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 *backoff* 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.

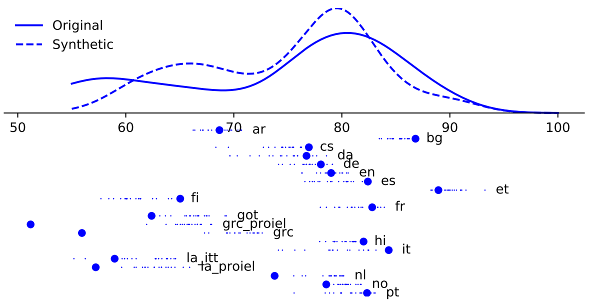


Figure 4: Parsability of 20 real treebanks vs. their many synthetic re-realizations (cf. Wang and Eisner, 2016, Figure 2).

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 $st$
 1: procedure ECOUNTNODE($a, \boldsymbol{\theta}$)
 2:     $a_0 = \text{BOS}$; $(a_1, \ldots, a_{n-1}) = \text{children}(a)$; $a_n = \text{head}(a)$; $a_{n+1} = \text{EOS}$    ▷ $\vec{a}$ is the node sequence defined in §3.1
 3:     $c_a \leftarrow \{\}$    ▷ map we're constructing, initialized to empty; undefined count $c_a[st]$ can be interpreted as 0
 4:     for $i = 1$ to $n - 1$ do
 5:         $c_{a_i} \leftarrow$ ECOUNTNODE($a_i$)    ▷ recursively compute expected counts for any subtrees rooted at children($a$)
 6:     $c_{a_n} \leftarrow \{\text{BOS}\, h \mapsto 1, h\, \text{EOS} \mapsto 1\}$ where $h = \text{POS}(\text{head}(a))$    ▷ serves as the base case of the recursive routine
 7:     $c_{a_0} \leftarrow \{\text{BOS EOS} \mapsto 1\}$    ▷ dummy boundary nodes
 8:     $c_{a_{n+1}} \leftarrow \{\text{BOS EOS} \mapsto 1\}$
 9:     $p_a \leftarrow$ LAZYCOMPUTE($\vec{a}, \boldsymbol{\theta}$)    ▷ call Algorithm 2 for node bigram probs $p_a$ (as defined above equation (5))
10:     for $i = 1$ to $n$ do
11:         for $st \in \text{keys}(c_{a_i})$ such that $s \neq \text{BOS}, t \neq \text{EOS}$ do
12:             $c_a[st]$ += $c_{a_i}[st]$    ▷ increase $c_a[st]$ by $c_a^{\text{within}}[st]$ using equation (5)
13:     for $i = 0$ to $n$ do
14:         for $j = 1$ to $n + 1$ such that $j \neq i$ do
15:             for $s, t$ such that $s\,\text{EOS} \in \text{keys}(c_{a_i})$ and $\text{BOS}\, t \in \text{keys}(c_{a_j})$ do
16:                 $c_a[st]$ += $p_a[i,j] \cdot c_{a_i}[s\,\text{EOS}] \cdot c_{a_j}[\text{BOS}\, t]$    ▷ increase $c_a[st]$ by $c_a^{\text{across}}[st]$ using equation (5)
17:     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:         $p[\pi_i, \pi_{i+1}]$ += $Z^{(t)} - Z^{(o_i)}$    ▷ 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 $p$
```

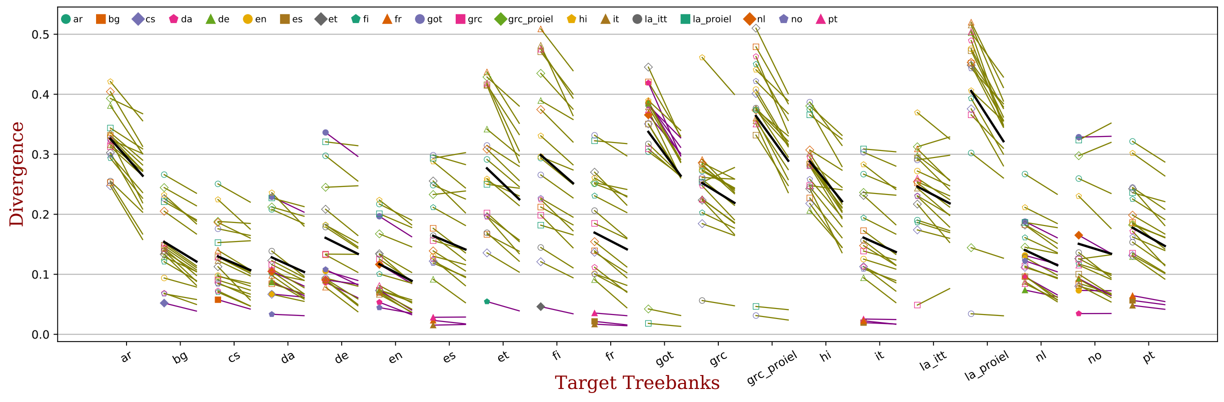


Figure 5: Divergences between 376 pairs of development treebanks. This is a different presentation of Figure 6 in which the source-target pairs are grouped into columns. Each column represents a target treebank, and each line segment within that column shows the divergence equation (4) from variants 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).

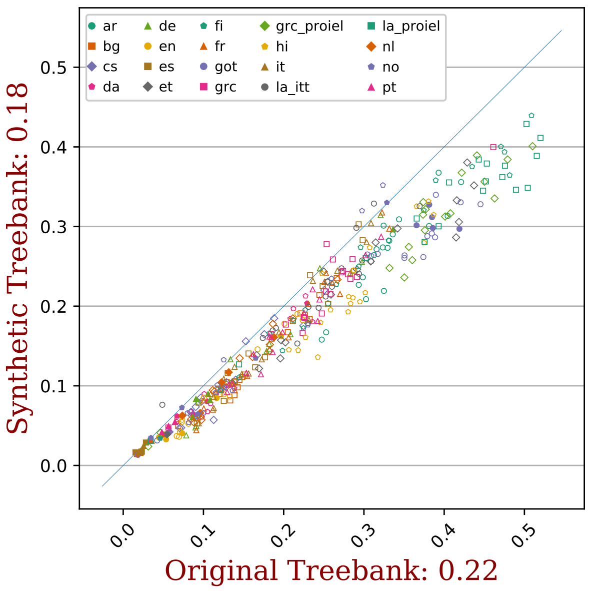


Figure 6: This graph plots the x-axes from the two graphs in Figure 1 against each other. We see that for almost every source-target pair (330/376 = 96.01% of the pairs), the SGD optimizer succeeded in constructing a permuted source treebank B' with lower divergence to the target than the original source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

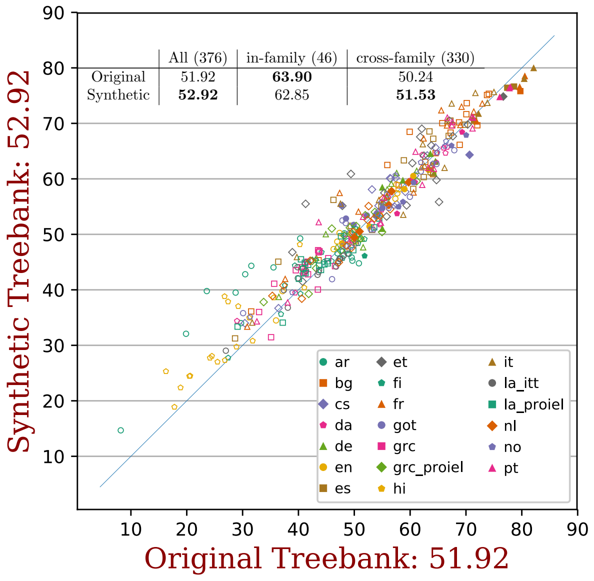
	Selection	Mix=	Mix≠
Original	**64.37**	62.31	64.77
+Synthetic	**64.55**	**62.77**	**65.00**

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**

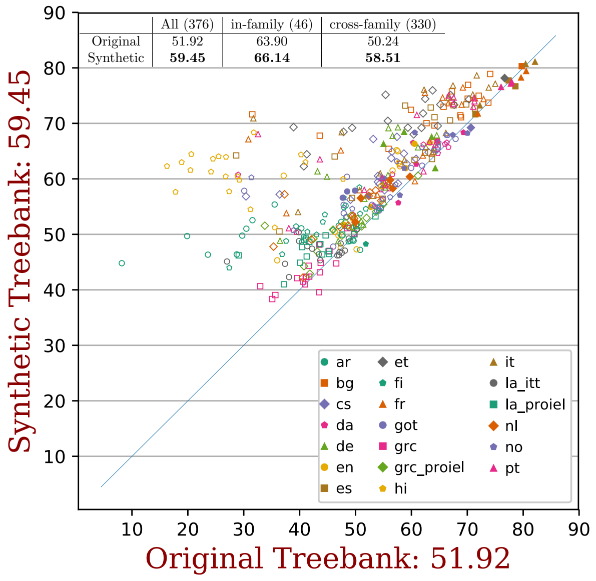
Figure 7: Unlabeled attachment scores (UAS) on 376 language pairs within the training languages. Each marker represents one pair, whose x-axis is the UAS on the target language using the original treebank of the source language, and the y-axis is the UAS using the synthetic treebank permuted from the 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.

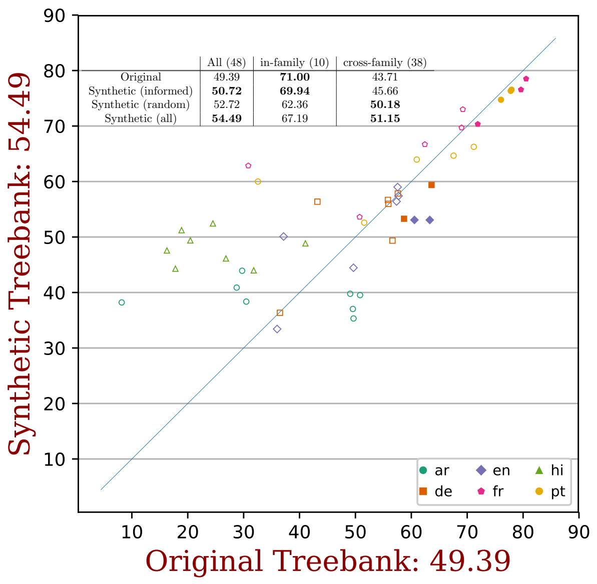


	All (376)	in-family (46)	cross-family (330)
Original	51.92	63.90	50.24
Synthetic	**59.45**	**66.14**	**58.51**

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.