize high-order features. Comparing the average performance of the model variants, we see that the accuracy on the benchmark languages consistently improves when higher-order features are added. This characteristic of the randomized greedy parser is in line with findings about other state-of-the-art high-order parsers (Martins et al., 2013; Zhang et al., 2014). Figure 4 breaks down these gains based on the sentence length. As expected, on most languages high-order features are particularly helpful when parsing longer sentences.

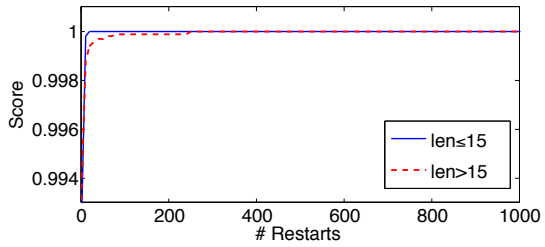
Impact of Initialization and Restarts Table 5 shows the impact of initialization on the model performance for several languages. We consider three strategies: the MAP estimate of the first-order score from the model, uniform sampling and sampling from the first-order distribution. The accuracy of initial trees varies greatly, ranging from 78.4% for the MAP estimate to 25.9% and 44.5% for the latter randomized strategies. However, the resulting parsing accuracy is not determined by the initial accuracy. In fact, the two sampling strategies result in almost identical parsing performance. While the first-order MAP estimate gives the best initial guess, the overall parsing accuracy of this method lags behind. This result demonstrates the importance of restarts – in contrast to the randomized strategies, the MAP initialization performs only a single run of hill-climbing.

	Length ≤ 15	Length > 15
Slovene	100	98.11
English	100	99.12

Table 6: Fractions (%) of the sentences that find the best solution among 3,000 restarts within the first 300 restarts.



(a) Slovene

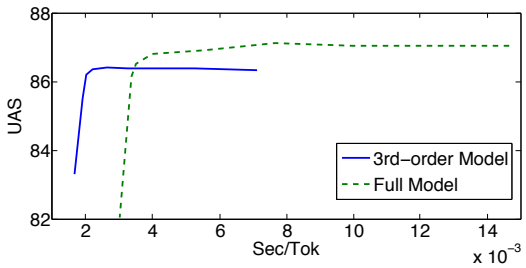


(b) English

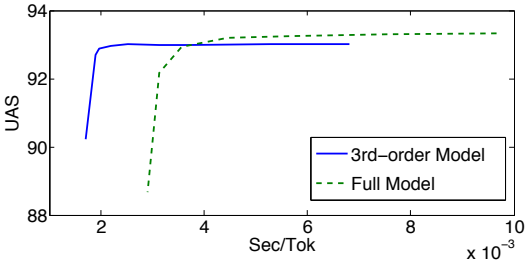
Figure 5: Convergence analysis on Slovene and English datasets. The graph shows the normalized score of the output tree as a function of the number of restarts. The score of each sentence is normalized by the highest score obtained for this sentence after 3,000 restarts. We only show the curves up to 1,000 restarts because they all reach convergence after around 500 restarts.

Convergence Properties Figure 5 shows the score of the trees retrieved by our full model with respect to the number of restarts, for short and long sentences in English and Slovene. To facilitate the comparison, we normalize the score of each sentence by the maximal score obtained for this sentence after 3,000 restarts. Overall, most sentences converge quickly. This view is also supported by Table 6 which shows the fraction of the sentences that converge within the first 300 restarts. We can see that all the short sentences (length up to 15) reach convergence within the allocated restarts. Perhaps surprisingly, more than 98% of the long sentences also converge within 300 restarts.

Decoding Speed As the number of restarts impacts the parsing accuracy, we can trade performance for speed. Figure 6 shows that the model



(a) Slovene



(b) English

Figure 6: Trade-off between performance and speed on Slovene and English datasets. The graph shows the accuracy as a function of decoding speed measured in second per token. Variations in decoding speed is achieved by changing the number of restarts.

achieves high performance with acceptable parsing speed. While various system implementation issues such as programming language and computational platform complicate a direct comparison with other parsing systems, our model delivers parsing time roughly comparable to other state-of-the-art graph-based systems (for example, TurboParser and MST parser) and the sampling-based parser.

7 Conclusions

We have shown that a simple, generally applicable randomized greedy algorithm for inference suffices to deliver state-of-the-art parsing performance. We argued that the effectiveness of such greedy algorithms is contingent on having a small number of local optima in the scoring function. By algorithmically counting the number of locally optimal solutions in the context of first-order parsing, we show that this number is indeed quite small. Moreover, we show that, as a decoding algorithm, the greedy method surpasses dual decomposition in second-order parsing. Finally, we empirically demonstrate that our approach with up to third-order and global features outperforms the state-of-the-art parsers when evaluated on 14 languages of

non-projective CoNLL datasets.

Appendix

We provide here a more detailed justification for the counting algorithm in Figure 2 and, by extension, a proof sketch of Theorem 1. The bullets below follow the operation of the algorithm.

- Whenever independent selection of the heads results in a valid tree, there is only 1 optimum (Lines 1&2 of the algorithm). Otherwise there must be a cycle C in y (Line 3 of the algorithm)
- We claim that any locally optimal tree y' of the graph $G = (V, E)$ must contain $|C| - 1$ arcs of the cycle $C \subseteq V$. This can be shown by contradiction. If y' contains less than $|C| - 1$ arcs of C, then (a) we can construct a tree y'' that contains $|C| - 1$ arcs; (b) the heads in y'' are strictly better than those in y' over the unused part of the cycle; (c) by reachability, there is a path $y' \rightarrow y''$ so y' cannot be a local optimum.
- Any locally optimal tree in G must select an arc in C and reassign it. The rest of the $|C|-1$ arcs will then result in a chain.
- By contracting cycle C we obtain a new graph G' of size $|G| - |C| + 1$ (Lines 5-11 of the algorithm). Easy to verify that (not shown): any local optimum in G' is a local optimum in G and vice versa.

The theorem follows as a corollary of these steps. To see this, let $F(G_m)$ be the number of local optima in the graph of size m:

$$F(G_m) \leq \max_{C \subseteq V(G)} \sum_i F(G^{(i)}_{m-c+1})$$

where $G^{(i)}_{m-c+1}$ is the graph (of size $m - c + 1$) created by selecting the i^{th} arc in cycle C and contracting G_m accordingly, and $c = |C|$ is the size of the cycle. Define $\hat{F}(m)$ as the upper bound of $F(G_m)$ for any graph of size m. By the above formula, we know that

$$\hat{F}(m) \leq \max_{2 \leq c < m} \hat{F}(m - c + 1) \times c$$

By solving for $\hat{F}(m)$ we get $\hat{F}(m) \leq 2^{m-2}$. Since $m = n+1$ for a sentence with n words, the upper-bound of local optima is 2^{n-1}.

To show the tightness, for any $n > 0$, create the graph G_{n+1} with arc scores $e_{ij} = e_{ji} = i$ for any $0 \leq i < j \leq n$. Note that $w_n \rightarrow w_{n-1} \rightarrow w_n$ forms the circle C of size 2, it can be shown by induction on n and $F(G_{n+1})$ that $F(G_{n+1}) = F(G_n) \times 2 = 2^{n-1}$.

Acknowledgments

This research is developed in collaboration with the Arabic Language Technologies (ALT) group at Qatar Computing Research Institute (QCRI) within the IYAS project. The authors acknowledge the support of the U.S. Army Research Office under grant number W911NF-10-1-0533, and of the DARPA BOLT program. We thank the MIT NLP group and the ACL reviewers for their comments. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors, and do not necessarily reflect the views of the funding organizations.

References

Nina Amenta and Günter Ziegler, 1999. *Deformed Products and Maximal Shadows of Polytopes*. Contemporary Mathematics. American Mathematics Society.

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06. Association for Computational Linguistics.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180. Association for Computational Linguistics.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 175–182.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural*

Language Processing - Volume 10, EMNLP '02. Association for Computational Linguistics.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine learning*, 75(3):297–325.

Anoop Deoras, Tomáš Mikolov, and Kenneth Church. 2011. A fast re-scoring strategy to capture long distance dependencies. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1116–1127. Association for Computational Linguistics.

Adrian Dumitrescu and Csaba D Tóth. 2013. The traveling salesman problem for lines, balls and planes. In *SODA*, pages 828–843. SIAM.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750. Association for Computational Linguistics.

Michael Held and Richard M Karp. 1970. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162.

Dorit S Hochbaum. 1982. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086. Association for Computational Linguistics.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151. Association for Computational Linguistics.

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

Richard Johansson and Pierre Nugues. 2007. Incremental dependency parsing using online learning. In *EMNLP-CoNLL*, pages 1134–1138.

Peter Jonsson, Victor Lagerkvist, Gustav Nordh, and Bruno Zanuttini. 2013. Complexity of sat problems, clone theory and the exponential time hypothesis. In *SODA*, pages 1264–1277. SIAM.

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*. Association for Computational Linguistics.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

Percy Liang, Hal Daumé III, and Dan Klein. 2008. Structure compilation: trading structure for features. In *Proceedings of the 25th international conference on Machine learning*, pages 592–599. ACM.

André F. T. Martins, Noah A. Smith, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. 2011. Dual decomposition with many overlapping components. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11. Association for Computational Linguistics.

André FT Martins, Miguel B Almeida, and Noah A Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

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

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*.

David Mitchell, Bart Selman, and Hector Levesque. 1992. Hard and easy distributions of sat problems. In *AAAI*, volume 92, pages 459–465. Citeseer.

Robert C Moore and Chris Quirk. 2008. Random restarts in minimum error rate training for statistical machine translation. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 585–592. Association for Computational Linguistics.

Tetsuji Nakagawa. 2007. Multilingual dependency parsing using global features. In *EMNLP-CoNLL*, pages 952–956.

Sujith Ravi and Kevin Knight. 2010. Does giza++ make search errors? *Computational Linguistics*, 36(3):295–302.

César Rego, Dorabela Gamboa, Fred Glover, and Colin Osterman. 2011. Traveling salesman problem heuristics: leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3):427–441.

Mauricio GC Resende, LS Pitsoulis, and PM Pardalos. 1997. Approximate solution of weighted max-sat problems using grasp. *Satisfiability problems*, 35:393–405.

Daniel Spielman and Shang-Hua Teng. 2001. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 296–305. ACM.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, CoNLL '08. Association for Computational Linguistics.

David Bruce Wilson. 1996. Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 296–303. ACM.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics.

Hao Zhang, Liang Zhao, Kai Huang, and Ryan McDonald. 2013. Online learning for inexact hypergraph search. In *Proceedings of EMNLP*.

Yuan Zhang, Tao Lei, Regina Barzilay, Tommi Jaakkola, and Amir Globerson. 2014. Steps to excellence: Simple inference with refined scoring of dependency trees. In *Proceedings of the 52th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.