Exploiting Dynamic Oracles to Train Projective Dependency Parsers on Non-Projective Trees

Lauriane Aufrant[1,2], **Guillaume Wisniewski**[1] & **François Yvon**[1]

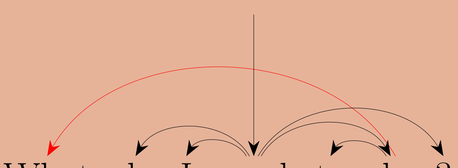
[1]LIMSI-CNRS, Univ. Paris-Sud, Université Paris-Saclay, France [2]DGA, France

first.last@limsi.fr

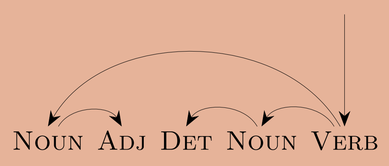
Why using our method?

- ✓ Dynamic oracles are always beneficial and becoming standard
- ✓ More data is always better
- ✓ Only one line of code to change

crossing edges ⟹ non-projectivity



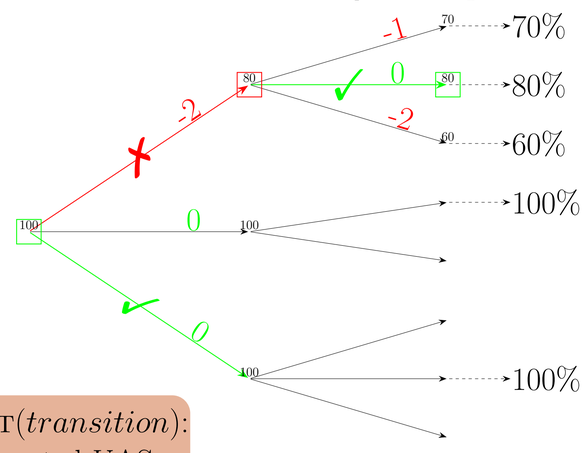
Transition-based dependency parsing



Transitions operating on a stack and a buffer:
e.g. Shift, Left, Right, Reduce

▶ **only projective outputs = better efficiency**

Dynamic oracle [Goldberg & Nivre, 2012]



Cost(*transition*):
Δ expected UAS

Train with **tailored references**:
[*instead of an arbitrary precomputed sequence*]

gold actions = zero-cost actions

For **arc-decomposable** systems:
[*i.e. from any configuration, a tree containing all reachable arcs can be built*]

cost = # newly unreachable arcs

▶ e.g. ArcEager, ArcHybrid...

▶ not ArcStandard: cf *The* ↶ *bag* ↶ *fell*

mandatory for static oracles,
suboptimal for dynamic ones

sound framework but
hard to apply

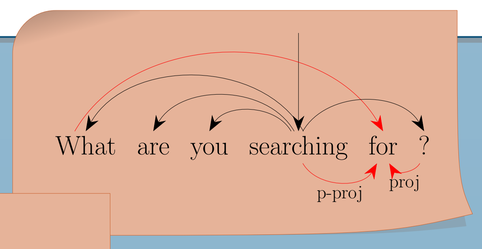
Standard strategies

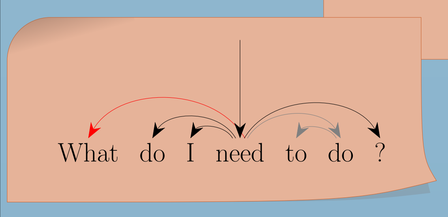
▶ Discard

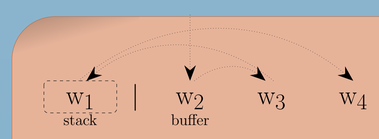
↪ throw away 7k (Ancient Greek), 2.5k (Dutch), 2k (German), 8k sentences (Czech)...

▶ Projectivize (Eisner's decoder, pseudo-projectivization)

↪ arbitrary dependencies, annotation inconsistencies







Projective cost:
Left=**2**, Right=**1**, Shift=**0**

Projective parsers & Non-projective examples

↪ How to train on trees out of the output space?

Proposal: cost approximation

↪ ignore incompatibilities (use projective cost)

↪ gold actions = minimum-cost actions

✓ always applicable

✗ bias toward late resolution of incompatibilities

With dynamic oracles

▶ Theoretically sound updates

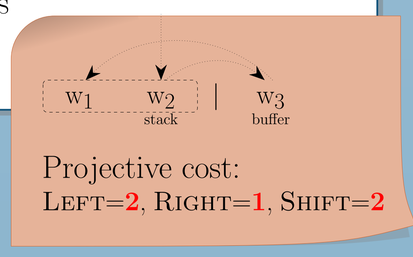
↪ zero-cost actions exist by definition

▶ Interpretation as a non-arc-decomposable case

cost = projective cost ± arc incompatibilities

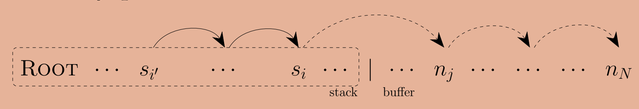
▶ Enumerate incompatibilities

↪ tedious, ad hoc analysis



Projective cost:
Left=**2**, Right=**1**, Shift=**2**

Identify patterns like:



Experiments on UD 2.0 (PanParser, ArcEager)

UAS	μ	%non-proj snt.		#training snt.	
		> 50%	< 10%	> 500	< 500
Greedy dynamic oracle (proj. snt.)	78.94	57.75	82.97	81.92	68.34
+ non-projective snt.	79.43	60.18	83.04	82.44	68.70
Global dynamic oracle (proj. snt.)	79.77	57.12	83.96	83.12	67.84
+ non-projective snt.	**80.40**	61.49	**84.04**	83.63	**68.87**
UDPipe (proj. & non-proj. parsers)	79.47	**66.99**	83.45	**83.67**	64.48

▶ Overall gain: **+0.48** UAS [greedy], **+0.63** UAS [beam]

▶ Ancient Greek (63% non-proj.): **+2.43** UAS [greedy], **+4.38** UAS [beam]

▶ Up to **+7 UAS** (Dutch-LassySmall, 30% non-proj.)

▶ **More reliable** than rewriting for small treebanks

↪ deprojectivizing the outputs does not help much

▶ **Outperforms a non-pretrained UDPipe** with non-projective outputs

↪ catches up on large treebanks, even better on small ones

NAACL'18 – North American Chapter of the Association for Computational Linguistics