Synthetic Data Made to Order: The Case of Parsing

Dingquan Wang and **Jason Eisner**
Department of Computer Science, Johns Hopkins University
{wdd,jason}@cs.jhu.edu

Abstract

To approximately parse an unfamiliar language, it helps to have a treebank of a similar language. But what if the closest available treebank still has the wrong word order? We show how to (stochastically) permute the constituents of an existing dependency treebank so that its surface part-of-speech statistics approximately match those of the target language. The parameters of the permutation model can be evaluated for quality by dynamic programming and tuned by gradient descent (up to a local optimum). This optimization procedure yields trees for a new artificial language that resembles the target language. We show that delexicalized parsers for the target language can be successfully trained using such "made to order" artificial languages.

1 Introduction

Dependency parsing is a core task in natural language processing (NLP). Given a sentence, a dependency parser produces a dependency tree, which specifies the typed head-modifier relations between pairs of words. While *supervised* dependency parsing has been successful (McDonald and Pereira, 2006; Nivre, 2008; Kiperwasser and Goldberg, 2016), unsupervised parsing can hardly produce useful parses (Mareček, 2016). So it is extremely helpful to have some treebank of supervised parses for training purposes.

1.1 Past work: Cross-lingual transfer

Unfortunately, manually constructing a treebank for a new target language is expensive (Böhmová et al., 2003). As an alternative, *cross-lingual transfer* parsing (McDonald et al., 2011) is sometimes possible, thanks to the recent development of multi-lingual treebanks (McDonald et al., 2013; Nivre et al., 2015; Nivre et al., 2017). The idea is to parse the sentences of the target language with a supervised parser trained on the treebanks of one or more source languages. Although the parser cannot be expected to know the words of the target language, it can make do with parts of speech (POS) (McDonald et al., 2011; Täckström et al., 2013; Zhang and Barzilay, 2015) or cross-lingual word embeddings (Duong et al., 2015; Guo et al., 2016; Ammar et al., 2016). A more serious challenge is that the parser may not know how to handle the word *order* of the target language, unless the source treebank comes from a closely related language (e.g., using German to parse Luxembourgish). Training the parser on trees from *multiple* source languages may mitigate this issue (McDonald et al., 2011) because the parser is more likely to have seen target part-of-speech sequences somewhere in the training data. Some authors (Rosa and Žabokrtský, 2015a,b; Wang and Eisner, 2016) have shown additional improvements by preferring source languages that are "close" to the target language, where the closeness is measured by distance between POS language models trained on the source and target corpora.

1.2 This paper: Tailored synthetic data

We will focus on delexicalized dependency parsing, which maps an input POS tag sequence to a dependency tree. We evaluate single-source transfer—train a parser on a single source language, and evaluate it on the target language. This is the setup of Zeman and Resnik (2008) and Søgaard (2011a).

Our novel ingredient is that *rather than seek a close source language that already exists, we create one*. How? Given a dependency treebank of a possibly distant source language, we stochastically permute the children of each node, according to some distribution that *makes the permuted language close to the target language*.

And how do we find this distribution? We adopt the tree-permutation model of Wang and Eisner (2016). We design a dynamic programming algorithm which, for any given distribution p in Wang and Eisner's family, can compute the *expected counts* of all POS bigrams in the permuted source treebank. This allows us to evaluate p by computing the divergence between the bigram POS language model formed by these expected counts, and the one formed by the *observed counts* of POS bigrams in the unparsed target language. In order to find a p that locally minimizes this divergence, we adjust the model parameters by stochastic gradient descent (SGD).

1.3 Key limitations in this paper

Better measures of surface closeness between two languages might be devised. However, even counting the expected POS N-grams is moderately expensive, taking time exponential in N if done exactly. So we compute only these local statistics, and only for $N = 2$. We certainly need $N > 1$ because the 1-gram distribution is not affected by permutation at all. $N = 2$ captures useful bigram statistics: for example, to mimic a verb-final language with prenominal modifiers, we would seek constituent permutations that result in matching its relatively high rate of VERB–PUNCT and ADJ–NOUN bigrams. While $N > 2$ might have improved the results, it was too slow for our large-scale experimental design. §7 discusses how richer measures could be used in the future.

We caution that throughout this paper, we assume that our corpora are annotated with gold POS tags, even in the target language (which lacks any gold training trees). This is an idealized setting that has often been adopted in work on unsupervised and cross-lingual transfer.§7 discusses a possible avenue for doing without gold tags.

2 Modeling Surface Realization

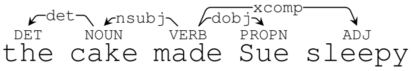
We begin by motivating the idea of tree permutation. Let us suppose that the dependency tree for a sentence starts as a labeled graph—a tree in which siblings are not yet ordered with respect to their parent or one another. Each language has some systematic way to *realize* its unordered trees as surface strings:[1] it imposes a particular order on the tree's word tokens. More precisely, a language specifies a *distribution* $p(\text{string} \mid \text{unordered tree})$ over a tree's possible realizations.

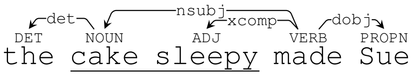
As an engineering matter, we now make the strong assumption that the unordered dependency trees are similar across languages. That is, we suppose that different languages use similar underlying syntactic/semantic graphs, but differ in how they realize this graph structure on the surface.

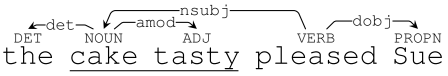
Thus, given a gold POS corpus $\mathbf{u}$ of the unknown target language, we may hope to explain its distribution of surface POS bigrams as the result of applying some target-language surface realization model to the distribution of cross-linguistically "typical" unordered trees. To obtain samples of the latter distribution, we use the treebanks of one or more *other* languages. The present paper evaluates our method when only a single source treebank is used. In the future, we could try tuning a mixture of all available source treebanks.

2.1 Realization is systematic

We presume that the target language applies the same stochastic realization model to all trees. All that we can optimize is the parameter vector of this model. Thus, we deny ourselves the freedom to realize each individual tree in an *ad hoc* way. To see why this is important, suppose the target language is French, whose corpus $\mathbf{u}$ contains many NOUN–ADJ bigrams. We could achieve such a bigram from the unordered source tree the cake made Sue sleepy by ordering it to yield the cake sleepy made Sue . However, that realization is not in fact appropriate for French, so that ordered tree would not be a useful training tree for French. Our approach should disprefer this tempting but incorrect realization, because any model with a high probability of this realization would, if applied *systematically* over the whole corpus, also yield sentences like He sleepy made Sue, with unwanted PRON–ADJ bigrams that would not match the surface statistics of French. We hope our approach will instead choose the realization model that is correct for French, in which the NOUN–ADJ bigrams arise instead from source trees where the ADJ is a dependent of the NOUN, yielding (e.g.) the cake tasty pleased Sue . This has the same POS sequence as the example above (as it happens), but now assigns the correct tree to it.







2.2 A parametric realization model

As our family of realization distributions, we adopt the log-linear model used for this purpose by Wang and Eisner (2016). The model assumes that the root node a of the unordered dependency tree selects an ordering $\pi(a)$ of the n_a nodes consisting

[1] Modeling this process was the topic of the recent Surface Realization Shared Task (Mille et al., 2018). Most relevant is work on *tree linearization* (Filippova and Strube, 2009; Futrell and Gibson, 2015; Puzikov and Gurevych, 2018).

of a and its $n_a - 1$ dependent children. The procedure is repeated recursively at the child nodes. This method can produce only projective trees.

Each node a draws its ordering $\pi(a)$ independently according to

$$p_{\boldsymbol{\theta}}(\pi \mid a) = \frac{1}{Z(a)} \exp \sum_{1 \leq i < j \leq n_a} \boldsymbol{\theta} \cdot \mathbf{f}(\pi, i, j) \quad (1)$$

which is a distribution over the $n_a!$ possible orderings. $Z(a)$ is a normalizing constant. $\mathbf{f}$ is a feature vector extracted from the ordered pair of nodes π_i, π_j, and $\boldsymbol{\theta}$ is the model's parameter vector of feature weights. See Appendix A for the feature templates, which are a subset of those used by Wang and Eisner (2016). These features are able to examine the tree's node labels (POS tags) and edge labels (dependency relations). Thus, when a is a verb, the model can assign a positive weight to "subject precedes verb" or "subject precedes object," thus preferring orderings with these features.

Following Wang and Eisner (2016, §3.1), we choose new orderings for the noun and verb nodes only,[2] preserving the source treebank's order at all other nodes a.

2.3 Generating training data

Given a source treebank B and some parameters $\boldsymbol{\theta}$, we can use equation (1) to randomly sample realizations of the trees in B. The effect is to reorder dependent phrases within those trees. The resulting permuted treebank B' can be used to train a parser for the target language.

2.4 Choosing parameters $\boldsymbol{\theta}$

So how do we choose $\boldsymbol{\theta}$ that works for the target language? Suppose $\mathbf{u}$ is a corpus of target-language POS sequences, using the same set of POS tags as B. We evaluate parameters $\boldsymbol{\theta}$ according to whether POS tag sequences in B' will be distributed like POS tag sequences in $\mathbf{u}$.

To do this, first we estimate a bigram language model $\hat{q}$ from the actual distribution q of POS sequences observed in $\mathbf{u}$. Second, let $p_{\boldsymbol{\theta}}$ denote the distribution of POS sequences that we expect to see in B', that is, POS sequences obtained by stochastically realizing observed trees in B according to $\boldsymbol{\theta}$. We estimate another bigram model $\hat{p}_{\boldsymbol{\theta}}$ from this distribution $p_{\boldsymbol{\theta}}$.

We then try to set $\boldsymbol{\theta}$, using SGD, to minimize a divergence $D(\hat{p}_{\boldsymbol{\theta}}, \hat{q})$ that we will define below.

2.4.1 Estimation of bigram models

Estimating $\hat{q}$ is straightforward: $\hat{q}(t \mid s) = c_q(st)/c_q(s)$, where $c_q(st)$ is the count of POS bigram st in the average[3] sentence of $\mathbf{u}$ and $c_q(s) = \sum_{t'} c_q(st')$. We estimate $\hat{p}_{\boldsymbol{\theta}}$ in the same way, where $c_p(st)$ denotes the *expected* count of st in a random POS sequence $\mathbf{y} \sim p_{\boldsymbol{\theta}}$. This is equivalent to choosing $\hat{q}, \hat{p}_{\boldsymbol{\theta}}$ to minimize the KL-divergences $\mathrm{KL}(q \mid\mid \hat{q}), \mathrm{KL}(p_{\boldsymbol{\theta}} \mid\mid \hat{p}_{\boldsymbol{\theta}})$. It ensures that each model's expected bigram counts match those in the POS sequences.

However, these maximum-likelihood estimates might overfit on our finite data, $\mathbf{u}$ and B. We therefore smooth both models by first adding $\lambda = 0.1$ to all bigram counts $c_q(st)$ and $c_p(st)$.[4]

2.4.2 Divergence of bigram models

We need a metric to evaluate $\boldsymbol{\theta}$. If p and q are bigram language models over POS sequences $\mathbf{y}$ (sentences), their Kullback-Leibler divergence is

$$\mathrm{KL}(p \mid\mid q) \stackrel{\text{def}}{=} \mathbb{E}_{\mathbf{y} \sim p}[\log p(\mathbf{y}) - \log q(\mathbf{y})] \quad (2)$$

$$= \sum_{s,t} c_p(st) \cdot (\log p(t \mid s) - \log q(t \mid s)) \quad (3)$$

where $\mathbf{y}$ ranges over POS sequences and st ranges over POS bigrams. These include bigrams where $s = \text{BOS}$ ("beginning of sequence") or $t = \text{EOS}$ ("end of sequence"), which are boundary tags that we take to surround $\mathbf{y}$.

All quantities in equation (3) can be determined directly from the (expected) bigram counts given by c_p and c_q. No other model estimation is needed.

A concern about equation (3) is that a single bigram st that is badly underrepresented in q may contribute an arbitrarily large term $\log \frac{p(t|s)}{q(t|s)}$. To limit this contribution to at most $\log \frac{1}{\alpha}$, for some small $\alpha \in (0, 1)$, we define $\mathrm{KL}_\alpha(p \mid\mid q)$ by a variant of equation (3) in which $q(t \mid s)$ has been replaced by $\tilde{q}(t \mid s) \stackrel{\text{def}}{=} \alpha p(t \mid s) + (1 - \alpha) q(t \mid s)$.[5]

[2] Specifically, the 93% of nodes tagged with NOUN, PROPN, PRON or VERB in Universal Dependencies format. In retrospect, this restriction was unnecessary in our setting, but it skipped only 4.4% of nodes on average (from 2% to 11% depending on language). The remaining nodes were nouns, verbs, or childless.

[3] A more familiar definition of c_q would use the *total* count in $\mathbf{u}$. Our definition, which yields the same bigram probabilities, is analogous to our definition of c_p. This c_p is needed for $\mathrm{KL}(p \mid\mid q)$ in (3), and c_q symmetrically for $\mathrm{KL}(q \mid\mid p)$.

[4] Ideally one should tune λ to minimize the language model perplexity on held-out data (e.g., by cross-validation).

[5] This is inspired by the α-skew divergence of Lee (1999,

Our final divergence metric $D(\hat{p}_{\boldsymbol{\theta}}, \hat{q})$ defines D as a linear combination of exclusive and inclusive KL_α divergences, which respectively emphasize $p_{\boldsymbol{\theta}}$'s precision and recall at matching q's bigrams:

$$D(p,q) = (1-\beta) \cdot \frac{\mathrm{KL}_{\alpha_1}(p \,||\, q)}{\mathbb{E}_{\mathbf{y} \sim p}[\,|\mathbf{y}|\,]} + \beta \cdot \frac{\mathrm{KL}_{\alpha_2}(q \,||\, p)}{\mathbb{E}_{\mathbf{y} \sim q}[\,|\mathbf{y}|\,]} \tag{4}$$

where $\beta, \alpha_1, \alpha_2$ are tuned by cross-validation to maximize the downstream parsing performance. The division by average sentence length converts KL from nats per sentence to nats per word,[6] so that the KL values have comparable scale even if B has much longer or shorter sentences than $\mathbf{u}$.

3 Algorithms

3.1 Efficiently computing expected counts

We now present a polynomial-time algorithm for computing the expected bigram counts c_p under $p_{\boldsymbol{\theta}}$ (or equivalently $\hat{p}_{\boldsymbol{\theta}}$), for use above. This averages expected counts from each unordered tree $\mathbf{x} \in B$. Algorithm 1 in the supplement gives pseudocode.

The insight is that rather than sampling a single realization of $\mathbf{x}$ (as B' does), we can use dynamic programming to sum efficiently over all of its exponentially many realizations. This gives an exact answer. It algorithmically resembles tree-to-string machine translation, which likewise considers the possible reorderings of a source tree and incorporates a language model by similarly tracking their surface N-grams (Chiang, 2007, §5.3.2).

For each node a of the tree $\mathbf{x}$, let the POS string $\mathbf{y}_a$ be the realization of the subtree rooted at a. Let $c_a(st)$ be the expected count of bigram st in $\mathbf{y}_a$, whose distribution is governed by equation (1). We allow $s = \text{BOS}$ or $t = \text{EOS}$ as defined in §2.4.2.

The c_a function can be represented as a sparse map from POS bigrams to reals. We compute c_a at each node a of $\mathbf{x}$ in a bottom-up order. The final step computes c_{root}, giving the expected bigram counts in $\mathbf{x}$'s realization $\mathbf{y}$ (that is, c_p in §2.4).

We find c_a as follows. Let $n = n_a$ and recall from §2.2 that $\pi(a)$ is an ordering of $a_1, \ldots, a_n$, where $a_1, \ldots, a_{n-1}$ are the child nodes of a, and a_n is a dummy node representing a's head token. Also, let a_0 and a_{n+1} be dummy nodes that always appear at the start and end of any ordering.

For all $0 \leq i \leq n$ and $1 \leq j \leq n+1$, let $p_a(i,j)$ denote the expected count of the $a_i a_j$ node bigram—the probability that $\pi(a)$ places node a_i immediately before node a_j. These *node bigram probabilities* can be obtained by enumerating all possible orderings π, a matter we return to below.

It is now easy to compute c_a:

$$c_a(st) = c_a^{\text{within}}(st) + c_a^{\text{between}}(st) \tag{5}$$

$$c_a^{\text{within}}(st) = \begin{cases} \sum_{i=1}^{n} c_{a_i}(st) & \text{if } s \neq \text{BOS}, t \neq \text{EOS} \\ 0 & \text{otherwise} \end{cases}$$

$$c_a^{\text{across}}(st) = \sum_{i=0}^{n} \sum_{j=1}^{n+1} p_a(i,j) c_{a_i}(s\,\text{EOS}) c_{a_j}(\text{BOS}\,t)$$

That is, c_a inherits all non-boundary bigrams st that fall within its child constituents (via c_a^{within}). It also counts bigrams st that cross the boundary between consecutive nodes (via c_a^{across}), where nodes a_i and a_j are consecutive with probability $p_a(i,j)$.

When computing c_a via (5), we will have already computed $c_{a_1}, \ldots, c_{a_{n-1}}$ bottom-up. As for the dummy nodes, a_n is realized by the length-1 string h where h is the head token of node a, while a_0 and a_{n+1} are each realized by the empty string. Thus, c_{a_n} simply assigns count 1 to the bigrams $\text{BOS}\,h$ and $h\,\text{EOS}$, and c_{a_0} and $c_{a_{n+1}}$ each assign expected count 1 to BOS EOS. (Notice that thus, $c_a^{\text{across}}(st)$ counts $\mathbf{y}_a$'s boundary bigrams—the bigrams st where $s = \text{BOS}$ or $t = \text{EOS}$—when $i = 0$ or $j = n+1$ respectively.)

3.2 Efficient enumeration over permutations

The main challenge above is computing the node bigram probabilities $p_a(i,j)$. These are marginals of $p(\pi \mid a)$ as defined by (1), which unfortunately is intractable to marginalize: there is no better way than enumerating all $n!$ permutations.

That said, there is a particularly efficient way to enumerate the permutations. The Steinhaus-Johnson-Trotter (SJT) algorithm (Sedgewick, 1977) does so in $O(1)$ time per permutation, obtaining each permutation by applying a single swap to the previous one. Only the features that are affected by this swap need to be recomputed. For our features (Appendix A), this cuts the runtime per permutation from $O(n^2)$ to $O(n)$.

Furthermore, the single swap of adjacent nodes only changes 3 bigrams (possibly including boundary bigrams). As a result, it is possible to

2001). Indeed, we may regard $\mathrm{KL}_\alpha(p \,||\, q)$ as the α-skew divergence between the unigram distributions $p(\cdot \mid s)$ and $q(\cdot \mid s)$, averaged over all s in proportion to $c_p(s)$. In principle, we could have used the α-skew divergence between the distributions $p(\cdot)$ and $q(\cdot)$ over POS sequences $\mathbf{y}$, but computing that would have required a sampling-based approximation (§7).

[6] Recall that the units of negated log-probability are called bits for log base 2, but nats for log base e.

obtain the marginal probabilities with $O(1)$ additional work per permutation. When a node bigram is destroyed, we increment its marginal probability by the total probability of permutations encountered since the node bigram was last created. This can be found as a difference of partial sums. The final partial sum is the normalizing constant $Z(a)$, which can be applied at the end. Pseudocode is given in supplementary material as Algorithm 2.

When we train the parameters $\boldsymbol{\theta}$ (§2.4), we must back-propagate through the whole computation of equation (4), which depends on tag bigram counts $c_a(st)$, which depend via (5) on expected node bigram counts $p_a(i,j)$, which depend via Algorithm 2 on the permutation probabilities $p(\pi \mid a)$, which depend via (1) on the feature weights $\boldsymbol{\theta}$.

4 Heuristics

4.1 Pruning high-degree trees

As a further speedup, we only train on trees with number of words < 40 and $\max_a n_a \leq 5$, so $n_a! \leq 120$.[7] We then produce the synthetic treebank B' (§2.3) by drawing a single realization of each tree in B for which $\max_a n_a \leq 7$. This requires sampling from up to $7! = 5040$ candidates per node, again using SJT.[8]

That is, in this paper we run exact algorithms (§3), but only on a subset of B. The subset is not necessarily representative. An improvement would use importance sampling, with a proposal distribution that samples the slower trees less often during SGD but upweights them to compensate.

§7 suggests a future strategy that would run on all trees in B via approximate, sampling-based algorithms. The exact methods would remain useful for calibrating the approximation quality.

4.2 Minibatch estimation of c_p

To minimize (4), we use the Adam variant of SGD (Kingma and Ba, 2014), with learning rate 0.01 chosen by cross-validation (§5.1).

SGD requires a stochastic estimate of the gradient of the training objective. Ordinarily this is done by replacing an expectation over the entire training set with an expectation over a minibatch.

Equation (2) with $p = \hat{p}_{\boldsymbol{\theta}}$ is indeed an expectation over sentences of B. It can be stochastically estimated as (3) where c_p gives the expected bigram counts averaged over only the sentences in a minibatch of B. These are found using §3's algorithms with the current $\boldsymbol{\theta}$. Unfortunately, the term $\log p(t \mid s)$ depends on bigram counts that should be derived from the *entire* corpus B in the same way. Our solution is to simply reuse the *minibatch* estimate of c_p for the latter counts. We use a large minibatch of 500 sentences from B so that this *drop-in estimate* does not introduce too much bias into the stochastic gradient: after all, we only need to estimate bigram statistics on 17 POS types.[9]

By contrast, the c_q values that are used for the expectation in the second term of (4) and in $\log q(t \mid s)$ do not change during optimization, so we simply compute them once from all of $\mathbf{u}$.

4.3 Informed initialization

Unfortunately the objective (4) is not convex, so the optimizer is sensitive to initialization (see §5.3 below for empirical discussion). Initializing $\boldsymbol{\theta} = \mathbf{0}$ (so that $p(\pi \mid a)$ is uniform) gave poor results in pilot experiments. Instead, we initially choose $\boldsymbol{\theta}$ to be the realization parameters of the source language, as estimated from the source treebank B. This is at least a linguistically realistic $\boldsymbol{\theta}$, although it may not be close to the target language.[10]

For this initial estimation, we follow Wang and Eisner (2016) and perform supervised training on B of the log-linear realization model (1), by maximizing the conditional log-likelihood of B, namely $\sum_{(\mathbf{x},\mathbf{t})\in B} \log p_{\boldsymbol{\theta}}(\mathbf{t} \mid \mathbf{x})$, where $(\mathbf{x},\mathbf{t})$ are an unordered tree and its observed ordering in B. This initial objective is convex.[11]

5 Experiments

We performed a large-scale experiment requiring hundreds of thousands of CPU-hours. To our knowledge, this is the largest study of parsing transfer yet attempted.

[7] We found that this threshold worked much better than ≤ 4 and about as well as the much slower ≤ 6.

[8] This pruning heuristic retains 36.1% of the trees (averaging over the 20 development treebanks (§5.1)) for training, and 66.6% for actual realization. The latter restriction follows Wang and Eisner (2016, §4.2): they too discarded trees with nodes having $n_a \geq 8$.

[9] We also used the minibatch to estimate the average sentence length $\mathbb{E}_{\mathbf{y}\sim p}[\,|\mathbf{y}|\,]$ in (4), although here we could have simply used all of B since this value does not change.

[10] As an improvement, one could also try initial realization parameters for B that are estimated from treebanks of *other* languages. Concretely, the optimizer could start by selecting a "galactic" treebank from Wang and Eisner (2016) that is already close to the target language, according to (4), and try to make it even closer. We leave this to future work.

[11] Unfortunately, we did not regularize it, which probably resulted in initializing some parameters too close to $\pm\infty$ for the optimizer to change them meaningfully.

5.1 Data and setup

As our main dataset, we use Universal Dependencies version 1.2 (Nivre et al., 2015)—a set of 37 dependency treebanks for 33 languages, with a unified POS-tag set and relation label set.

Our evaluation metric was unnormalized attachment score (UAS) when parsing a target treebank with a parser trained on a (possibly permuted) source treebank. For both evaluation and training, we used only the training portion of each treebank.

Our parser was Yara (Rasooli and Tetreault, 2015), a fast and accurate transition-based dependency parser that can be rapidly retrained. We modified Yara to ignore the input words and use only the input *gold* POS tags (see §1.3). To train the Yara parser on a (possibly permuted) source treebank, we first train on 80% of the trees and use the remaining 20% to tune Yara's hyperparameters. We then retrain Yara on 100% of the source trees and evaluate it on the target treebank.

Similar to Wang and Eisner (2017), we use 20 treebanks (18 distinct languages) as *development data*, and hold out the remaining 17 treebanks for the final evaluation. We chose the hyperparameters $(\alpha_1, \alpha_2, \beta)$ of (4) to maximize the target-language UAS, averaged over all 376 transfer experiments where the source and target treebanks were development treebanks of different languages.[12] (See Appendix C for details.)

The next few sections perform some exploratory analysis on these 376 experiments. Then, for the final test in §5.4, we will evaluate UAS on all 337 transfer experiments where the source is a development treebank and the target is a test treebank of a different language.[13]

5.2 Exploratory analysis

We have assumed that a smaller divergence between source and target treebanks results in better transfer parsing accuracy. Figure 1 shows that these quantities are indeed correlated, both for the original source treebanks and for their "made to order" permuted versions.

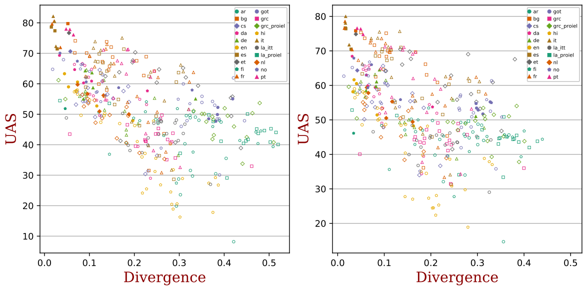


Figure 1: UAS is higher when divergence is lower. Each point represents a pair of source and target languages, whose shape and color identify the treebank of the target language (see legend). The marker is solid if the source and target languages belong to the same language family.[15] The left graph uses the original source treebank (Kendall's $\tau = -0.41$), while the right graph uses its permuted version ($\tau = -0.39$).

Thus, we hope that the optimizer will find a systematic permutation that reduces the divergence. Does it? Yes: Figures 5 and 6 in the supplementary material show that the optimizer almost always manages to reduce the objective on training data, as expected.

One concern is that our divergence metric might misguide us into producing dysfunctional languages whose trees cannot be easily recovered from their surface strings, i.e., they have no good parser. In such a language, the word order might be extremely free (e.g., $\boldsymbol{\theta} = \mathbf{0}$), or common constructions might be syntactically ambiguous. Fortunately, Appendix D shows that our synthetic languages appear natural with respect to their their parsability.

The above findings are promising. So does permuting the source language in fact result in better transfer parsing of the target language? We experiment on the 376 development pairs.

The solid lines in Figure 2 show our improvements on the dev data, with a simpler scatterplot given by in Figure 7 in the supplementary material. The upshot is that the synthetic source treebanks yield a transfer UAS of 52.92 on average. This is not yet a result on held-out test data: recall that 52.92 was the best transfer UAS achieved by any hyperparameter setting. That said, it is 1.00 points better than transferring from the original source treebanks, a significant difference (paired permutation test by language pair, $p < 0.01$).

Figure 2 shows that this average improvement is mainly due to the many cases where the source and target languages come from different families.

[12] We have 19*20=380 pairs in total, minus the four excluded pairs (grc, grc_proiel), (grc_proiel, grc), (la_proiel, la_itt) and (la_itt, la_proiel). Unlike Wang and Eisner (2017), we exclude duplicated languages in development and testing.

[13] Specifically, there are 3 duplicated sets: {grc, grc_proiel}, {la, la_proiel, la_itt}, and {fi, fi_ftb}. Whenever one treebank is used as the target language, we exclude the other treebanks in the same set.

[15] According to the family (and sub-family) information at http://universaldependencies.org.

Permutation tends to improve source languages that were doing badly to start with. However, it tends to hurt a source language that is already in the target language family.

A hypothetical experiment shows that permuting the source does have good *potential* to help (or at least not hurt) in both cases. The dashed lines in Figure 2—and the scatterplot in Figure 8—show the potential of the method, by showing the improvement we would get from permuting each source treebank using an "oracle" realization policy—the supervised realization parameters $\boldsymbol{\theta}$ that are estimated from the actual target treebank. The usefulness of this oracle-permuted source varies depending on the source language, but it is usually much better than the automatically-permuted version of the same source.

This shows that large improvements would be possible if we could only find the best permutation policy allowed by our model family. The question for future work is whether such gains can be achieved by a more sensitive permutation model than (1), a better divergence objective than (4), or a better search algorithm than §4.2. Identifying the best available source treebank, or the best mixture of all source treebanks, would also help greatly.

5.3 Sensitivity to initializer

Figure 2 makes clear that performance of the synthetic source treebanks is strongly correlated with that of their original versions. Most points in Figure 7 lie near the diagonal (Kendall's $\tau = 0.85$). Even with oracle permutation in Figure 8, the correlation remains strong ($\tau = 0.59$), suggesting that the choice of source treebank is important even beyond its effect on search initialization.

We suspected that when "made to order" source treebanks (more than the oracle versions) have performance close to their original versions, this is in part because the optimizer can get stuck near the initializer (§4.3). To examine this, we experimented with random restarts, as follows. In addition to informed initialization (§4.3), we optimized from 5 other starting points $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, I)$. From these 6 runs, we selected the final parameters that achieved the best divergence (4). As shown by Figure 9 in the supplement, greater gains appear to be possible with more aggressive search methods of this sort, which we leave to future work. We could also try non-random restarts based on the realization parameters of other languages, as suggested in footnote 10.

5.4 Final evaluation on the test languages

For our final evaluation (§5.1), we use the same hyperparameters (Appendix C) and report on single-source transfer to the 17 held-out treebanks.

The development results hold up in Figure 3. Using the synthetic languages yields 50.36 UAS on average—1.75 points over the baseline, which is significant (paired permutation test, $p < 0.01$).

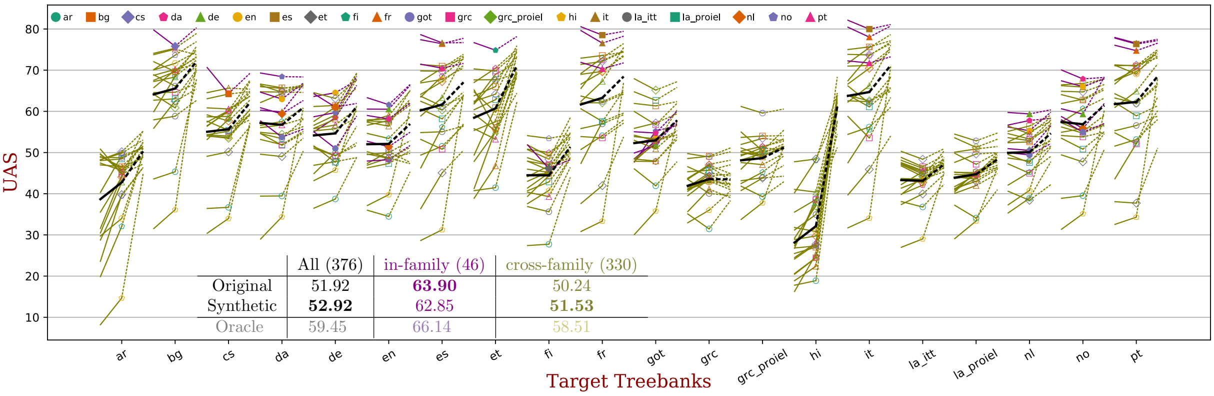
In the supplementary material (Appendix E), we include some auxiliary experiments on multi-source transfer.

6 Related Work

6.1 Unsupervised parsing

Unsupervised parsing has remained challenging for decades (Mareček, 2016). Classical grammar induction approaches (Lari and Young, 1990; Carroll and Charniak, 1992; Klein and Manning, 2004; Headden III et al., 2009; Naseem et al., 2010) estimate a generative grammar to explain the sentences, for example by the Expectation-Maximization (EM) algorithm, and then use it to parse. Some such approaches try to improve the grammar model. For example, Klein and Manning (2004)'s dependency model with valence was the first to beat a trivial baseline; later improvements considered higher-order effects and punctuation (Headden III et al., 2009; Spitkovsky et al., 2012). Other approaches try to avoid search error, using strategies like convexified objectives (Wang et al., 2008; Gimpel and Smith, 2012), informed initialization (Klein and Manning, 2004; Mareček and Straka, 2013), search bias (Smith and Eisner, 2005, 2006; Naseem et al., 2010; Gillenwater et al., 2010), branch-and-bound search (Gormley and Eisner, 2013), and switching objectives (Spitkovsky et al., 2013).

The alternative of cross-lingual transfer has recently flourished thanks to the development of consistent cross-lingual datasets of POS-tagged (Petrov et al., 2012) and dependency-parsed (McDonald et al., 2013) sentences. McDonald et al. (2011) showed a significant improvement over grammar induction by simply using the delexicalized parser trained on other language(s). Subsequent improvements have come from re-weighting source languages (Søgaard, 2011b; Rosa and Žabokrtský, 2015a,b; Wang and Eisner, 2016), adapting the model to the target language using WALS (Dryer and Haspelmath, 2013) features (Naseem et al., 2012; Täckström et al., 2013;



	All (376)	in-family (46)	cross-family (330)
Original	51.92	**63.90**	50.24
Synthetic	**52.92**	62.85	**51.53**
Oracle	59.45	66.14	58.51

Figure 2: Unlabeled attachment scores (UAS) from 376 pairs of development treebanks. Each column represents a target treebank, and each polyline within that column shows transfer from variants of a different source treebank. The three points on the polyline (from left to right) represent the target UAS for parsers trained on three sources: the original source treebank, the "made to order" permutation that attempts to match surface statistics of the target treebank, and an oracle permutation that uses a realization model trained on the target language. We use solid markers and purple lines if the transfer is *within-family* (source and target treebank from the same language family), and hollow and olive for cross-family transfer. The **black** polyline in each column is the mean of the others. The table in the lower left gives summary results; the number in each column header gives the number of points summarized. For each column, we boldface the better result between the "Synthetic" and "Original", or both if they are not significantly different (paired permutation test, $p < 0.01$). We also show the oracle permutation result in row "Oracle".

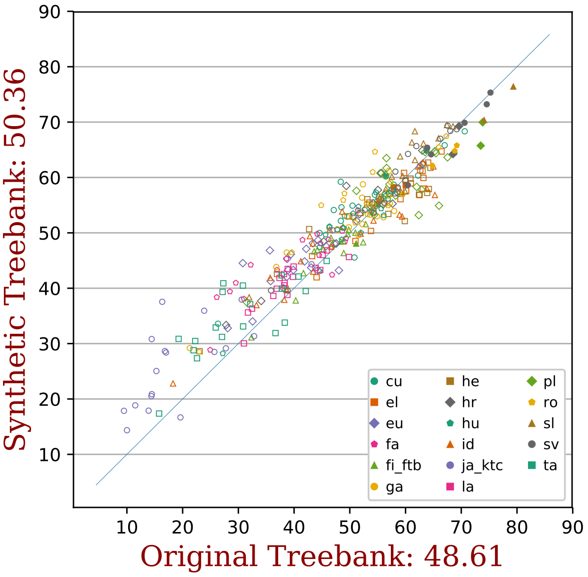


Figure 3: UAS on 337 language pairs from the training languages to the test languages.

Zhang and Barzilay, 2015; Ammar et al., 2016), and improving the lexical representations via multilingual word embeddings (Duong et al., 2015; Guo et al., 2016; Ammar et al., 2016) and synthetic data generation (§6.2).

6.2 Synthetic data generation

Our novel proposal ties into the recent interest in data augmentation in supervised machine learning. In unsupervised parsing, the most widely adopted synthetic data method has been *annotation projection*, which generates synthetic analyses of target-language sentences by "projecting" the analysis from a source-language translation. Of course, this requires bilingual corpora as an additional resource. Annotation projection was proposed by Yarowsky et al. (2001), gained promising results on sequence labelling tasks, and was later developed for unsupervised parsing (Hwa et al., 2005; Ganchev et al., 2009; Smith and Eisner, 2009; Tiedemann, 2014; Ma and Xia, 2014; Tiedemann et al., 2014). Recent work in this vein has mainly focused on improving the synthetic data, including reweighting the training trees (Agić et al., 2016) or pruning those that cannot be aligned well (Rasooli and Collins, 2015, 2017; Lacroix et al., 2016).

On the other hand, Wang and Eisner (2016) proposed to permute source language treebanks using word order realization models trained on other source languages. They generated on the order of 50,000 synthetic languages by "mixing and matching" a few dozen source languages. Their idea was that with a large set of synthetic languages, they could use them as supervised examples to train an unsupervised structure discovery system that could analyze any new language. Systems built with this dataset were competitive in single-source parser transfer (Wang and Eisner, 2016), typology prediction (Wang and Eisner, 2017), and parsing unknown languages (Wang and Eisner, 2018).

Our work in this paper differs in that our synthetic treebanks are “made to order.” Rather than combine aspects of different treebanks and hope to get at least one combination that is close to the target language, we “combine” the source treebank with a POS corpus of the target language, which guides our customized permutation of the source.

Beyond unsupervised parsing, synthetic data has been used for several other tasks. In NLP, it has been used for complex tasks such as question-answering (QA) (Serban et al., 2016) and machine reading comprehension (Weston et al., 2016; Hermann et al., 2015; Rajpurkar et al., 2016), where highly expressive neural models are used and not enough real data is available to train them. In the playground of supervised parsing, Gulordava and Merlo (2016) conduct a controlled study on the parsibility of languages by generating treebanks with short dependency length and low variability of word order.

7 Conclusion & Future Work

We have shown how cross-lingual transfer parsing can be improved by permuting the source treebank to better resemble the target language on the surface (in its distribution of gold POS bigrams). The code is available at `https://github.com/wddabc/ordersynthetic`. Our work is grounded in the notion that by trying to explain the POS bigram counts in a target corpus, we can discover a stochastic realization policy for the target language, which correctly “translates” the source trees into appropriate target trees.

We formulated an objective for evaluating such a policy, based on KL-divergence between bigram models. We showed that the objective could be computed efficiently by dynamic programming, thanks to the limitation to bigram statistics.

Experimenting on the Universal Dependencies treebanks v1.2, we showed that the synthetic treebanks were—on average—modestly but significantly better than the corresponding real treebanks for single-source transfer (and in Appendix E, on multi-source transfer).

On the downside, Figure 7 shows that with our current method, permuting the source language to be more like the target language is helpful (on average) only when the source language is from a different language family. This contrast would be even more striking if we had a better optimizer: Figure 9 shows that SGD’s initialization bias limits its permutation’s benefit for cross-family training, as well as its harm for within-family training.

Several opportunities for future work have already been mentioned throughout the paper. We are also interested in experimenting with richer families of permutation distributions, as well as “conservative” distributions that tend to prefer the original source order. We could use entropy regularization (Grandvalet and Bengio, 2005) to encourage more “deterministic” patterns of realization in the synthetic languages.

We would also like to consider more sensitive divergence measures that go beyond bigrams, for example using recurrent neural network language models (RNNLMs) for $\hat{q}$ and $\hat{p}_{\boldsymbol{\theta}}$. This means abandoning our exact dynamic programming methods; we would also like to abandon exact exhaustive enumeration in order to drop §4.1’s bounds on n. Fortunately, there exist powerful MCMC methods (Eisner and Tromble, 2006) that can sample from interesting distributions over the space of $n!$ permutations, even for large n. Thus, we could approximately sample from $p_{\boldsymbol{\theta}}$ by drawing permuted versions of each tree in B.

Given this change, a very interesting direction would be to graduate from POS language models to word language models, using cross-lingual unsupervised word embeddings (Ruder et al., 2017). This would eliminate the need for the gold POS tags that we unrealistically assumed in this paper (which are typically unavailable for a low-resource target language). Furthermore, it would enable us to harness richer lexical information beyond the 17 UD POS tags. After all, even a (gold) POS corpus might not be sufficient to determine the word order of the target language: “`NOUN VERB NOUN`” could be either subject-verb-object or object-verb-subject. However, “`water drink boy`” is presumably object-verb-subject. Thus, using cross-lingual embeddings, we would try to realize the unordered source trees so that their word strings, with few edits, can achieve high probability under a neural language model of the target.

Acknowledgements

This work was supported by National Science Foundation Grants 1423276 & 1718846. We are grateful to the state of Maryland for the Maryland Advanced Research Computing Center, a crucial resource. We thank Shijie Wu and Adithya Renduchintala for early discussion, Argo lab members for further discussion, and the 3 reviewers for quality comments.

References

Željko Agić, Anders Johannsen, Barbara Plank, Héctor Martínez Alonso, Natalie Schluter, and Anders Søgaard. 2016. Multilingual projection for parsing truly low-resource languages. *Transactions of the Association for Computational Linguistics*, 4:301–312.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah Smith. 2016. Many languages, one parser. *Transactions of the Association of Computational Linguistics*, 4:431–444.

Alena Böhmová, Jan Hajič, Eva Hajičová, and Barbora Hladká. 2003. The Prague dependency treebank. In *Treebanks*, pages 103–127. Springer.

Glenn Carroll and Eugene Charniak. 1992. Two experiments on learning probabilistic dependency grammars from corpora. In *Statistically-Based Natural Language Processing Techniques: Papers from the Workshop*, pages 1–13, Menlo Park: AAAI Press. AAAI. Technical Report WS-92-01.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228.

Matthew S. Dryer and Martin Haspelmath, editors. 2013. *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig. http://wals.info/.

Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. 2015. Cross-lingual transfer for unsupervised dependency parsing without parallel data. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 113–122.

Jason Eisner and Roy W. Tromble. 2006. Local search with very large-scale neighborhoods for optimal permutations in machine translation. In *Proceedings of the HLT-NAACL Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, pages 57–75.

Katja Filippova and Michael Strube. 2009. Tree linearization in English: Improving language model based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 225–228.

Richard Futrell and Edward Gibson. 2015. Experiments with generative models for dependency tree linearization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1978–1983.

Kuzman Ganchev, Jennifer Gillenwater, and Ben Taskar. 2009. Dependency grammar induction via bitext projection constraints. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 369–377.

Jennifer Gillenwater, Kuzman Ganchev, Joo Graa, Fernando Pereira, and Ben Taskar. 2010. Sparsity in dependency grammar induction. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 194–199.

Kevin Gimpel and Noah A. Smith. 2012. Concavity and initialization for unsupervised dependency parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 577–581.

Matthew Gormley and Jason Eisner. 2013. Nonconvex global optimization for latent-variable models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 444–454.

Yves Grandvalet and Yoshua Bengio. 2005. Semi-supervised learning by entropy minimization. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 529–536. MIT Press.

Kristina Gulordava and Paola Merlo. 2016. Multi-lingual dependency parsing evaluation: A large-scale analysis of word order properties using artificial data. *Transactions of the Association for Computational Linguistics*, 4:343–356.

Jiang Guo, Wanxiang Che, David Yarowsky, Haifeng Wang, and Ting Liu. 2016. A representation learning framework for multi-source transfer parsing. In *AAAI*, pages 2734–2740.

William P. Headden III, Mark Johnson, and David McClosky. 2009. Improving unsupervised dependency parsing with richer contexts and smoothing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 101–109.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1684–1692.

Rebecca Hwa, Philip Resnik, Amy Weinberg, Clara Cabezas, and Okan Kolak. 2005. Bootstrapping parsers via syntactic projection across parallel texts. *Natural Language Engineering*, 11(3):311–325.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association of Computational Linguistics*, 4:313–327.

Dan Klein and Christopher Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 478–485.

Ophélie Lacroix, Lauriane Aufrant, Guillaume Wisniewski, and François Yvon. 2016. Frustratingly easy cross-lingual transfer for transition-based dependency parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1058–1063.

Karim Lari and Steve J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4(1):35–56.

Lillian Lee. 1999. Measures of distributional similarity. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 25–32.

Lillian Lee. 2001. On the effectiveness of the skew divergence for statistical language analysis. In *Proceedings of AISTATS*.

Xuezhe Ma and Fei Xia. 2014. Unsupervised dependency parsing with transferring distribution via parallel guidance and entropy regularization. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1337–1348.

David Mareček and Milan Straka. 2013. Stop-probability estimates computed on a large corpus improve unsupervised dependency parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 281–290.

David Mareček. 2016. Twelve years of unsupervised dependency parsing. In *Proceedings of the 16th ITAT Conference on Information Technologies—Applications and Theory*, pages 56–62.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97.

Ryan McDonald and Fernando Pereira. 2006. *Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. thesis, University of Pennsylvania.

Ryan McDonald, Slav Petrov, and Keith Hall. 2011. Multi-source transfer of delexicalized dependency parsers. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 62–72.

Simon Mille, Anja Belz, Bernd Bohnet, Yvette Graham, Emily Pitler, and Leo Wanner. 2018. The first multilingual surface realisation shared task (SR'18): Overview and evaluation results. In *Proceedings of the 1st Workshop on Multilingual Surface Realization (MSR), 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–12.

Tahira Naseem, Regina Barzilay, and Amir Globerson. 2012. Selective sharing for multilingual dependency parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 629–637.

Tahira Naseem, Harr Chen, Regina Barzilay, and Mark Johnson. 2010. Using universal linguistic knowledge to guide grammar induction. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1234–1244.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Cristina Bosco, Gosse Bouma, Sam Bowman, Marie Candito, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Fabricio Chalub, Jinho Choi, Çağrı Çöltekin, Miriam Connor, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Linh Hà Mỹ, Dag Haug, Barbora Hladká, Petter Hohle, Radu Ion, Elena Irimia, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Natalia Kotsyba, Simon Krek, Veronika Laippala, Phng Lê H`ông, Alessandro Lenci, Nikola Ljubešić, Olga Lyashevskaya, Teresa Lynn, Aibek Makazhanov, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Shunsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Nina Mustafina, Kaili Müürisep, Lng Nguy˜ên Thị, Huy`ên Nguy˜ên Thị Minh, Vitaly Nikolaev, Hanna Nurmi, Stina Ojala, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Loganathan Ramasamy, Livy Real, Laura Rituma, Rudolf Rosa, Shadi Saleh, Manuela Sanguinetti, Baiba Saulīte, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Lena Shakurova, Mo Shen, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Larraitz Uria, Gertjan van Noord, Viktor Varga, Veronika Vincze, Jonathan North Washington, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, and Hanzhi Zhu. 2017. Universal dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre et al. 2015. Universal dependencies 1.2. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University in Prague. Data available at http://universaldependencies.org.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*. European Language Resources Association (ELRA).

Yevgeniy Puzikov and Iryna Gurevych. 2018. BinLin: A simple method of dependency tree linearization. In *Proceedings of the First Workshop on Multilingual Surface Realisation*, pages 13–28.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Mohammad Sadegh Rasooli and Michael Collins. 2015. Density-driven cross-lingual transfer of dependency parsers. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 328–338.

Mohammad Sadegh Rasooli and Michael Collins. 2017. Cross-lingual syntactic transfer with limited resources. *Transactions of the Association for Computational Linguistics*, 5:279–293.

Mohammad Sadegh Rasooli and Joel R. Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *Computing Research Repository*, arXiv:1503.06733 (version 2).

Rudolf Rosa and Zdeněk Žabokrtský. 2015a. KLcpos3 — a language similarity measure for delexicalized parser transfer. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 243–249.

Rudolf Rosa and Zdeněk Žabokrtský. 2015b. MST-Parser model interpolation for multi-source delexicalized transfer. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 71–75.

Sebastian Ruder, Ivan Vulić, and Anders Søgaard. 2017. A survey of cross-lingual word embedding models. *Computing Research Repository*, arXiv:1706.04902.

Robert Sedgewick. 1977. Permutation generation methods. *ACM Computing Surveys*, 9(2):137–164.

Iulian Vlad Serban, Alberto García-Durán, Caglar Gulcehre, Sungjin Ahn, Sarath Chandar, Aaron Courville, and Yoshua Bengio. 2016. Generating factoid questions with recurrent neural networks: The 30M factoid question-answer corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 588–598.

David A. Smith and Jason Eisner. 2009. Parser adaptation and projection with quasi-synchronous grammar features. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 822–831.

Noah A. Smith and Jason Eisner. 2005. Guiding unsupervised grammar induction using contrastive estimation. In *International Joint Conference on Artificial Intelligence (IJCAI) Workshop on Grammatical Inference Applications*, pages 73–82.

Noah A. Smith and Jason Eisner. 2006. Annealing structural bias in multilingual weighted grammar induction. In *Proceedings of the International Conference on Computational Linguistics and the Association for Computational Linguistics (COLING-ACL)*, pages 569–576.

Anders Søgaard. 2011a. Data point selection for cross-language adaptation of dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 682–686. Association for Computational Linguistics.

Anders Søgaard. 2011b. Data point selection for cross-language adaptation of dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 682–686.

Valentin I Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2012. Three dependency-and-boundary models for grammar induction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 688–698. Association for Computational Linguistics.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2013. Breaking out of local optima with count transforms and model recombination: A study in grammar induction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*.

Oscar Täckström, Ryan McDonald, and Joakim Nivre. 2013. Target language adaptation of discriminative transfer parsers. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1061–1071.

Jörg Tiedemann. 2014. Rediscovering annotation projection for cross-lingual parser induction. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING): Technical Papers*, pages 1854–1864. Dublin City University and Association for Computational Linguistics.

Jörg Tiedemann, Željko Agić, and Joakim Nivre. 2014. Treebank translation for cross-lingual parser induction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 130–140.

Dingquan Wang and Jason Eisner. 2016. The Galactic Dependencies treebanks: Getting more data by synthesizing new languages. *Transactions of the Association of Computational Linguistics*, 4:491–505. Data available at https://github.com/gdtreebank/gdtreebank.

Dingquan Wang and Jason Eisner. 2017. Fine-grained prediction of syntactic typology: Discovering latent structure with *supervised* learning. *Transactions of the Association for Computational Linguistics (TACL)*, 5.

Dingquan Wang and Jason Eisner. 2018. Surface statistics of an unknown language indicate how to parse it. *Transactions of the Association for Computational Linguistics (TACL)*. To appear.

Qin Iris Wang, Dale Schuurmans, and Dekang Lin. 2008. Semi-supervised convex training for dependency parsing. In *Proceedings of ACL-HLT*, pages 532–540.

Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. 2016. Towards AI-complete question answering: A set of prerequisite toy tasks. In *Proceedings of the International Conference on Learning Representations*.

David Yarowsky, Grace Ngai, and Richard Wicentowski. 2001. Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of the First International Conference on Human Language Technology Research*.

Daniel Zeman and Philip Resnik. 2008. Cross-language parser adaptation between related languages. In *Proceedings of the IJCNLP-08 Workshop on NLP for Less Privileged Languages*.

Yuan Zhang and Regina Barzilay. 2015. Hierarchical low-rank tensors for multilingual transfer parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1857–1867.

A Feature templates for realization

§2.2 assigns a probability distribution to each ordering π of node a and its child nodes. π_i represents the i^{th} node in this ordering. Recall that equation (1) scores each pair (π_i, π_j) for which $i < j$, using a feature vector $\mathbf{f}(\pi, i, j)$.

To construct the feature vector $\mathbf{f}(\pi, i, j)$, we use the following subset of the feature templates of Wang and Eisner (2016). Borrowing their notation, we write t_i for the POS tag of π_i, and we write r_i for the dependency relation of π_i to its parent, or $r_i = \texttt{head}$ in the special case of $\pi_i = a$.

- $\texttt{L}.t_i.r_i$, provided that $r_j = \texttt{head}$. For example, `L.ADJ.amod` will fire on each adjectival modifier with POS `ADJ` to the left of the head.
- $\texttt{L}.t_i.r_i.t_j.r_j$, provided that $r_i \neq \texttt{head}$ and $r_j \neq \texttt{head}$. This feature detects the relative order of two siblings.
- $d.t_i.r_i.t_j.r_j$, where d is `l` (left), `m` (middle), or `r` (right) according to whether the head position h satisfies $i < j < h$, $i < h < j$, or $h < i < j$. For example, `l.nsubj.dobj` will fire on SOV clauses. This is a specialization of the previous feature (in that it also takes the head position into account), and is similarly skipped if $i = h$ or $j = h$.
- $\texttt{A}.t_i.r_i.t_j.r_j$, provided that $j = i + 1$. These *bigram features* detect two adjacent nodes. For this feature, we extend the summation in equation (1) to allow $0 \leq i < j \leq n_a + 1$, taking $t_0 = r_0 = \text{BOS}$ ("beginning of sequence") and $t_{n+1} = r_{n+1} = \text{EOS}$ ("end of sequence"), as in §2.4.2.

These templates are instantiated with all tags and relations that appear in the source treebank. In contrast to Wang and Eisner (2016), the ordering model that we tune on the source treebank is never applied to any other treebank. Thus, there is no need to include tags or relations that do not appear in the source treebank, nor do we need the *back-off* features of Wang and Eisner (2016). Also, for speed, we exclude the "high-order" features from that paper.

B Pseudocode

Algorithm 1 is the algorithm from §3.1 for computing expected POS bigram counts. It calls Algorithm 2.

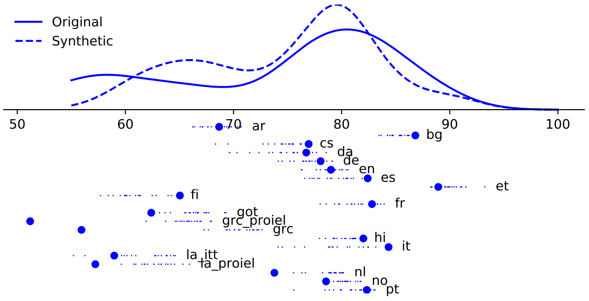


Figure 4: Parsability of 20 real treebanks vs. their many synthetic re-realizations (cf. Wang and Eisner, 2016, Figure 2).

Algorithm 2 is the algorithm from §3.2 for efficiently computing the expected node bigram counts $p_a(i, j)$. The key is that UPDATE is called when a bigram is about to be destroyed; it increments the bigram's unnormalized probability by the cumulative change to the running total $Z(a)$ since that bigram was last created. Each enumerated permutation swaps two adjacent nodes in the previous permutation. This destroys 3 bigrams, so it first calls UPDATE on those (lines 15–17).

C Hyperparameter setting

For tuning hyperparameters in §5.1, we performed a grid search that evaluated all $(\alpha_1, \alpha_2, \beta)$ triples in $\{0.0, 0.2, \ldots, 1\}^3$. The optimal setting was $(\alpha_1, \alpha_2, \beta) = (0.2, 1, 0.2)$.

For multi-source transfer (Appendix E), we reused the same synthetic treebanks B' that we generated for single-source transfer, and tuned only the augmentation ratio g. the optimal setting was 0.2 for all 3 approaches.

D Parsability

For reasons explained in §5.2, we evaluated the parsability of our "made to order" synthetic languages, when the parser was given only POS sequences as input. For each synthetic treebank B', we trained the Yara parser on a training portion and evaluated its UAS on a development portion. In fact, the synthetic treebanks were slightly *more* parsable than the originals (mean UAS of 74.96 vs. 73.61), though the improvement was far from significant under an unpaired permutation test ($p = 0.48$). By contrast, Wang and Eisner (2016) produced synthetic treebanks that were significantly *less* parsable. We observed some regression to the mean: highly parsable treebanks usually became less parsable when permuted, and vice-versa.

Algorithm 1 A recursive routine (§3.1) for computing the expected bigram counts c_a from $p_{\boldsymbol{\theta}}$. c_{root} is the c_p function needed by §2.4.

```
Input: A node $a$ in the dependency tree; current model parameters $\boldsymbol{\theta}$
Output: Sparse map $c_a$ where $c_a[st]$ gives the expected count $c_a(st)$ for each POS bigram $st$
procedure ECOUNTNODE($a, \boldsymbol{\theta}$)
    $a_0 = \textsc{bos}$; $(a_1, \ldots, a_{n-1}) = \text{children}(a)$; $a_n = \text{head}(a)$; $a_{n+1} = \textsc{eos}$    ▷ $\vec{a}$ is the node sequence defined in §3.1
    $c_a \leftarrow \{\,\}$    ▷ map we're constructing, initialized to empty; undefined count $c_a[st]$ can be interpreted as 0
    for $i = 1$ to $n - 1$ do
        $c_{a_i} \leftarrow$ ECOUNTNODE($a_i$)    ▷ recursively compute expected counts for any subtrees rooted at children($a$)
    $c_{a_n} \leftarrow \{\textsc{bos}\,h \mapsto 1, h\,\textsc{eos} \mapsto 1\}$ where $h = \text{POS}(\text{head}(a))$    ▷ serves as the base case of the recursive routine
    $c_{a_0} \leftarrow \{\textsc{bos eos} \mapsto 1\}$    ▷ dummy boundary nodes
    $c_{a_{n+1}} \leftarrow \{\textsc{bos eos} \mapsto 1\}$
    $p_a \leftarrow$ LAZYCOMPUTE($\vec{a}, \boldsymbol{\theta}$)    ▷ call Algorithm 2 for node bigram probs $p_a$ (as defined above equation (5))
    for $i = 1$ to $n$ do
        for $st \in \text{keys}(c_{a_i})$ such that $s \neq \textsc{bos}, t \neq \textsc{eos}$ do
            $c_a[st] \mathrel{+}= c_{a_i}[st]$    ▷ increase $c_a[st]$ by $c_a^{\textit{within}}[st]$ using equation (5)
    for $i = 0$ to $n$ do
        for $j = 1$ to $n + 1$ such that $j \neq i$ do
            for $s, t$ such that $s\,\textsc{eos} \in \text{keys}(c_{a_i})$ and $\textsc{bos}\,t \in \text{keys}(c_{a_j})$ do
                $c_a[st] \mathrel{+}= p_a[i, j] \cdot c_{a_i}[s\,\textsc{eos}] \cdot c_{a_j}[\textsc{bos}\,t]$    ▷ increase $c_a[st]$ by $c_a^{\textit{across}}[st]$ using equation (5)
    return $c_a$
```

Algorithm 2 Computing Node Bigram Probabilities

```
Input: Sequence of nodes $\vec{a} = (a_1, \ldots, a_n)$; current model parameters $\boldsymbol{\theta}$
Output: Array $p$ where $p[i, j]$ = marginal probability of node bigram $a_i a_j$ for all $0 \leq i < n+1, 0 < j \leq n+1$ with $j \neq i$
 1: procedure LAZYCOMPUTE($\vec{a}, \boldsymbol{\theta}$)
 2:     $p \leftarrow \mathbf{0}$    ▷ initialize all marginal bigram probabilities to zero
 3:     $t \leftarrow 0$    ▷ number of permutations considered so far
 4:     $Z^{(t)} \leftarrow 0$    ▷ $Z^{(t)}$ is always total unnormalized probability of first $t$ permutations
 5:     $o_i \leftarrow t$ for $0 \leq i < n + 1$    ▷ $o_i$ is the latest permutation at which bigram $(\pi_i, \pi_{i+1})$ was not yet adjacent
 6:     $\pi \leftarrow (1, 2, \ldots, n)$    ▷ initialize $\pi$ to be identity permutation, $(\forall i)\pi_i = i$
 7:     procedure UPDATE($i$)
 8:         ▷ This procedure updates the unnormalized marginal probability of the bigram $(\pi_i, \pi_{i+1})$, which is about to change
 9:         $p[\pi_i, \pi_{i+1}] \mathrel{+}= Z^{(t)} - Z^{(o_i)}$    ▷ total partial sum of $Z(a)$ since $(\pi_i, \pi_{i+1})$ acquired its current value
10:         $o_i \leftarrow t$    ▷ current time is last time at which $(\pi_i, \pi_{i+1})$ will have its current value (until later)
11:     $w \leftarrow \boldsymbol{\theta} \cdot \sum_{1 \leq i < j \leq n} \mathbf{f}(\pi, i, j)$    ▷ unnormalized log-probability of $\pi$ from equation (1)
12:     $t \leftarrow t + 1$; $Z^{(t)} \leftarrow Z^{(t-1)} + \exp w$    ▷ add the first permutation's unnormalized prob into $Z$
13:     ▷ SJT iterates over a sequence of $n! - 1$ swaps, to get the remaining permutations
14:     for $k$ in SJT($n$) do    ▷ here $1 \leq k < n$, meaning to swap $(\pi_k, \pi_{k+1})$
15:         UPDATE($k - 1$)    ▷ increment prob of current bigram $(\pi_{k-1}, \pi_k)$ before that bigram goes away
16:         UPDATE($k$)    ▷ similarly for $(\pi_k, \pi_{k+1})$
17:         UPDATE($k + 1$)    ▷ similarly for $(\pi_{k+1}, \pi_{k+2})$
18:         SWAP($\pi_k, \pi_{k+1}$)
19:         ▷ Update $w$ from line 11 using only the difference of feature vectors, which is sparse and computable in $O(n)$ time
20:         $w \leftarrow w + \boldsymbol{\theta} \cdot \sum_{1 \leq i < j \leq n} (\mathbf{f}(\pi, i, j) - \mathbf{f}(\pi_{\text{old}}, i, j))$    ▷ where $\pi_{old}$ is the pre-swap $\boldsymbol{\theta}$ and is similar to $\boldsymbol{\theta}$
21:         $t \leftarrow t + 1$; $Z^{(t)} \leftarrow Z^{(t-1)} + \exp w$    ▷ add the new permutation's unnormalized prob into $Z$ (same as line 12)
22:     for $i = 1$ to $n$ do    ▷ count all bigrams in final permutation as we move on from it
23:         UPDATE($i$)
24:     for $i = 0$ to $n$ do
25:         for $j = 1$ to $n + 1$ such that $j \neq i$ do
26:             $p[i, j] \leftarrow \frac{p[i,j]}{Z^{(t)}}$    ▷ normalize the probabilities
27:     return the array $p$
```

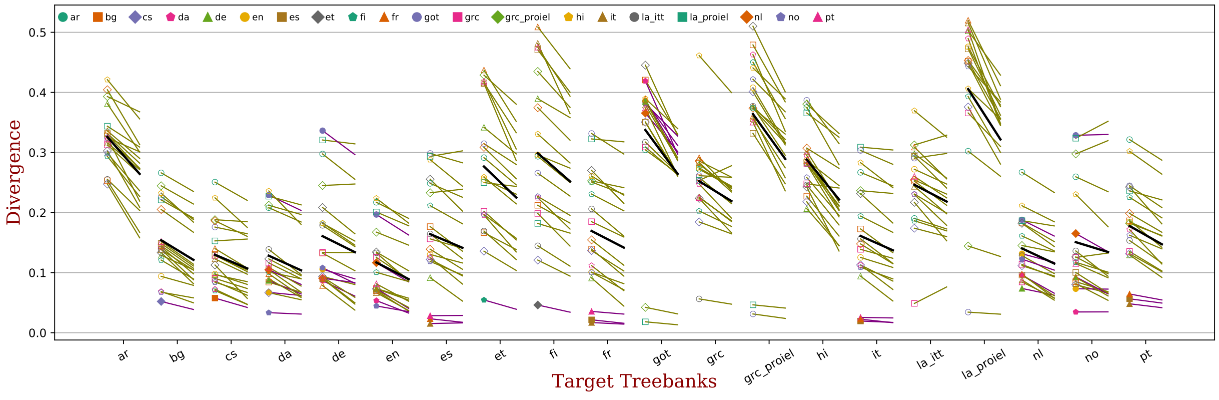


Figure 5: Divergences between 376 pairs of development treebanks. This is a different presentation of Figure 6 in which the source-target pairs are grouped into columns. Each column represents a target treebank, and each line segment within that column shows the divergence equation (4) from variants of a different source treebank. The two points on that segment (from left to right) represent the original source treebank and its "made ot order" permutation. We use solid markers and purple lines if the transfer is *within-family* (source and target treebank from the same language family), and hollow and olive for *cross-family* transfer. The **black** segment in each column is the mean of the others.

E Multi-source transfer

While the main paper considers *single-source* transfer parsers, we are also interested in whether *multi-source* transfer parsers can be improved by *augmenting* the source treebanks with synthetic (permuted) versions.

In each of these experiments, we trained the delexicalized parser by sampling 50000 sentences with replacement from one or more source treebanks, and then tested it on the target treebank. We considered the following methods for sampling a sentence:

Single-source selection (Rosa and Žabokrtský, 2015a; Wang and Eisner, 2016): Sample all sentences uniformly from a single source treebank, namely the one whose trigram POS language model has the highest likelihood on the unparsed corpus of the target language. This method considers multiple sources only to select one.

Equal mixture : Select one of the source treebanks uniformly at random, then sample a sentence uniformly from that treebank. To succeed on this mixture of source treebanks, this source parser must, in effect, analyze the input POS sequence to determine what sort of parse tree is called for in the input language, and we hope that this will also work on the target language.[18]

Unequal mixture : As above, but the selection probability of each source treebank is proportional to its $KL_{cpos^3}^{-4}$ similarity to the target corpus Rosa and Žabokrtský (2015a),[19] which is again determined from the POS trigrams of the two corpora.

In any of these three methods, we can use either the collection of original source treebanks ($g = 0$), or the collection of permuted versions that have been permuted to resemble our target language ($g = 1$). These two collections are the same size. For each sentence that we sample, we use a coin with weight $g \in [0, 1]$ to decide which collection to use. See Appendix C for the value of the hyperparameter g. Notice that the single-source selection method now really becomes double-source selection—we separately select one real and one

[17] For speed, we restricted the experiment of Figure 9 to choose 48 of the 376 pairs. The source treebanks were en, no, de, es, fr, pt, hi, it, ar. The target treebanks were fr, hi, de, ar, pt, en. This covers both in-family transfer and cross-family transfer. By excluding the cases where source = target, we got $9 * 6 - 6 = 48$ pairs.

[18] This is inspired by McDonald et al. (2011)'s method of concatenating the source treebanks. However, our version does not give more weight to treebanks with more sentences (although it does effectively give more weight to treebanks whose sentences are longer).

[19] In contrast, (Rosa and Žabokrtský, 2015b) used these probabilities to interpolate among separately trained source parsers (specifically, interpolating the linear scoring functions of trained instances of MSTParser). We use them to mix treebanks before training a single parser (an instance of Yara parser).

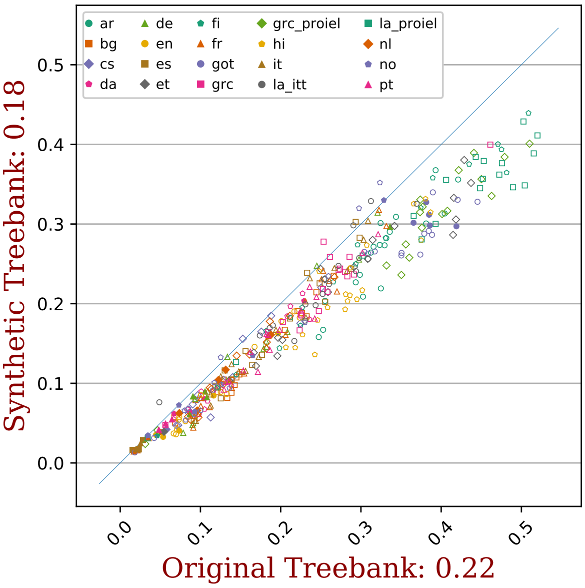


Figure 6: This graph plots the x-axes from the two graphs in Figure 1 against each other. We see that for almost every source-target pair (330/376 = 96.01% of the pairs), the SGD optimizer succeeded in constructing a permuted source treebank B' with lower divergence to the target than the original source treebank B. The diagonal line $y = x$ is also shown for readability. The number on each axis is the mean value.

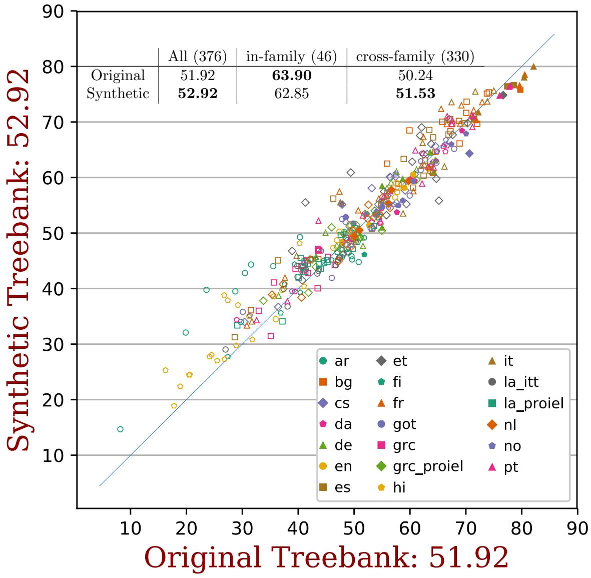
	Selection	Mix=	Mix≠
Original	**64.37**	62.31	64.77
+Synthetic	**64.55**	**62.77**	**65.00**

Table 1: Cross-validation results on UAS using multi-source transfer. "Original" uses the original treebanks from UD ($g = 0$), and "+Synthetic" augments with synthetic languages (allowing $g > 0$). Within each column, we highlight the better result, as well as the other if it is not significantly worse (paired permutation test by language, $p < 0.05$).

synthetic treebank. Similarly, in the unequal mixture method, we have two sets of mixture weights, one for each treebank collection.

The data split is shown in Table 2. In this setting, we tuned the hyperparameters a bit differently than in §5.1, using 5-fold cross validation with the 5-fold split shown in Table 2. That is, to evaluate a given hyperparameter setting, we evaluated the unlabeled attachment score (UAS) on each group of 4 treebanks when transferring parsers from the other 16.

Both during hyperparameter tuning and during testing, we excluded any additional treebanks of the target language from the collection of sources, just as in footnotes 12–13.



	All (376)	in-family (46)	cross-family (330)
Original	51.92	**63.90**	50.24
Synthetic	**52.92**	62.85	**51.53**

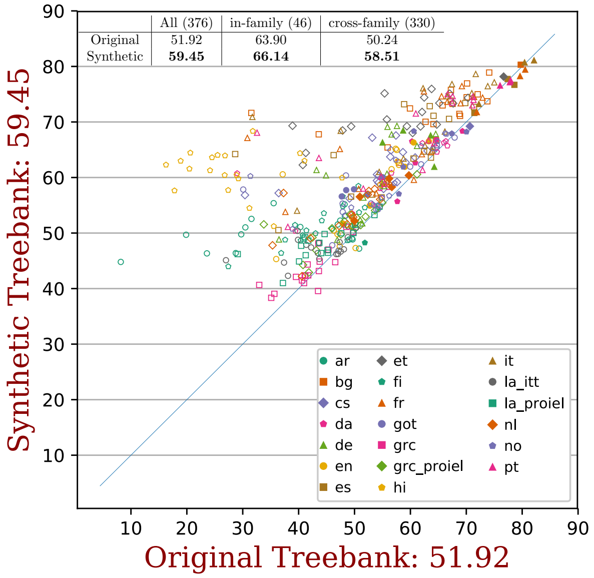
Figure 7: Unlabeled attachment scores (UAS) on 376 language pairs within the training languages. Each marker represents one pair, whose x-axis is the UAS on the target language using the original treebank of the source language, and the y-axis is the UAS using the synthetic treebank permuted from the original treebank. The table in the upper left gives summary results; the number in each column header gives the number of points summarized. For each column, we boldface the better result, as well as the other if it is not significantly worse (paired permutation test, $p < 0.01$).

Train					Test
bg	en	de	pt	hi	la, hr, ga, he, hu,
es	la_proiel	fr	no	cs	fa, ta, cu, el, ro,
grc_proiel	la_itt	it	et	grc	sl, ja_ktc, sv,
ar	fi	got	nl	da	fi_ftb, id, eu, pl

Table 2: Data split of the 37 treebanks (33 languages) from Wang and Eisner (2016, 2017). The dashed lines in "Train" separate the 5 folds.

As shown in Table 1, the improvement from adding the synthetic treebanks ($g > 0$ compared to $g = 0$) varies for different methods. Specifically, the synthetic treebanks do not significantly aid single-source selection, which is reasonable because the selection criteria is more likely to pick a source treebank that belongs to the same language family as the target language,[20] and as we have seen, these cases are not effectively improved by permutation (§5.2). They do significantly improve the mixing methods, because source treebanks from other families contribute to the parsing model, and these are improved by our approach.

[20] In the $g = 0$ experiment, 12/20 target languages selected their single source from the same family.

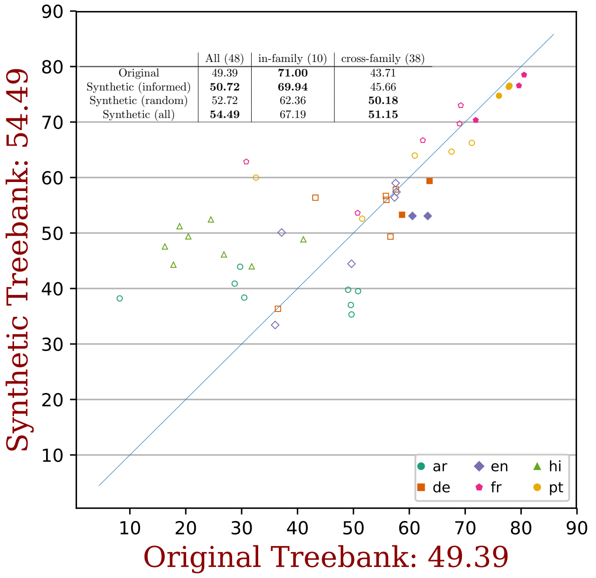


	All (376)	in-family (46)	cross-family (330)
Original	51.92	63.90	50.24
Synthetic	**59.45**	**66.14**	**58.51**

Figure 8: UAS on 376 language pairs within the training languages. The design is similar to Figure 7, but the synthetic treebanks are generated using an oracle—the actual realization model of the target language.

F Full result tables

We show breakdown results for multi-source transfer in Table 3 and for single-source transfer in Table 4.



	All (48)	in-family (10)	cross-family (38)
Original	49.39	**71.00**	43.71
Synthetic (informed)	**50.72**	**69.94**	45.66
Synthetic (random)	52.72	62.36	**50.18**
Synthetic (all)	**54.49**	67.19	**51.15**

Figure 9: UAS on 48 of the language pairs within the development languages.[17] The design is similar to Figure 7, but we optimize divergence more aggressively by selecting the best of 6 optimization runs for each pair (informed initialization plus 5 random restarts). In 36 of 48 cases, the best run used a random restart. The average x and y values are given in the first and last rows of the table, with the intermediate rows showing the results if we had used *only* informed initialization or *only* random restarts. Each column boldfaces the best result as well as all others that are not significantly worse (paired permutation test, $p < 0.01$).

	Selection		Mix=		Mix≠	
Target	orig.	+syn.	orig.	+syn.	orig.	+syn.
ar	48.08	**51.83**	47.5	**48.08**	**51.69**	**51.62**
bg	**80.66**	80.25	**76.97**	**77.77**	**82.06**	81.87
cs	**70.67**	69.52	**67.33**	**67.39**	66.59	**67.28**
da	**69.86**	**69.94**	**70.08**	**69.83**	70.47	**70.87**
de	**64.27**	63.65	64.97	**65.44**	**65.66**	**65.51**
en	**64.00**	**63.91**	62.57	**63.13**	**63.30**	**63.57**
es	77.74	**77.85**	**75.58**	**75.26**	**79.14**	**79.16**
et	**76.00**	**75.77**	67.11	**69.26**	**75.94**	**76.11**
fi	**50.38**	**50.47**	**51.19**	**51.21**	**51.56**	**51.56**
fr	**80.51**	**80.57**	**77.89**	**77.96**	**80.66**	**80.83**
got	**68.20**	**67.58**	62.18	**62.75**	**68.23**	**67.93**
grc	**42.49**	**42.56**	**48.94**	**49.19**	**44.07**	**44.22**
grc_proiel	**61.28**	**61.52**	56.99	**57.19**	**61.60**	61.4
hi	**41.39**	**41.60**	28.59	**31.42**	35.06	**37.62**
it	**82.01**	**81.88**	**79.62**	**79.62**	81.9	**81.94**
la_itt	48.61	**50.00**	**50.84**	**51.07**	**51.89**	**52.06**
la_proiel	54.02	**54.62**	52.14	**52.51**	**55.13**	**55.23**
nl	**59.27**	**59.08**	**59.94**	**60.81**	61.16	**61.46**
no	**70.33**	**70.38**	**69.37**	**69.39**	**71.54**	**71.53**
pt	**77.69**	**78.07**	**76.34**	**76.22**	77.68	**78.19**

Table 3: Breakdown results from Table 1. For each language and method, we boldface the better result, as well as the other if it is not significantly worse (paired permutation test by sentence, $p < 0.05$). Notice that for the Mix= method, augmenting with synthetic permuted languages always yields a boldfaced result.

	bg	es	grc_proiel	ar	en	la_proiel	la_itt	fi	de	fr	it	got	pt	no	et	nl	hi	cs	grc	da
bg	-	69.66	**60.85**	**45.34**	**71.65**	**63.05**	**58.83**	**68.48**	**68.34**	**70.04**	**75.11**	66.13	**70.18**	**73.65**	**62.50**	**69.67**	**36.11**	75.81	**64.64**	**75.33**
es	70.99	-	60.32	**51.54**	**67.74**	58.18	**55.05**	**56.21**	**63.34**	76.42	76.64	61.23	70.49	**70.50**	**45.07**	**67.23**	**31.25**	**69.76**	50.81	**68.55**
grc_proiel	**54.02**	**49.28**	-	39.27	**50.23**	50.42	**43.89**	**45.23**	**49.77**	**47.06**	48.93	59.58	49.44	**51.04**	**43.81**	**51.20**	**37.80**	**53.44**	-	**51.50**
ar	46.58	44.78	45.63	-	**34.00**	**48.46**	**49.82**	**32.08**	**42.81**	46.48	45.83	**48.75**	45.25	**39.50**	**39.78**	**44.04**	**14.68**	**50.18**	**49.26**	**44.33**
en	**57.78**	**57.40**	**48.69**	34.49	-	**47.34**	**49.97**	**53.42**	**60.52**	**59.00**	**56.41**	**48.26**	**48.56**	61.62	48.68	**51.42**	**39.77**	**58.11**	**50.25**	58.15
la_proiel	**50.87**	45.14	**51.26**	**34.09**	**44.34**	-	-	**44.88**	**43.80**	**41.99**	**43.58**	**52.84**	44.78	**45.50**	**43.01**	**44.51**	**33.37**	**49.65**	47.15	44.59
la_itt	45.57	**46.18**	44.19	36.78	43.20	-	-	**44.08**	**43.44**	43.55	44.78	45.21	**45.62**	**45.34**	**39.95**	**42.71**	**29.03**	48.37	46.54	**42.10**
fi	47.00	**46.78**	**45.02**	**27.75**	49.15	**42.86**	35.62	-	**45.70**	**44.38**	**45.01**	**45.32**	39.30	53.44	46.12	**45.18**	**40.81**	48.38	47.07	**49.99**
de	**61.44**	**61.05**	**55.77**	**38.72**	**64.51**	47.66	49.20	**50.03**	-	**58.11**	**59.12**	51.00	**56.68**	**59.71**	**47.79**	61.03	**45.75**	63.13	49.22	**58.45**
fr	**73.57**	78.51	**62.09**	**54.09**	**69.71**	57.54	**56.97**	**57.46**	**67.28**	-	76.56	**62.37**	70.34	**73.00**	**41.96**	**69.62**	**33.36**	**72.12**	**53.56**	**72.35**
it	**75.65**	79.97	62.53	**56.19**	**71.14**	61.09	**62.34**	**55.53**	**66.24**	78.03	-	61.98	**71.74**	**75.48**	**45.91**	**70.45**	**34.09**	**73.70**	53.53	**73.57**
got	**61.33**	**53.35**	65.16	41.92	**53.42**	62.67	47.83	**52.03**	**51.71**	47.94	**50.89**	-	**52.85**	**55.20**	**52.51**	**52.85**	**35.80**	**57.17**	56.76	**54.74**
pt	71.02	76.34	61.99	**53.17**	69.09	58.92	**56.57**	**52.20**	**64.89**	74.74	76.55	61.82	-	**70.26**	37.72	**69.62**	**34.31**	**71.19**	52.10	**71.04**
no	**66.77**	62.74	**55.85**	**39.53**	65.99	50.82	**54.71**	**60.67**	59.33	**62.97**	**65.91**	54.97	55.14	-	47.73	55.86	**35.14**	64.72	53.79	67.88
et	**66.02**	**60.89**	**67.57**	**41.48**	59.79	**62.84**	**55.50**	74.84	55.22	**46.78**	**57.47**	**69.03**	53.22	**67.69**	-	55.84	**55.14**	**64.18**	69.80	**70.47**
nl	**52.60**	**56.46**	**50.44**	**38.91**	55.29	47.57	47.93	**45.24**	59.38	**52.89**	**55.09**	**49.42**	**54.53**	50.52	38.41	-	**40.81**	**53.30**	**44.96**	**57.79**
hi	**27.02**	**24.45**	**37.04**	**18.89**	30.81	**37.88**	**34.96**	**48.18**	**40.39**	**22.38**	**25.31**	**38.82**	**28.07**	**27.31**	**48.42**	**29.74**	-	**27.74**	**38.60**	**24.50**
cs	64.33	**64.21**	53.48	**36.69**	**53.65**	**55.41**	**54.00**	**58.09**	**58.78**	**60.03**	**65.64**	55.42	**60.58**	**60.11**	**50.16**	**57.66**	**33.99**	-	**55.16**	**60.16**
grc	**49.11**	**43.06**	-	31.46	**42.81**	**45.70**	40.05	**43.53**	**44.07**	**41.10**	**43.31**	48.92	**44.63**	**46.76**	**45.07**	**46.96**	**36.00**	**44.27**	-	**47.14**
da	**65.72**	**64.69**	**54.42**	**39.46**	62.99	**51.93**	**53.39**	**57.54**	59.87	**64.33**	**65.58**	53.74	**56.70**	68.43	**49.03**	**59.39**	**34.39**	64.65	51.76	-
cu	**64.84**	**55.15**	64.42	45.55	**56.87**	65.82	**49.97**	**54.62**	**52.16**	**50.95**	**54.11**	68.34	**55.78**	**59.22**	**54.91**	**54.12**	**33.59**	**60.19**	60.35	**58.69**
el	62.69	56.82	58.68	**45.80**	**59.49**	50.34	**57.11**	48.52	61.31	58.95	57.99	57.61	59.92	**60.82**	41.97	56.08	**39.93**	64.70	**58.24**	**57.97**
eu	**48.68**	**40.02**	**45.31**	**32.79**	**44.84**	**46.14**	**43.73**	**41.29**	43.24	**34.01**	**44.53**	**42.07**	**43.28**	**43.29**	**48.12**	**47.11**	**48.03**	**47.60**	43.22	**46.83**
fa	**50.33**	**48.78**	42.43	**45.96**	**38.08**	**48.37**	**49.03**	**39.40**	**45.34**	**48.73**	**48.11**	**50.91**	**47.47**	**40.97**	**38.37**	**43.30**	**28.86**	**54.08**	**49.82**	**44.22**
fi_ftb	**49.76**	**47.07**	**51.10**	**31.12**	**50.56**	**46.65**	37.79	-	**51.57**	**42.72**	**49.08**	**48.42**	**47.99**	**54.20**	48.06	**48.54**	**46.93**	**50.03**	48.29	46.35
ga	55.21	52.82	**53.29**	**55.77**	51.58	49.57	**51.01**	**46.43**	**52.90**	**55.08**	55.33	53.00	53.96	**56.35**	**43.84**	**50.51**	**29.18**	**61.18**	**50.65**	**58.77**
he	59.80	57.91	61.15	**55.17**	**52.93**	53.43	**56.49**	**50.67**	**53.00**	52.12	56.90	61.75	57.22	**56.07**	**42.57**	**53.60**	**28.62**	62.36	**55.46**	**53.86**
hr	64.23	62.44	**52.19**	**37.70**	**55.99**	**55.34**	**54.35**	**58.48**	**57.13**	**57.74**	**64.86**	**55.13**	57.31	**53.39**	**48.44**	55.28	**33.35**	**69.26**	**48.08**	**60.81**
hu	**56.65**	**50.57**	**53.06**	**28.23**	**54.81**	**47.40**	43.08	**53.83**	**57.33**	**49.67**	**50.87**	**48.79**	**51.16**	**56.98**	50.03	56.10	**53.50**	53.15	**54.38**	**54.74**
id	**62.73**	58.01	49.84	**48.08**	**37.91**	52.21	**43.00**	**49.40**	**41.87**	53.28	62.00	52.95	56.84	**50.61**	**36.96**	**47.48**	**22.80**	**62.00**	**44.87**	**53.85**
ja_ktc	**20.87**	**18.85**	**35.94**	**14.36**	**28.50**	**37.55**	**28.39**	**50.45**	31.34	**17.89**	**17.86**	**30.82**	**20.52**	**29.15**	**44.33**	16.67	62.09	**25.05**	**37.95**	**28.65**
la	**46.83**	**39.91**	**43.68**	**30.04**	**40.52**	-	-	**43.91**	**38.81**	**36.32**	**38.62**	**46.01**	**41.86**	**42.55**	**42.06**	**40.97**	**35.62**	43.28	45.64	**43.49**
pl	65.75	**64.74**	53.20	53.27	**57.12**	57.79	58.31	**57.59**	**58.78**	**60.78**	**64.39**	54.91	63.64	**61.70**	**60.45**	**56.73**	**37.49**	69.97	**64.60**	**63.49**
ro	**67.44**	64.81	**55.10**	**51.82**	**59.18**	**53.32**	**56.65**	**54.98**	**57.10**	62.13	65.79	54.93	57.00	**60.97**	**46.47**	**55.91**	**28.52**	**65.48**	**55.33**	**64.64**
sl	70.37	**69.47**	**56.33**	**39.64**	**63.15**	**56.11**	57.49	**63.81**	**67.06**	**68.35**	**69.23**	57.37	**57.92**	**66.13**	**54.58**	**60.26**	**38.36**	76.41	**60.16**	**66.34**
sv	**68.72**	**66.99**	**58.19**	**39.60**	**69.88**	**55.46**	**57.12**	**64.25**	**65.37**	**65.67**	**69.35**	58.62	**61.06**	**75.33**	53.59	**64.17**	**39.60**	**68.41**	**59.44**	73.22
ta	**40.48**	**27.35**	**39.34**	**17.37**	**42.11**	**40.89**	**37.11**	**42.32**	39.48	**28.80**	**30.86**	**32.91**	**30.47**	**39.92**	**44.93**	31.90	**57.59**	**33.10**	33.79	**31.21**

Table 4: UAS scores on single-source transfer results using the synthetic languages, where the columns represent source treebanks and the rows represent target treebanks. The upper half of the table is the 5-fold cross-validation result used for generating the y-axis of Figure 7. The lower half is the final test result used for the y-axis of Figure 3. For each pair, we boldface the results that are not significantly worse (paired permutation test by sentence, $p < 0.05$) than using the original treebanks.