Non-Projective Dependency Parsing with PCFG

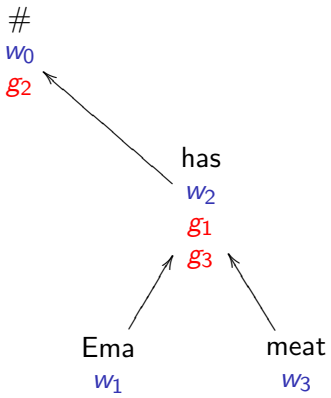
Václav Novák, Keith Hall

ÚFAL, Charles University, Prague
CLSP, Johns Hopkins University, Baltimore, MD, USA

17th November, 2005

1. Introduction
   - Definitions
   - PCFG Parser
2. Projective Solutions
   - Collins
   - Charniak
3. Non-Projective Solutions
   - Overview
   - Before Parsing
   - While Parsing
   - After Parsing
4. Output Repairs
   - Model
   - Implementation
   - Results
   - Conclusion

Parsing



Parsing Task

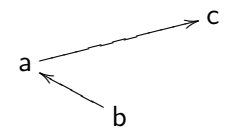
- Input: sentence $\vec{W} = w_0, w_1, \ldots, w_n$
- Output: tree $\vec{G} = g_1, g_2, \ldots, g_n$

Evaluation

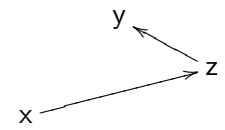
- Thanks to PDT we have $\vec{G}^r$ – the real tree
- Parser accuracy:

$$\frac{\sum_{i=1}^{n} \delta(g_i = g_i^r)}{n}$$

Non-Projectivity



Tree 1



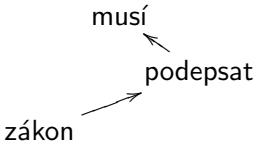
Tree 2

- Tree is projective, iff all edges are projective
- Edge $w_i \rightarrow g_i$ is projective, iff all nodes between* w_i and g_i are in the subtree of g_i.
- If $w_i \rightarrow g_i$ is non-projective, we call w_i a non-projective node
- In PDT there are 2% of non-projective nodes, over 20% of sentences [Zeman 2004]
- *with respect to the surface word order

PCFG Parser

Musí podepsat zákon (he_must sign law)

- $VP \rightarrow Vaux\ VP$
- $VP \rightarrow V\ NP$



Zákon musí podepsat (law he_must sign)

- $VP \rightarrow Vaux\ VP$
- $VP \rightarrow NP$??? V

Projectivization

- Projectivization is reordering of the surface order such that the resulting tree is projective.
- Canonical projectivization is the projectivization that doesn't change the siblings' order and parent-child order.
- There is always a unique canonical projectivization [Havelka 2005].
- Collins' solution: Train the parser on the projectivized trees.
- Advantage: It trains the right dependencies
- Disadvantage: The parser is trained on another language

Head-Lifting

(I)
- Change of parents for non-projective nodes

  Replace a non-projective edge $w_i \rightarrow g_i$ by $w_i \rightarrow g_{g_i}$

(II) Repeat step (I) until projective

- Advantage: It trains on the real sentences
- Disadvantage: It learns wrong dependencies (e.g., reflexive particles depending on auxiliary verbs)

Real Solution Possibilities

- Change of the parser input
- Encoding of non-projectivities into the constituency tree
- Parser output repair

Changes of Parser Input

- Let's train a function t, which for every sentence s returns its projective permutation s'
- t reorders the words to make the sentence projective

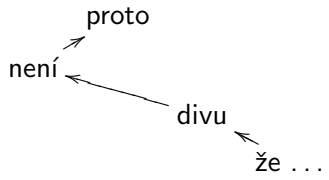
Parser Training

- For the input sentence s find $s' = t(s)$
- Use the (hopefully) projective s' to train the parser

Parsing

- For the input sentence s find $s' = t(s)$
- Run the parser on s'
- Perform the inverse permutation on the resulting tree

Training of Reordering



- It is difficult to train the shifts, the baseline: $\sim$ 94.3% – 98.7%
- The canonical projectivization isn't optimal w.r.t. the word shifts
- It's possible to reduce the shifts (to max. distance 30 from originaly 100)
- Individual shifts don't always result in a permutation ($O(60^n)$, but not a practical problem)

Experiments

Shift	Data	Data %	Succ.	REC.	Guess	PREC.
-6:	6	0.0%	0	0.00%	0	N/A
-5:	47	0.1%	0	0.00%	2	0.00%
-4:	42	0.1%	0	0.00%	0	N/A
-3:	89	0.2%	2	2.25%	7	28.57%
-2:	300	0.5%	6	2.00%	21	28.57%
-1:	1141	2.1%	161	14.11%	240	67.08%
0:	51804	94.3%	51659	99.72%	54602	94.61%
1:	1113	2.0%	114	10.24%	201	56.72%
2:	221	0.4%	37	16.74%	44	84.09%
3:	74	0.1%	0	0.00%	0	N/A
4:	44	0.1%	0	0.00%	0	N/A
5:	37	0.1%	1	2.70%	1	100.00%
6:	19	0.0%	0	0.00%	0	N/A

Overall: 94.3%

Table: The words after permutization

Offenders

- Some words are typical offenders.

Form	n-p total	Gap	Dependent	Governing	Total Occurences
li	0.98	0.92	0.00	0.06	53
však	0.91	0.91	0.00	0.00	238
ovšem	0.74	0.73	0.01	0.00	70
lze	0.72	0.68	0.00	0.05	40
sice	0.71	0.69	0.02	0.00	42
jich	0.71	0.06	0.65	0.00	17
mohlo	0.67	0.67	0.00	0.00	12
mohli	0.65	0.65	0.00	0.00	17
sami	0.61	0.11	0.50	0.00	18
nelze	0.58	0.54	0.00	0.04	26
muset	0.57	0.50	0.00	0.07	14
začali	0.55	0.55	0.00	0.00	11
můžeme	0.50	0.50	0.00	0.00	16
hodlá	0.50	0.50	0.00	0.00	14
nemusí	0.50	0.33	0.00	0.17	12
dělat	0.50	0.00	0.06	0.44	16

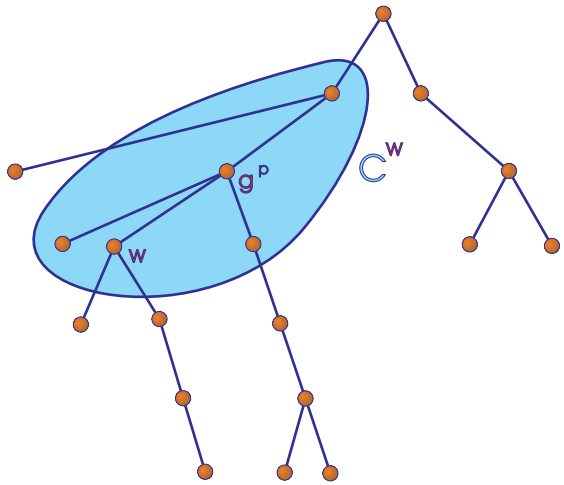
Encoding of Non-Projectivities into the Tree

- As head-lifting, but remember the changes being made and try to reconstruct.
- False parents are marked as false and the nodes on the path to the real parent are also marked.
- → Increase in number of node labels → sparser data
- [Nivre and Nilsson 2005] – 80.1% from originally 79.1% (dependency parsing)

Parser Output Correction

- Let the parser do the best
- Learn what mistakes it does
- Similar to reranking
- In case of non-projectivities we can train on the parser trainnig data
- Otherwise we can use the jackknife method [Quenouille 1956]
- Following technique described in [Hall and Novák 2005]

Output Correction – Overview



Corrective Model

- For every node w_i we have g_i^p – parent proposed by the parser
- Task: assign w_i a new parent g_i^n, so that $g_i^n = g_i^r$ (g_i^r is the real parent)
- Solution: Let's define C^{w_i}, the set of candidate nodes, that could be the real parent of w_i
- Let's define the feature function $\vec{f}(w_i, g_i^p, c)$ and let

$$g_i^n = \arg\max_{c \in C^{w_i}} p(c = g_i^r | \vec{f}(w_i, g_i^p, c))$$

- Return a new tree with parents g^n instead of g^p

Used Approximations

- Our solution again: $g_i^n = \arg\max_{c \in C^{w_i}} p(c = g_i^r | \vec{f}(w_i, g_i^p, c))$
- But in fact we want: $\vec{G}^n = \arg\max_{\vec{G} \text{ tree}} p(\vec{G} = \vec{G}^r)$
- We assume for simplicity:

Approximations

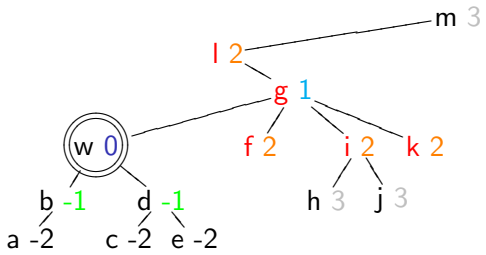
I: $p(\vec{G} = \vec{G}^r) \approx \prod_{i=1}^{n} p(g_i = g_i^r | \vec{G}^p)$

II: $\arg\max_{x \in \{1,\ldots,n\}} p(x = g_i^r | \vec{G}^p) \approx \arg\max_{c \in C^{w_i}} p(c = g_i^r | \vec{G}^p)$

III: $p(c = g_i^r | \vec{G}^p) \approx p(c = g_i^r | \vec{f}(w_i, g_i^p, c))$

How to Choose C^{w_i}?

- We want to choose C in our equation $g_i^n = \arg\max_{c \in C^{w_i}} p(c = g_i^r | \vec{f}(w_i, g_i^p, c))$
- We know the distances* between g_i^r and w_i in proposed trees:



Distance*	Freq. Charniak	Freq. Collins
less	0.5%	0.5%
-2	0.6%	0.6%
-1	2.0%	2.2%
1	84.3%	82.4%
2	8.4%	9.2%
3	2.3%	2.9%
4	0.9%	1.2%
more	0.9%	1.0%

Let $C^{w_i} = \{x | \text{distance}^*(w_i, x) = 1 \text{ to } 2\}$

Implementation

Again, we want to compute $g_i^n = \arg\max_{c \in C^{w_i}} p(c = g_i^r | \vec{f}(w_i, g_i^p, c))$

We need to:

- Design $\vec{f}(w_i, g_i^p, c)$
- Estimate p
- Make sure, that the resulting graph is a tree

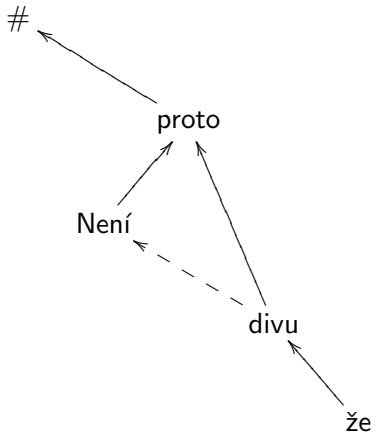
Selection of $\vec{f}(w_i, g_i^p, c)$

- all forms, lemmas, tags
- $c = g_i^p$?

} basic model

- children counts
- combinations MTag-L, MTag-F, POS-L
- combinations TTT, LLT, TLT, LLL
- "grammatical agreement"

Estimation of p

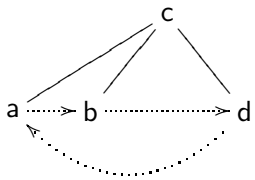


MaxEnt classifier

1	Wnení	Gproto	Cproto	pY
0	Wnení	Gproto	C#	pN
0	Wnení	Gproto	Cdivu	pN
1	Wproto	G#	C#	pY
1	Wdivu	Gproto	Cnení	pN
0	Wdivu	Gproto	Cproto	pY
...				

- After training we can for every row get $p(c = g_i^r | \vec{f}(w_i, g_i^p, c))$
- We choose the most probable candidate.

Cycles



- We repair from the bottom each level separately
- If the parent changes to grandparent, we don't reconsider the node
- However, we can still create a cycle
- Thanks to the selection of C the cycles are always among siblings
- We test for cycles, in case of cycle detection we remove the weakest edge.
- In PDT with a good $\vec{f}(w_i, g_i^p, c)$ no cycles were detected

Results

Model	Charniak		Collins	
	Accur.	NonP Accur.	Accur.	NonP Accur.
Baseline	84.3%	15.9%	82.4%	12.0%
F,T,L	84.3%	16.0%	82.5%	12.2%
F,T,L + Count	84.3%	16.7%	82.5%	13.8%
F,T,L + MtagL	84.8%	43.5%	83.2%	44.1%
F,T,L + MtagF	84.8%	42.2%	83.2%	43.2%
F,T,L + POS	84.3%	16.0%	82.4%	12.1%
F,T,L + TTT	84.3%	16.0%	82.5%	12.2%
F,T,L + Agr	84.3%	16.2%	82.5%	12.2%
F,T,L + Trig	84.9%	47.9%	83.1%	47.7%
All	85.0%	51.9%	83.5%	57.5%

Repairs Results

	Charniak	Collins
Correct → Incorrect	13.0%	20.0%
Incorrect → Incorrect	21.6%	25.8%
Incorrect → Correct	65.5%	54.1%

Best and Worst Words		
	Best	Worst
Repaired children	se i si až jen	, se na že -
Repaired proposed parents	v však li na o	a , však musí li
Repaired new parents	a je stát ba ,	# sklo , je -

Summary and Future Work

Summary

- Method not limited to a particular parser
- Should not decrease the accuracy (if the data is consistent)
- 4% relative error reduction; non-proj. nodes: over 50% correct

Future Work

- Softening of Approximation I:
  $p(\vec{G} = \vec{G}^r) \approx \prod_{i=1}^{n} p(g_i = g_i^r | \vec{G}^p)$ $\rightarrow$ global features
  (non-projectivities, linguistic features)
- More precise C^{w_i} selection
- Better $\vec{f}(w_i, g_i^p, c)$
- Try to repair MST parser's output, where reranking is difficult

Reference

Hall, Keith and Václav Novák
Corrective Modeling for Non-Projective Dependency Parsing
In *Proceedings of IWPT, Vancouver*, 42–52, 2005.

Havelka, Jiří
An Alternative Definition of Projectivity and Optimal Algorithms for Detecting Non-Projective Edges and Projectivizing Totally Ordered Rooted Trees
In *Prague Bulletin of Mathematical Linguistics*, 84, 2005

Nivre, Joakim and Jens Nilsson
Pseudo-Projective Dependency Parsing
In *Proceedings of ACL, Ann Arbor*, 99–106, 2005.

Quenouille, M.H.
Notes on bias in estimation
In *Biometrika*, 61, 353-360, 1956.

Zeman, Daniel
Neprojektivity v Pražském závislostním korpusu (PDT)
CKL/ÚFAL Technical Report, 2004-22, Univerzita Karlova, Praha, 2004.