Non-Deterministic Oracles for Unrestricted Non-Projective Transition-Based Dependency Parsing

Anders Björkelund and Joakim Nivre

anders@ims.uni-stuttgart.de

July 27, 2015

Table of Contents

Motivation

- Recent progress in **greedy** transition-based dependency parsing using *dynamic oracles*
  - Statistical model trained to select the *next best transition*, after making a local mistake

Motivation

- Recent progress in **greedy** transition-based dependency parsing using *dynamic oracles*
  - Statistical model trained to select the *next best transition*, after making a local mistake
- **Search-based** transition-based parsers (beam search/DP) – trained to find optimal *sequence of transitions*

Motivation

- Recent progress in **greedy** transition-based dependency parsing using *dynamic oracles*
  - Statistical model trained to select the *next best transition*, after making a local mistake

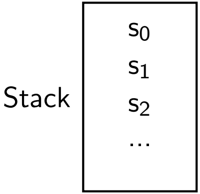
- **Search-based** transition-based parsers (beam search/DP) – trained to find optimal *sequence of transitions*
  - ? Globally trained model, dynamic oracles not entirely applicable

Motivation

- Recent progress in **greedy** transition-based dependency parsing using *dynamic oracles*
  - Statistical model trained to select the *next best transition*, after making a local mistake
- **Search-based** transition-based parsers (beam search/DP) – trained to find optimal *sequence of transitions*
  - ? Globally trained model, dynamic oracles not entirely applicable
- Can spurious ambiguity be exploited to increase accuracy of search-based parsers?

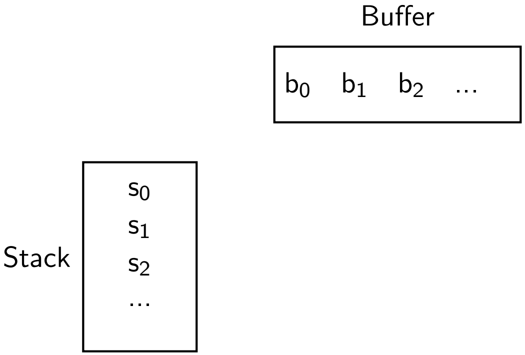
Arc Standard system

- **Stack** of partially processed tokens



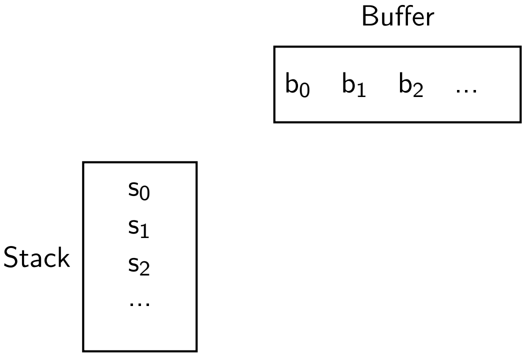
Arc Standard system

- **Stack** of partially processed tokens
- **Buffer** of remaining input tokens



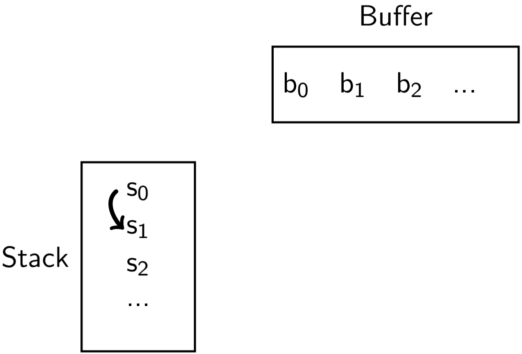
Arc Standard system

- **Stack** of partially processed tokens
- **Buffer** of remaining input tokens
- Transitions:
  - Shift (SH)



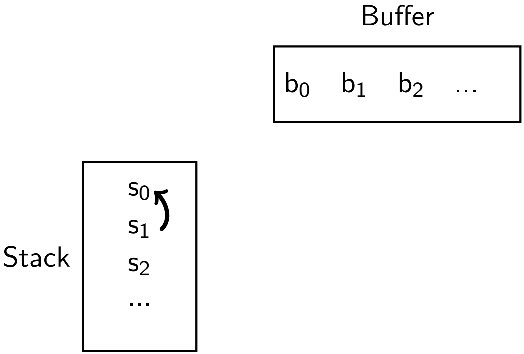
Arc Standard system

- **Stack** of partially processed tokens
- **Buffer** of remaining input tokens
- Transitions:
  - Shift (SH)
  - LeftArc (LA)



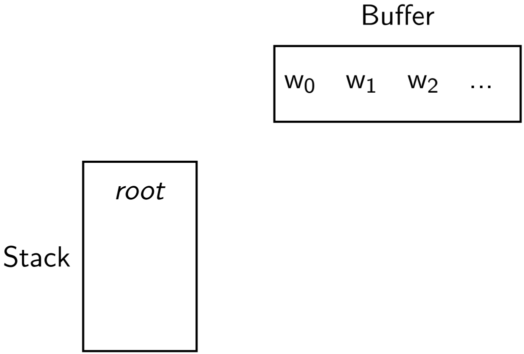
Arc Standard system

- **Stack** of partially processed tokens
- **Buffer** of remaining input tokens
- Transitions:
  - Shift (`SH`)
  - LeftArc (`LA`)
  - RightArc (`RA`)



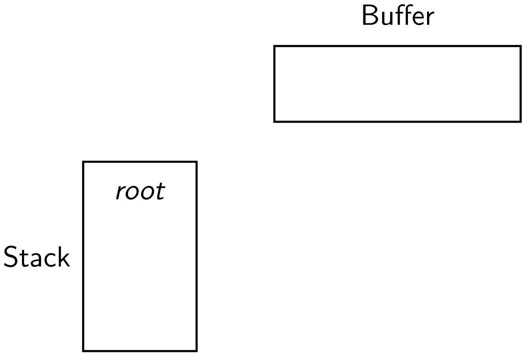
Initial and Terminal states

- **Initial state** – root on stack, input on buffer

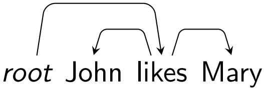


Initial and Terminal states

- **Initial state** – root on stack, input on buffer
- **Terminal state** - only root on stack, empty buffer



Example



Example parse

root John likes Mary

Buffer

John likes Mary

Stack

root

History:

Example parse

root John likes Mary

Buffer

likes Mary

John
root

Stack

History: `SH`

Example parse

root John likes Mary

Buffer

Mary

Stack

likes
John
root

History: `SH SH`

Example parse

root John likes Mary

Buffer

Mary

Stack

likes

root

History: `SH SH LA`

Example parse

root John likes Mary

Buffer

Stack

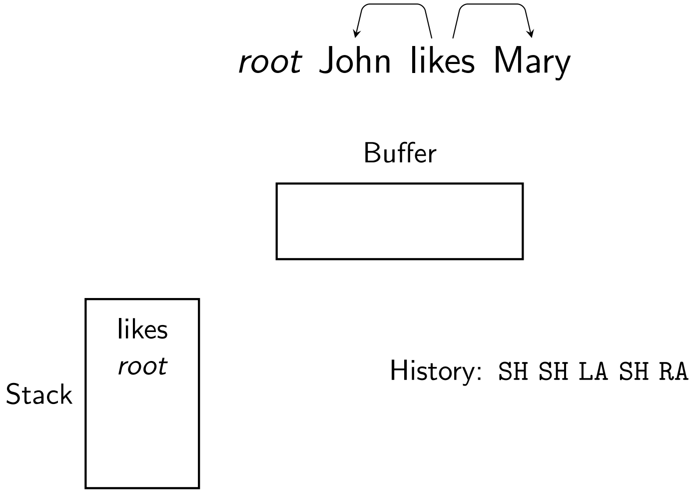
Mary

likes

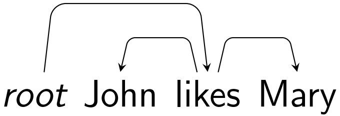
root

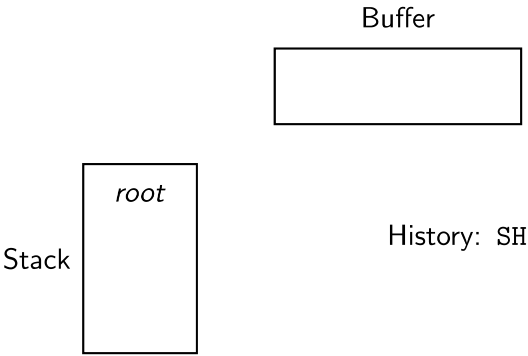
History: `SH SH LA SH`

Example parse



Example parse





History: `SH SH LA SH RA RA`

Example parse

root John likes Mary

Buffer

John likes Mary

Stack

root

History: SH SH LA SH RA RA

History:

Example parse

root John likes Mary

Buffer

likes Mary

Stack

John
root

History: `SH SH LA SH RA RA`

History: `SH`

Example parse

root John likes Mary

Buffer

Mary

Stack

likes
John
root

History: `SH SH LA SH RA RA`

History: `SH SH`

Example parse

root John likes Mary

Buffer

Stack

Mary
likes
John
root

History: `SH SH LA SH RA RA`

History: `SH SH SH`

Example parse

root John likes Mary

Buffer

Stack

likes
John
root

History: `SH SH LA SH RA RA`

History: `SH SH SH RA`

Example parse

root John likes Mary

Buffer

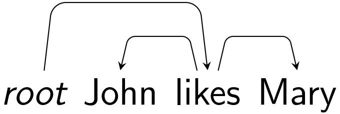
Stack

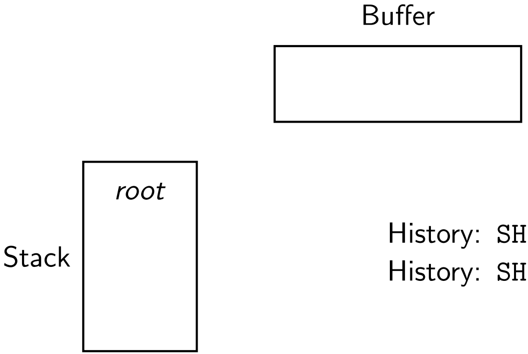
likes
root

History: `SH SH LA SH RA RA`

History: `SH SH SH RA LA`

Example parse

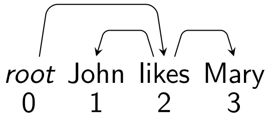




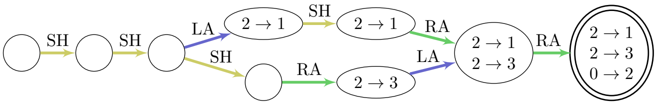
History: `SH SH LA SH RA RA`

History: `SH SH SH RA LA RA`

Ambiguity as a lattice

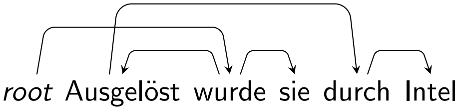


The possible transition sequences can be illustrated as a lattice



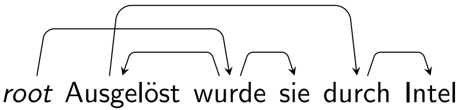
The SH-LA ambiguity a *spurious ambiguity*

Dealing with non-projectivity



- Non-projective trees cannot be drawn without crossing edges
- Treatment: introduce new transition swap (`SW`) that moves the second stack item back onto the buffer (Nivre, 2009)
- Increases the amount of spurious ambiguity considerably

Lattice for non-projective sentence



Corresponding lattice

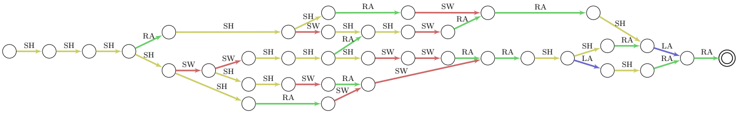


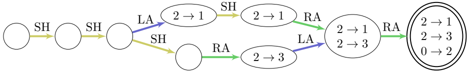
Table of Contents

Static oracle

```
if CanLA(c, x) then
    return LA
else if CanRA(c, x) then
    return RA
else
    return SH
```

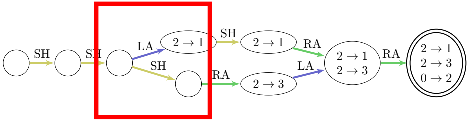
Static oracle

```
if CanLA(c, x) then
    return LA
else if CanRA(c, x) then
    return RA
else
    return SH
```



Static oracle

if $\textsc{CanLA}(c, x)$ **then**
    return `LA`
else if $\textsc{CanRA}(c, x)$ **then**
    return `RA`
else
    return `SH`



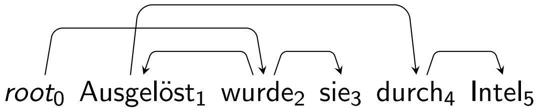
- Spurious ambiguity resolved by order of if-clauses

Static oracle (with Swap)

if $\textsc{CanLA}(c, x)$ **then**
    return `LA`
else if $\textsc{CanRA}(c, x)$ **then**
    return `RA`
else if $\textsc{CanSW}(c, x)$ **then**
        return `SW`
else
    return `SH`

CanSwap

- Relies on the notion of projective order, obtained by in-order traversal



CanSwap

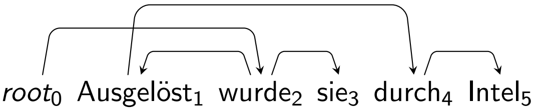
- Relies on the notion of projective order, obtained by in-order traversal

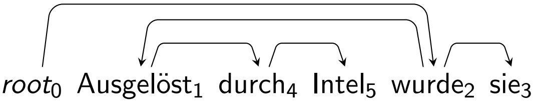
$root_0$ Ausgelöst$_1$ wurde$_2$ sie$_3$ durch$_4$ Intel$_5$

$root_0$ Ausgelöst$_1$ durch$_4$ Intel$_5$ wurde$_2$ sie$_3$

CanSwap

- Relies on the notion of projective order, obtained by in-order traversal





CanSwap

- Nivre (2009) swap as soon as possible (EAGER)
  - $\Rightarrow$ leads to many unneccessary swaps
- Nivre et al. (2009) block some swaps when more substructure can be built (LAZY)
  - $\Rightarrow$ still not always minimal

Potential spurious ambiguities

- Possible
  - `SH-LA`
  - `SH-RA`
  - `SH-SW`
- Impossible
  - `LA-RA` – (*implies cycle*)
  - `SW-RA` – (*violates projective order*)
  - `SW-LA` – (*violates projective order*)
  - And any superset of these

CanShift ?

- Static oracles define when `LA`, `RA`, `SW` are permissble
- `SH` treated as fallback

- Simple solution:
  try and see if the correct parse can be recovered
  using EAGER

Can now build complete lattices

- With tests for all transitions we can construct lattices
- Cover *all* possible spurious ambiguities

- Searching the lattice for the shortest path
  yields minimally swapping oracle (MINIMAL)

Non-deterministic oracles

- Allow all possible spurious ambiguities (ND-ALL)
- Allow only SH-SW ambiguity (ND-SW)

Table of Contents

Oracles

- Static
  - EAGER – (Nivre, 2009)
  - LAZY – (Nivre et al., 2009)
  - MINIMAL – *new*
- Non-deterministic
  - ND-ALL – *new*
  - ND-SW – *new*

Data and Evaluation

Data

- SPRML Shared Task: Arabic, Basque, French, German, Hebrew, Hungarian, Korean, Polish, Swedish
- English: Penn Treebank converted to Stanford dependencies
- Standard splits train/dev/test

Data and Evaluation

Data

- SPRML Shared Task: Arabic, Basque, French, German, Hebrew, Hungarian, Korean, Polish, Swedish
- English: Penn Treebank converted to Stanford dependencies
- Standard splits train/dev/test

Evaluation

- Labeled Attachment Score (LAS)
- Significance Testing: Wilcoxon signed rank test

$^{\dagger} < 0.05, \quad ^{\ddagger} < 0.01$

Data set stats (training data)

	% proj.
ar	97.32
de	67.23
en	99.90
eu	94.71
fr	99.97
he	99.82
hu	87.75
ko	100.00
pl	99.54
sv	93.62

Data set stats (training data)

	% proj.
ar	97.32
de	67.23
en	99.90
eu	94.71
fr	99.97
he	99.82
hu	87.75
ko	100.00
pl	99.54
sv	93.62

Most non-proj

Fully proj.

Wide range of projectivity: German (alot) to Korean (none)

Data set stats (training data)

	% proj.	LAZY red.
ar	97.32	80.59
de	67.23	75.09
en	99.90	71.92
eu	94.71	53.46
fr	99.97	16.67
he	99.82	8.33
hu	87.75	51.07
ko	100.00	-
pl	99.54	59.34
sv	93.62	75.90

Reduction of swaps from EAGER to LAZY

Data set stats (training data)

Biggest reduction

Heavily non-proj.

	% proj.	LAZY red.
ar	97.32	80.59
de	67.23	75.09
en	99.90	71.92
eu	94.71	53.46
fr	99.97	16.67
he	99.82	8.33
hu	87.75	51.07
ko	100.00	-
pl	99.54	59.34
sv	93.62	75.90

Reduction of swaps from EAGER to LAZY

- Reduces swaps by up to 80% (Arabic), 75% for German
- Corroborates results by Nivre et al. (2009)

Data set stats (training data)

	% proj.	LAZY red.
ar	97.32	80.59
de	67.23	75.09
en	99.90	71.92
eu	94.71	53.46
fr	99.97	16.67
he	99.82	8.33
hu	87.75	51.07
ko	100.00	-
pl	99.54	59.34
sv	93.62	75.90

Reduction of swaps from EAGER to LAZY

- Reduces swaps by up to 80% (Arabic), 75% for German
- Corroborates results by Nivre et al. (2009)
- Extremely few non-proj arcs in French and Hebrew since they are basically projective

Data set stats (training data)

	% proj.	Lazy red.	Minimal red.
ar	97.32	80.59	80.79
de	67.23	75.09	83.88
en	99.90	71.92	-
eu	94.71	53.46	-
fr	99.97	16.67	-
he	99.82	8.33	-
hu	87.75	51.07	54.24
ko	100.00	-	-
pl	99.54	59.34	-
sv	93.62	75.90	77.79

Reduction of swaps from Eager to Minimal

Data set stats (training data)

	% proj.	Lazy red.	Minimal red.
ar	97.32	80.59	80.79
de	67.23	75.09	83.88
en	99.90	71.92	-
eu	94.71	53.46	-
fr	99.97	16.67	-
he	99.82	8.33	-
hu	87.75	51.07	54.24
ko	100.00	-	-
pl	99.54	59.34	-
sv	93.62	75.90	77.79

Reduction of swaps from Eager to Minimal

- Lazy already minimal in several cases

Data set stats (training data)

	% proj.	LAZY red.	MINIMAL red.
ar	97.32	80.59	80.79
de	67.23	75.09	83.88
en	99.90	71.92	-
eu	94.71	53.46	-
fr	99.97	16.67	-
he	99.82	8.33	-
hu	87.75	51.07	54.24
ko	100.00	-	-
pl	99.54	59.34	-
sv	93.62	75.90	77.79

Reduction of swaps from EAGER to MINIMAL

- LAZY already minimal in several cases
- Reduction relative to LAZY very small

Data set stats (training data)

	% proj.	Lazy red.	Minimal red.	unique
ar	97.32	80.59	80.79	9.94
de	67.23	75.09	83.88	7.81
en	99.90	71.92	-	1.31
eu	94.71	53.46	-	1.06
fr	99.97	16.67	-	2.66
he	99.82	8.33	-	2.82
hu	87.75	51.07	54.24	10.25
ko	100.00	-	-	0.27
pl	99.54	59.34	-	10.57
sv	93.62	75.90	77.79	7.28

Amount of sentences without spurious ambiguity

Data set stats (training data)

	% proj.	LAZY red.	MINIMAL red.	unique
ar	97.32	80.59	80.79	9.94
de	67.23	75.09	83.88	7.81
en	99.90	71.92	-	1.31
eu	94.71	53.46	-	1.06
fr	99.97	16.67	-	2.66
he	99.82	8.33	-	2.82
hu	87.75	51.07	54.24	10.25
ko	100.00	-	-	0.27
pl	99.54	59.34	-	10.57
sv	93.62	75.90	77.79	7.28

Amount of sentences without spurious ambiguity

- Only 10% without spurious ambiguity

Data set stats (training data)

	% proj.	LAZY red.	MINIMAL red.	unique
ar	97.32	80.59	80.79	9.94
de	67.23	75.09	83.88	7.81
en	99.90	71.92	-	1.31
eu	94.71	53.46	-	1.06
fr	99.97	16.67	-	2.66
he	99.82	8.33	-	2.82
hu	87.75	51.07	54.24	10.25
ko	100.00	-	-	0.27
pl	99.54	59.34	-	10.57
sv	93.62	75.90	77.79	7.28

Amount of sentences without spurious ambiguity

- Only 10% without spurious ambiguity
- Despite being projective, Korean still lots of ambiguity

Training (static)

- Greedy parser
  - Averaged perceptron (Collins, 2002)
- Beam search parser
  - Passive-aggressive algorithm (Crammer et al., 2006)
  - Using max-violation updates (Huang et al., 2012)
  - Averaging (Collins, 2002)

Training (non-deterministic)

- What is the “correct” solution to update against?
- Leave it latent – let the current parameters decide

Training (non-deterministic)

- What is the "correct" solution to update against?
- Leave it latent – let the current parameters decide
- Greedy –

Training (non-deterministic)

- What is the "correct" solution to update against?
- Leave it latent – let the current parameters decide

- Greedy – next transition t latent

Training (non-deterministic)

- What is the “correct” solution to update against?
- Leave it latent – let the current parameters decide

- Greedy – next transition t latent

  Given current weights w,
  and state c

Latent gold

$$\tilde{t} = \operatorname*{arg\,max}_{t \in \textsc{Nd-Oracle}(c)} score(t, w)$$

Training (non-deterministic)

- What is the "correct" solution to update against?
- Leave it latent – let the current parameters decide

- Greedy – next transition t latent

  Given current weights w,
  and state c

Latent gold

$$\tilde{t} = \underset{t \in \textsc{Nd-Oracle}(c)}{\arg\max} \; score(t, w)$$

Prediction

$$\hat{t} = \underset{t \in \textsc{Permissible}(c)}{\arg\max} \; score(t, w)$$

Training (non-deterministic)

- What is the "correct" solution to update against?
- Leave it latent – let the current parameters decide

- Beam search

Training (non-deterministic)

- What is the "correct" solution to update against?
- Leave it latent – let the current parameters decide

- Beam search – transition sequence z latent

Training (non-deterministic)

- What is the "correct" solution to update against?
- Leave it latent – let the current parameters decide

- Beam search – transition sequence z latent

  Given current weights w,
  and sentence x

Latent Gold

$$\tilde{z} = \operatorname*{arg\,max}_{z \in \textsc{Nd-Oracle}(x)} score(z, w)$$

Training (non-deterministic)

- What is the "correct" solution to update against?
- Leave it latent – let the current parameters decide

- Beam search – transition sequence z latent

  Given current weights w,
  and sentence x

Latent Gold

$$\tilde{z} = \arg\max_{z \in \textsc{Nd-Oracle}(x)} score(z, w)$$

Prediction

$$\hat{z} = \arg\max_{z \in \textsc{Possible}(x)} score(z, w)$$

Training (non-deterministic)

- What is the "correct" solution to update against?
- Leave it latent – let the current parameters decide

- Beam search – transition sequence z latent

  Given current weights w,
  and sentence x

Latent Gold

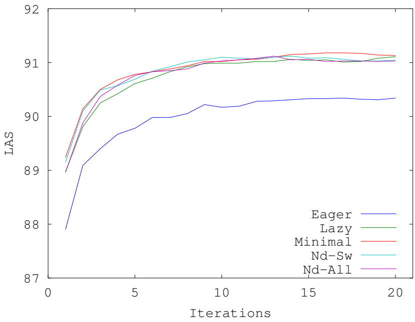
$$\tilde{z} = \underset{z \in \textsc{Nd-Oracle}(x)}{\arg\max} \; score(z, w)$$

Prediction

$$\hat{z} = \underset{z \in \textsc{Possible}(x)}{\arg\max} \; score(z, w)$$

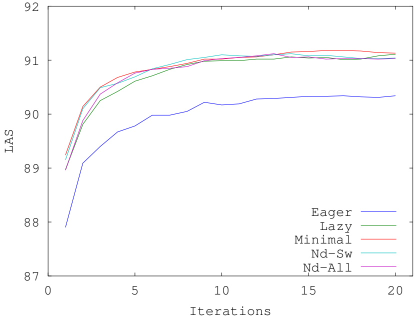
Approximate search with beam search (beam size 20)

Tuning



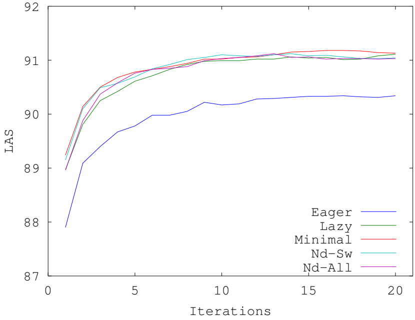
92
91
90
89
88
87
LAS
0
5
10
15
20
Iterations
Eager
Lazy
Minimal
Nd-Sw
Nd-All

Tuning



- Problem 1: Most oracles generally extremely close
- Problem 2: Performance on dev set not monotonically increasing as a function of training iterations

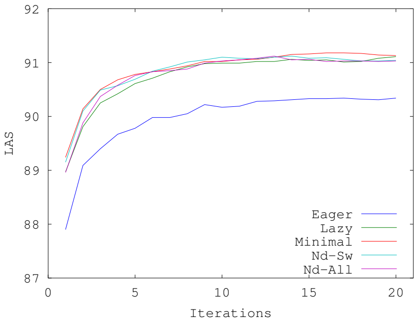
Tuning



- Problem 1: Most oracles generally extremely close
- Problem 2: Performance on dev set not monotonically increasing as a function of training iterations

- Solution: Tune number of iterations on dev data for each oracle

Tuning



- Problem 1: Most oracles generally extremely close
- Problem 2: Performance on dev set not monotonically increasing as a function of training iterations

- Solution: Tune number of iterations on dev data for each oracle

- Final evaluation (test set): best static oracle vs best non-deterministic oracle

Results – beam

	Static	Δ non-det.
ar	85.05	+0.06
de	87.53	-0.23
en	90.35	+0.13
eu	79.97	+0.55
fr	83.10	-0.11
he	78.65	-0.39
hu	83.60	+0.08
ko	85.03	+0.09
pl	82.08	+1.26‡
sv	79.05	-0.07
Macro Avg.	83.59	0.14

Results – beam

	Static	Δ non-det.
ar	85.05	+0.06
de	87.53	-0.23
en	90.35	+0.13
eu	79.97	+0.55
fr	83.10	-0.11
he	78.65	-0.39
hu	83.60	+0.08
ko	85.03	+0.09
pl	82.08	+1.26‡
sv	79.05	-0.07
Macro Avg.	83.59	0.14
Macro Avg. (w/o pl)	83.44	0.01

- Basically no difference, except Polish

Results – greedy

	Static	Δ non-det.
ar	82.99	+0.04
de	84.22	+0.03
en	87.85	+0.60‡
eu	78.58	+0.24
fr	81.12	+0.40‡
he	75.27	+0.70†
hu	81.45	+0.22
ko	84.52	+0.30
pl	79.10	+1.33‡
sv	75.89	+0.39
Macro Avg. (w/o pl)	82.39	+0.32

- Without pl. not just zero

Results – greedy

	Static	Δ non-det.
ar	82.99	+0.04
de	84.22	+0.03
en	87.85	$+0.60^{\ddagger}$
eu	78.58	+0.24
fr	81.12	$+0.40^{\ddagger}$
he	75.27	$+0.70^{\dagger}$
hu	81.45	+0.22
ko	84.52	+0.30
pl	79.10	$+1.33^{\ddagger}$
sv	75.89	+0.39
Macro Avg. (w/o pl)	82.39	+0.32
Macro Avg.	81.10	+0.43

- Without pl. not just zero
- Increases for all treebanks

Why does it only work with greedy? (speculative)

- Beam (search)
  - Search-based parsers are good at managing suboptimal local decisions (i.e., little error progapation)
  - No need to introduce additional ambiguity, search does the trick

Why does it only work with greedy? (speculative)

- Beam (search)
  - Search-based parsers are good at managing suboptimal local decisions (i.e., little error progapation)
  - No need to introduce additional ambiguity, search does the trick
- Greedy
  - Exposed to (some) more states during training, $\Rightarrow$ generalizes better
  - Never harmful

Table of Contents

Summary[1]

- Spurious ambiguity in ArcStandard+Swap

[1] Parser implementation available on my website
http://www.ims.uni-stuttgart.de/~anders/

Summary[1]

- Spurious ambiguity in ArcStandard+Swap
- Non-deterministic oracles

[1] Parser implementation available on my website
`http://www.ims.uni-stuttgart.de/~anders/`

Summary[1]

- Spurious ambiguity in ArcStandard+Swap
- Non-deterministic oracles
- Parser accuracy
  - **Beam**: No improvement

[1] Parser implementation available on my website
`http://www.ims.uni-stuttgart.de/~anders/`

Summary[1]

- Spurious ambiguity in ArcStandard+Swap
- Non-deterministic oracles
- Parser accuracy
  - **Beam**: No improvement
  - **Greedy**: Sometimes

[1] Parser implementation available on my website `http://www.ims.uni-stuttgart.de/~anders/`

Questions

Thank you.

Questions?

References I

Bohnet, B., Nivre, J., Boguslavsky, I., Farkas, R., Ginter, F., and Hajič, J. (2013). Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive–aggressive algorithms. *Journal of Machine Learning Reseach*, 7:551–585.

Huang, L., Fayong, S., and Guo, Y. (2012). Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151, Montréal, Canada. Association for Computational Linguistics.

Nivre, J. (2009). Non-projective dependency parsing in expected linear time. 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 351–359, Suntec, Singapore. Association for Computational Linguistics.

Nivre, J., Kuhlmann, M., and Hall, J. (2009). An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76, Paris, France. Association for Computational Linguistics.

Backup slide – Other ways of training (beam)

- Use early update, and update against the last correct item that fell off the beam
- Update against any gold sequence, pick the highest scoring (partial) one (may not coincide with best scoring complete sequence

- Moving target problem: across training iterations, correct sequence may change – more difficult to learn?
  - Train a model (with some oracle), apply it to the training data over the lattices and pick a single unique sequence for each sentence
  - Same as above, but do it with cross-validation (jack-knifing)

- All of these did worse than static oracle

Backup slide – Complexity of CanShift

- Theoretically $\mathcal{O}(n^2)$

Backup slide – Complexity of CanShift

- Theoretically $\mathcal{O}(n^2)$
- However, can stop if stakc gets reduced to two tokens

Backup slide – Complexity of CanShift

- Theoretically $\mathcal{O}(n^2)$
- However, can stop if stakc gets reduced to two tokens
- In practice, marginal difference on overall training time